Università degli studi di Modena e Reggio Emilia

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in Informatica

Fingerprinting tramite analisi di protocolli di rete e strategie di offuscamento

Relatore:
Luca Ferretti

Candidato:
Fabio Zanichelli

Anno Accademico 2021/2022

Indice

Capitolo 1

Introduzione

1.1 OS Fingerprinting

L'OS fingerprinting consiste nel rilevare da remoto il sistema operativo di un dispositivo analizzandone i pacchetti inviati. Le differenze di implementazione dello stack TCP/IP, infatti, determinano comportamenti diversi che, analizzati, consentono di ottenere informazioni utili a questo scopo.
Il fingerprinting può essere effettuato in due modalità: attiva e passiva.
Nella prima si analizzano le risposte ricevute in seguito ad alcuni pacchetti inviati; questi sono appositamente costruiti in modo da massimizzare le informazioni che si possono ottenere dalla risposta. Nella seconda, invece, viene ispezionato il normale traffico del dispositivo target; si tratta quindi di una tecnica meno invasiva e che si espone meno al rischio di essere scoperti.

1.2 Obiettivo

L'obiettivo consiste nell'analizzare le differenze che portano all'individuazione del sistema operativo, e successivamente modificare determinati parametri di quest'ultimo in modo da riuscire ad ingannare i principali strumenti per il fingereprinting. I risultati ottenuti, quindi, dovranno essere errati e portare all'individuazione di un sistema operativo differente rispetto a quello realmente in uso.

Si è proceduto utilizzando un server HTTP, gestendo quindi pacchetti di non cifrati; si è successivamente cercato di effettuare un'analisi sull'handshake TLS, ovvero sullo scambio di messaggi che precede una comunicazione cifrata.

1.3 Strumenti e sistemi operativi utilizzati

Per la realizzazione dell'obiettivo sono stati utilizzati due differenti sistemi operativi: Windows 11 e Kali (una distribuzione Linux basata su Debian). La motivazione della scelta di questi sistemi risiede nel fatto che Kali sia stato progettato per la sicurezza informatica, e Windows sia il sistema operativo attualmente più utilizzato; per questa ragione, l'individuazione di quest'ultimo (seppur falsificata) desterebbe meno sospetti.

I tool utilizzati sono stati i seguenti:

- **Nmap**: principale strumento per effettuare fingerprinting attivo tramite l'invio di specifici pacchetti (*probe*) costruiti appositamente per massimizzare le differenze tra i comportamenti dei sistemi operativi. Consente inoltre di effettuare altre operazioni, alcune delle quali fondamentali per il fingerprinting stesso, come ad esempio il port scanning.
- **p0f**: tool per effettuare fingerprinting passivo. Esso analizza solamente i pacchetti ricevuti da una determinata interfaccia o analizza quelli passati tramite un file con estensione pcap. È anche in grado di effettuare fingerprinting a livello 7.
- **Wireshark**: si tratta di uno strumento che permette la visualizzazione dei pacchetti inviati e ricevuti dal dispositivo tramite un'interfaccia grafica. Consente inoltre di filtrare pacchetti sulla base di determinati campi o protocolli utilizzati.
- **Server Apache**: Web server che consente di rispondere alle richieste di tipo HTTP/HTTPS. È stato installato sia su Windows 11 che su Kali per poter effettuare il confronto tra le risposte inviate.
- **Scapy**: libreria Python in grado di inviare pacchetti modificabili in ogni campo. Molto utile il suo utilizzo per quanto riguarda l'invio di pacchetti "patologici" che stimolano risposte utili ai fini del fingerprinting.

- **nftables**: tool per che permette la modifica o il blocco di pacchetti sulla base del loro contenuto negli header.

Capitolo 2

Teoria

2.1 Stack TCP/IP

Per effettuare comunicazioni tramite internet, vi è il bisogno che tutti i dispositivi connessi rispettino determinati meccanismi; questo si rende necessario a causa dell'elevata eterogeneità derivata da hardware e software differenti. Questi meccanismi, che prendono il nome di *protocolli*, sono strutturati secondo diversi layer (livelli) formando lo stack TCP/IP.
Sebbene l'idea originale prevedesse un modello composto da sette livelli, de facto lo schema attualmente in uso ne prevede solamente quattro. Nonostante ciò, nella terminologia informatica la numerazione dei livelli è rimasta quella precedente.

Livello 7	Applicativo
Livello 4	Trasporto
Livello 3	Rete
Livello 2	Fisico

Tabella 2.1: Livelli dello stack TCP/IP

2.1.1 Funzionamento dello stack TCP/IP

Il meccanismo dello stack prevede che ad ogni livello vengano aggiunti al messaggio delle intestazioni (header), che verranno valutate dal rispettivo livello del ricevente. Si precisa che l'ordine in cui si valutano gli header dei livelli è inverso rispetto a quello del mittente; chi invia partirà dal livello più alto, mentre chi riceve dal più basso. Questa procedura prende il nome di *incapsulamento* ed è riassunta nella seguente figura:

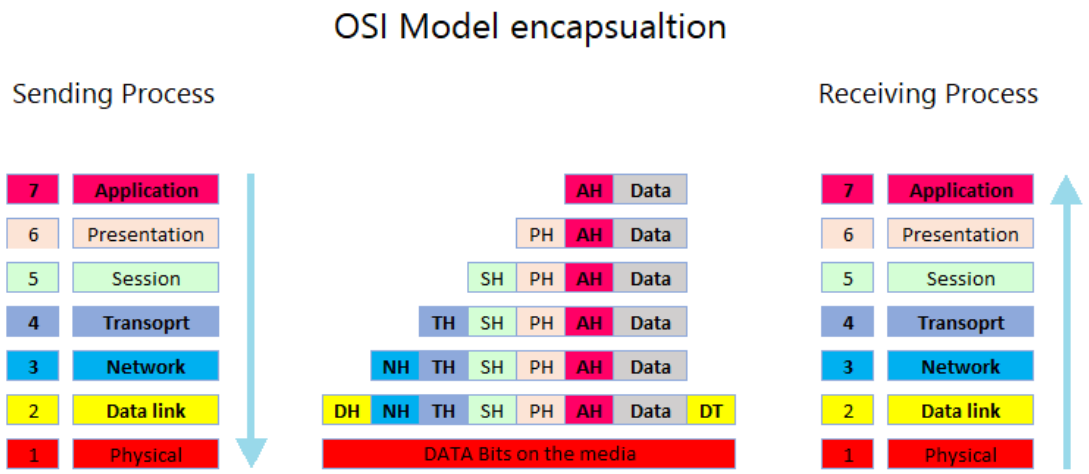

Figura 2.1: METTERE RIFERIMENTO IMMAGINE: http://infodoc.altervista.org/sistemi-e-reti/incapsulamento/

2.1.2 Header per il fingerprinting

Gli header aggiunti ad ogni livello sono formati da vari campi contententi informazioni utili per la comunicazione, e il valore che questi assumono in determinate situazioni è dipendente dal sistema operativo che si sta utilizzando.

Si prenda ad esempio l'header del protocollo TCP:

TCP Header				
Bits	0-15			16-31
0	Source port			Destination port
32	Sequence number			
64	Acknowledgment number			
96	Offset	Reserved	Flags	Window size
128	Checksum			Urgent pointer
160	Options			

Figura 2.2: METTERE RIFERIMENTO IMMAGINE: https://www.ionos.it/digitalguide/server/know-how/presentazione-tcp/

Il campo *option* permette di segnalare al ricevente l'uso di alcune opzioni di comunicazione, come il window scaling; il loro supporto e l'effettivo utilizzo, essendo queste facoltative e quindi peculiari di specifici sistemi operativi, rivestono quindi particolare importanza ai fini del fingerprinting. Esempi analoghi si possono trovare nei protocolli di tutti i livelli dello stack, e l'unione delle informazioni acquisite consente di poter individuare con una discreta precisione il sistema operativo del dispositivo target.

Capitolo 3

Analisi

L'attività di analisi iniziale, ovvero senza l'ausilio di librerie specifiche per l'invio di determinati pacchetti, è stata effettuata installando un Server Apache sia su Windows 11 che su sistema operativo Kali. Si è quindi proceduto ad inviare richieste ai server tramite browser web e all'analisi dei pacchetti ricevuti in risposta con l'utilizzo di Wireshark. Successivamente sono state analizzate anche le risposte a determinati pacchetti inviati con Scapy. Di seguito sono riportate le differenze più importanti rilevate durante l'intera attività di analisi.

3.1 Analisi a livello 2

3.2 Livello 3: protocolli IP e ICMP

È stata effettuata in primis l'analisi del TTL (*Time To Live*), essendo un valore estremamente semplice da analizzare. Questo campo contiene un numero intero positivo che viene decrementato ad ogni *hop*, ovvero ad ogni router che il pacchetto incontra nel percorso verso l'host ricevente, e viene eliminato dalla rete quando questo raggiunge lo 0. Il protocollo, però, non impone nessun valore di partenza, lasciando libertà di scelta al sistema operativo del mittente; l'unico limite è rappresentato dagli 8 bit riservati a quel campo (ovvero ad un valore massimo di 255). L'analisi ha portato ad evidenziare la seguente differenza riguardo al TTL:

Windows 11	128
Kali	7

Tabella 3.1: TTL iniziale

È inoltre stata effettuata l'analisi del protocollo ICMP, e in particolare è stata notata una differenza nel campo *code* in caso di risposta a determinati pacchetti inviati utilizzando la libreria di Python scapy.

```
1  from scapy.all import *
2
3
4  pkt = sr1(IP(dst='192.168.63.1')/ICMP(code=14))
5  pkt.show()
6
```

Figura 3.1: Comando python per l'invio del pacchetto specifico

Questo comando invia un pacchetto ICMP con il campo *code* impostato ad un numero casuale, in questo caso 14: in questo specifico contesto, Windows 11 risponde inviando un pacchetto in cui *code*=0, mentre Kali copia il valore che ha ricevuto nella richiesta.

Windows 11	0
Kali	Stesso valore inviato nella richiesta

Tabella 3.2: Campo code quando nella richiesta è diverso da 0

3.3 Livello 4: analisi protocollo TCP

Il livello 4 dello stack è formato da due protocolli: User Datagram Protocol (UDP) e Transmission Control Protocol (TCP). Entrambi forniscono informazioni utili per il fingerprinting, ma il TCP possiede un numero maggiore di campi e di opzioni utilizzabili, pertanto è stata data priorità all'analisi di quest'ultimo.

Analizzando l'handshake TCP, si può notare una discrepanza nell'utilizzo del Window Scale; quest'opzione consente di aumentare la Window Size oltre il valore che si otterrebbe settando tutti i bit di quel campo a 1. Ciò è dovuto al fatto che il valore dell'opzione rappresenta il numero di shift verso sinistra dei bit del Window Size, che corrispondono a raddoppi del valore contenuto. L'utilizzo di questa opzione viene concordato nei segmenti SYN e SYN+ACK dell'handshake e il suo valore non viene più modificato per il resto della connessione. Confrontando i valori ottenuti da Windows 11 e Kali, si ottiene il seguente risultato:

Windows 11	8
Kali	7

Tabella 3.3: Window Scaling Factor

Proseguendo l'analisi si possono notare ulteriori differenze riguardanti il flag sulla notifica esplicita di congestione (ECN). Si tratta di un flag che consente di comunicare a livello end to end una congestione, evitando però la perdita dei pacchetti. Il mittente, infatti, se riceve un pacchetto contenente quest'informazione diminuisce i pacchetti inviati in quella comunicazione, seguendo specifici algoritmi (ad esempio, Reno). Se si invia un pacchetto simulando l'inizio di un handshake (SYN) con i flag sulla congestione attivi e si osserva la risposta (SYN+ACK), si possono notare risultati differenti tra i pacchetti ricevuti da Windows 11 e Kali:

```
1  from scapy.all import *
2
3  pkt=sr1(IP(dst='192.168.63.1')/TCP(dport=80, flags='SCE'))
4  pkt.show()
```

Figura 3.2: Comando python per l'invio del pacchetto specifico

Windows 11	0
Kali	1

Tabella 3.4: ECN in risposta a specifico pacchetto

3.4 Livello 7: analisi protocollo HTTP

Sebbene HTTP sia un protocollo a livello applicativo, esso consente ugualmente di ricavare alcune informazioni utili per l'OS fingerprinting: il campo *user-agent*, infatti, contiene informazioni esplicite riguardanti il browser che si sta utilizzando e il sistema operativo in utilizzo. Questo tipo di situazione prende il nome di *banner grabbing*.

A questo livello dello stack, si può inoltre tentare un fingerprinting che vada oltre l'individuazione del sistema operativo, ponendo come obiettivo quello di indovinare il tipo di applicativo in uso. Questo è reso possibile dal fatto che HTTP è un protocollo di tipo testuale, pertanto non vi sono campi prefissati per ogni funzione. Ogni header contiene varie coppie, secondo lo schema:
chiave:valore

Questo, a differenza dei protocolli analizzati precedentemente, permette un diverso ordine con le quali le coppie vengono elencate, essendo questo ininfluente per una corretta comunicazione. Analizzare l'ordinamento è molto importante se si vuole tentare di individuare, ad esempio, il tipo di server che sta rispondendo.

Capitolo 4

Offuscamento

Capitolo 5

Analisi handshake TLS

I livelli analizzati in precedenza consentono una corretta comunicazione, ignorando però aspetti importanti quali confidenzialità, integrità ed autenticità; vi è quindi il bisogno di estendere lo stack introducendo un nuovo layer, facoltativo, in grado di garantire queste caratteristiche. Questo livello viene posizionato tra quello di trasporto e quello applicativo.

Livello 7	Applicativo
Livello 5	Sicurezza
Livello 4	Trasporto
Livello 3	Rete
Livello 2	Fisico

Tabella 5.1: Livelli dello stack con layer di sicurezza

Per garantire la confidenzialità dei dati in rete, ovvero la loro segretezza, si fa uso della crittografia ovvero una procedura che modifica i pacchetti in modo da renderli incomprensibili ad attori esterni alla comunicazione. Questa avviene solitamente in forma simmetrica, ovvero la chiave per cifrare i dati è la medesima utilizzata per decifrarli.

5.1 Handshake TLS

La presenza di una chiave comune ad entrambi i partecipanti della comunicazione pone il problema di come questi ne vengano a conoscenza, non potendosi effettuare lo scambio di chiavi sul medesimo canale delle comunicazioni ordinarie in quanto insicuro. Oltre alla confidenzialità della chiave, si ha inoltre il problema dovuto all'autenticità della stessa; un eventuale attaccante potrebbe infatti intromettersi nella comunicazione prendendo il posto di uno dei due host, all'insaputa dell'altro (attacco Man In The Middle, MITM).

TLS è un protocollo che prevede uno scambio di messaggi precedente all'invio dei dati veri e propri (handshake) che permette una successiva comunicazione sicura. Si precisa che i pacchetti inviati a questo scopo non sono cifrati, non essendoci ancora una chiave di cifratura in possesso degli host. Solo al termine di questa procedura si saranno scambiate correttamente le chiavi.

Il primo messaggio è inviato dal client al server e prende il nome di *client hello*. In questo messaggio vengono inviate informazioni quali cifrari supportati (in ordine di preferenza), versione TLS in uso, ed estensioni che specificano altri aspetti come ad esempio il supporto alle curve ellittiche o a determinati algoritmi di hashing per il controllo dell'autenticità.

Il server risponde indicando il cifrario da utilizzare, le estensioni da lui supportate, il suo certificato (per garantire l'autenticità della risposta) oltre ad iniziare lo scambio di chiavi, effettuato sfruttando meccanismi matematici in grado di garantire la confidenzialità di quest'ultima anche in caso di MITM passivo. A questo punto il client termina lo scambio di chiavi.

5.2 Browser fingerprinting

L'analisi delle differenze nel contenuto dei pacchetti client hello inviati da differenti client permette di notare differenze peculiari di quest'ultimi, consentendone la sua individuazione; in questo documento saranno analizzati i browser più popolari.

I browser analizzati sono stati i seguenti:

- Chrome versione 105.0.5195.127
- Mozilla Firefox versione 105.0
- Microsoft Edge versione 105.0.1343.42
- Opera versione 91.0.4516.20

Non è stato preso in considerazione Safari, in quanto non vi è una versione aggiornata per Windows.

Nel processo di analisi sono stati ispezionati i cifrari utilizzati, l'utilizzo del GREASE, gli algoritmi di hashing supportati e altre estensioni come ad esempio il supporto alle curve ellittiche.

Esaminando i cifrari supportati, si può notare la loro differenza in numero:
Questo rende Edge e Firefox differenti rispetto agli altri due, in quanto supportano

Chrome	16
Firefox	17
Edge	17
Opera	16

Tabella 5.2: Cifrari supportati dai vari browser

un numero di cifrari diverso. Firefox, però, presenta anche altre differenze rispetto agli altri come per esempio il supporto ad un maggior numero di algoritmi di hashing e la non presenza del GREASE nei cifrari. Per quanto riguarda il primo valore, Firefox è l'unico a presentare un numero diverso rispetto agli altri:

Chrome	8
Firefox	11
Edge	8
Opera	8

Tabella 5.3: Algoritmi di hashing supportati

Il GREASE, acroninimo di Generate Random Extensions And Sustain Extensibility, è un valore che corrisponde ad un cifrario inesistente nella realtà e che quindi il ricevente deve ignorare; il protocollo prevede infatti che se il server non supporta un cifrario nella lista questo vada ignorando, valutando quello successivo. Si tratta perciò di un test utile per prevenire bug. È quindi logico che il GREASE venga posizionato in cima alla lista dei cifrari, in modo che sia sempre valutata dal server.
Anche in questo caso, Firefox è l'unico browser a comportarsi differentemente rispetto agli altri:

Chrome	Sì
Firefox	No
Edge	Sì
Opera	Sì

Tabella 5.4: Presenza del GREASE nella lista dei cifrari supportati

Analizzando nel dettaglio il client hello inviato da Edge, si può notare una caratteristica piuttosto peculiare. Vi è infatti un cifrario, per l'esattezza il **AES 256 GCM SHA384**, che viene ripetuto due volte all'interno della lista dei cifrari. Questo è ovviamente inutile, in quanto se il server rifiuta quest'ultimo poi procederebbe a valutarlo nuovamente, producendo quindi un nuovo rifiuto. Si tratta dell'unico browser dei quattro che ha questo comportamento, che lo rende molto vulnerabile al fingerprinting. Di seguito lo screenshot di Wireshark che mostra questa caratteristica.

Questo causa una differenza di comportamento tra Chrome e Firefox, agevolando il loro riconoscimento. Eliminare il cifrario ripetuto porterebbe Edge ad una situazione uguale a quella di Chrome e Opera, rendendoli quindi indistinguibili.

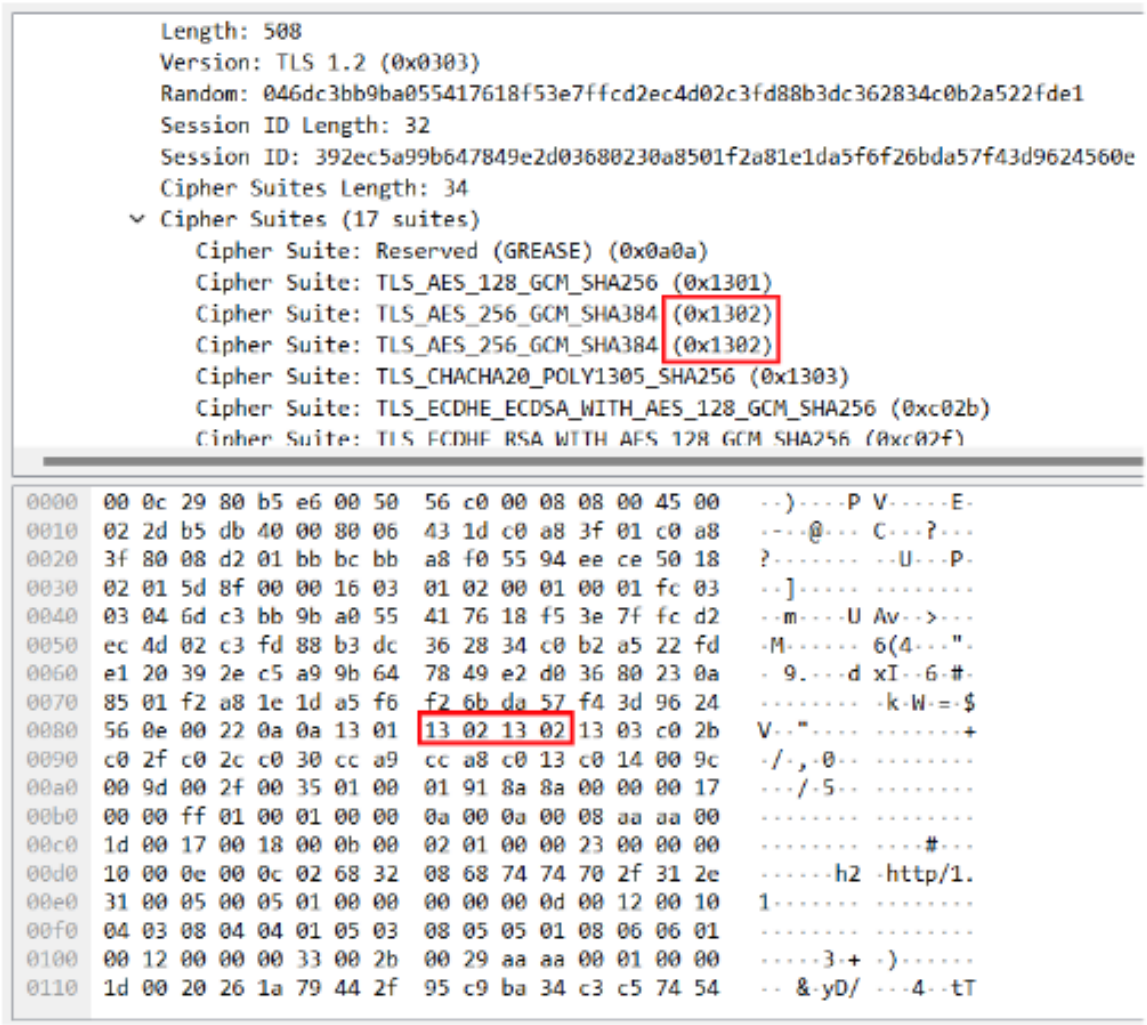

Figura 5.1: Pacchetto client hello inviato tramite Microsoft Edge

Capitolo 6

Conclusioni

Elenco delle figure

Bibliografia