

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

1	Introduzione	1
1.1	OS Fingerprinting	1
1.2	Obiettivo	1
1.3	Strumenti e sistemi operativi utilizzati	2
2	Base knowledge	4
2.1	Funzionamento dello stack TCP/IP	4
2.2	Header per il fingerprinting	5
3	Analisi	7
3.1	Livello 3: protocolli IP e ICMP	7
3.2	Livello 4: analisi protocollo TCP	8
3.3	Livello 7: analisi protocollo HTTP	9
4	Offuscamento	11
4.1	Algoritmo utilizzato da Nmap	11
4.2	Modifiche al file sysctl.conf	12
4.3	Modifiche tramite nftables	13
4.4	Offuscamento da fingerprinting passivo	17
5	Analisi handshake TLS	19
5.1	Handshake TLS	20
5.2	Browser fingerprinting	20
6	Conclusioni	26

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 appositamente costruiti allo scopo di 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 grazie all'assenza di pacchetti inviati.

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 a ingannare i principali strumenti per il fingerprinting. 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.

I tool utilizzati sono stati i seguenti:

- **Nmap**¹: principale strumento per effettuare fingerprinting attivo tramite l'invio di specifici pacchetti (*probe*) costruiti appositamente per massimizzare le differenze ricevute nelle risposte, evidenziando maggiormente le caratteristiche peculiari dei sistemi operativi. Consente inoltre di effettuare altre operazioni, alcune delle quali fondamentali per il fingerprinting stesso come ad esempio il port scanning.
- **p0f**²: tool per effettuare fingerprinting passivo. Esso analizza i pacchetti ricevuti da una determinata interfaccia oppure quelli passati tramite un file con estensione pcap. È anche in grado di effettuare fingerprinting a livello 7.
- **Wireshark**³: si tratta di uno strumento di sniffing che permette la visualizzazione dei pacchetti inviati e ricevuti dal dispositivo tramite un'interfaccia grafica. Consente inoltre di filtrarli sulla base di determinati campi o protocolli utilizzati.
- **Server Apache**⁴: 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.

¹<https://nmap.org>

²<https://www.kali.org/tools/p0f/>

³<https://www.wireshark.org/download.html>

⁴<https://httpd.apache.org>

- **nginx** ⁵: altro web server, installato per cercare le differenze rispetto al server Apache.
- **Scapy** ⁶: libreria Python in grado di inviare pacchetti con la possibilità di impostare determinati valori 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** ⁷: tool per che permette la manipolazione e il blocco di pacchetti sulla base del loro contenuto negli header.

⁵<https://www.nginx.com>

⁶<https://scapy.net>

⁷<https://manpages.debian.org/testing/nftables/nft.8.en.html>

Capitolo 2

Base knowledge

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 (modello ISO/OSI) 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 Funzionamento dello stack TCP/IP

Il meccanismo dello stack prevede che ad ogni livello vengano aggiunte al messaggio delle intestazioni (header), a partire dal layer più alto, che verranno valutate dal mede-

simo livello del ricevente. Si precisa che l'ordine in cui si valutano gli header dei livelli è inverso rispetto a quello del mittente; il ricevente partirà infatti dal livello più basso. Questa procedura prende il nome di *incapsulamento* ed è riassunta nella seguente figura:

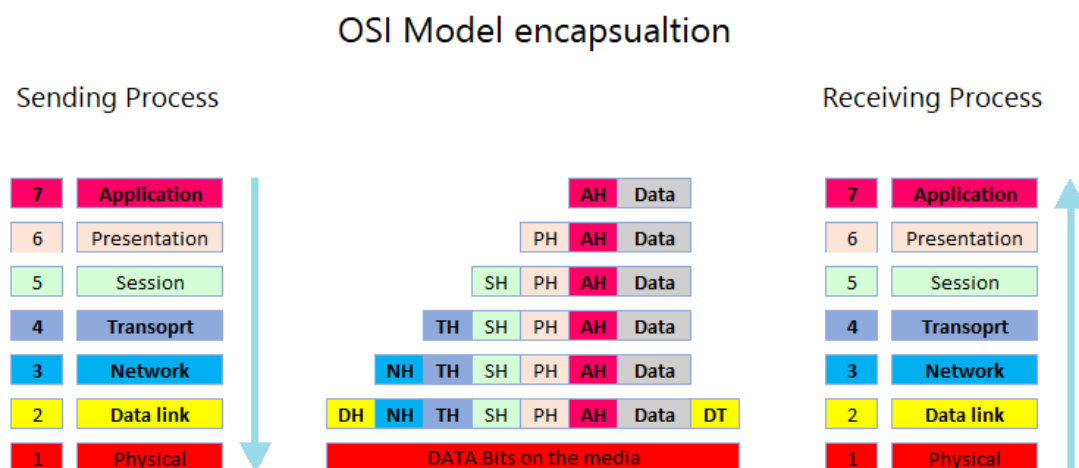


Figura 2.1: Incapsulamento nel modello ISO/OSI

[1]

2.2 Header per il fingerprinting

Gli header aggiunti ad ogni livello sono formati da vari campi contenenti 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 TCP, un protocollo del livello 4 dello stack:

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: Header TCP
[2]

Il campo *option* permette di segnalare al ricevente l'uso di alcune opzioni di comunicazione; 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 ad ogni livello dello stack, e l'unione delle informazioni acquisite dall'analisi degli header 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 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 utilizzando la libreria Scapy. Di seguito sono riportate le differenze più importanti rilevate durante l'intera attività di analisi.

3.1 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  pkt = sr1(IP(dst='192.168.63.1')/ICMP(code=9))
4  pkt.show()
```

Listing 3.1: Comando Python per l'invio del pacchetto

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

3.2 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=srl(IP(dst='192.168.63.1')/TCP(dport=80, flags='SCE'))
4  pkt.show()
```

Listing 3.2: Comando Python per l'invio del pacchetto

Windows 11	0
Kali	1

Tabella 3.4: ECN in risposta a specifico pacchetto

3.3 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

In questo capitolo si ha l'obiettivo di ingannare i tool di fingerprinting, portandoli ad un risultato errato. Più precisamente, si ha lo scopo di modificare parametri di Kali per ottenere il riconoscimento di Windows.

Non essendo Windows 11 presente nel database di Nmap e non potendo quindi essere un risultato del fingerprinting, si cercherà di portare i tool all'individuazione della sua versione precedente, ovvero Windows 10.

4.1 Algoritmo utilizzato da Nmap

Prima di procedere occorre precisare il funzionamento dell'algoritmo di Nmap per il riconoscimento dei sistemi operativi. Questo avviene secondo un meccanismo di punteggi che viene dato ad ogni test effettuato, specificato nella prima entry del database. I test che non ottengono alcun risultato utile non vengono conteggiati nel totale dei punti. A questo punto, vi sono due possibili situazioni:

- Se i risultati dei test combaciano perfettamente con quelli di una entry del database, allora quella verrà mostrata come risultato finale,
- Se i risultati dei test non coincidono con nessun sistema operativo, allora verranno calcolati i punti di ogni test soddisfatti per ogni sistema; quello con cui è stato realizzato un numero maggiore di punti sarà quello mostrato.

Segue l'elenco di test con il relativo punteggio:

```

1  MatchPoints
2  SEQ(SP=25%GCD=75%ISR=25%TI=100%CI=50%II=100%SS=80%TS=100)
3  OPS(O1=20%O2=20%O3=20%O4=20%O5=20%O6=20)
4  WIN(W1=15%W2=15%W3=15%W4=15%W5=15%W6=15)
5  ECN(R=100%DF=20%T=15%TG=15%W=15%O=15%CC=100%Q=20)
6  T1(R=100%DF=20%T=15%TG=15%S=20%A=20%F=30%RD=20%Q=20)
7  T2(R=80%DF=20%T=15%TG=15%W=25%S=20%A=20%F=30%O=10%RD=20%Q=20)
8  T3(R=80%DF=20%T=15%TG=15%W=25%S=20%A=20%F=30%O=10%RD=20%Q=20)
9  T4(R=100%DF=20%T=15%TG=15%W=25%S=20%A=20%F=30%O=10%RD=20%Q=20)
10 T5(R=100%DF=20%T=15%TG=15%W=25%S=20%A=20%F=30%O=10%RD=20%Q=20)
11 T6(R=100%DF=20%T=15%TG=15%W=25%S=20%A=20%F=30%O=10%RD=20%Q=20)
12 T7(R=80%DF=20%T=15%TG=15%W=25%S=20%A=20%F=30%O=10%RD=20%Q=20)
13 U1(R=50%DF=20%T=15%TG=15%IPL=100%UN=100%RIPL=100%RID=100%RIPCK=100%RUCK=100%RUD=100)
14 IE(R=50%DFI=40%T=15%TG=15%CD=100)

```

Listing 4.1: Punteggi che Nmap attribuisce ad ogni test

4.2 Modifiche al file sysctl.conf

Per ottenere un sistema operativo differente, occorre modificare alcuni parametri del file `sysctl.conf` in modo da ottenere i valori di Windows. La prima modifica riguarda il campo del TTL; variarlo da 64 a 128 consente di ingannare il test TG. La modifica vale per tutti i pacchetti che verranno inviati, quindi questo consente di spostare un grande quantità di punti a favore del riconoscimento di Windows, essendo il test ripetuto per molti pacchetti.

```

1  net.ipv4.ip_default_ttl=128

```

Listing 4.2: Modifica al campo TTL nel file `sysctl.conf`

Un'ulteriore modifica può essere effettuata per quanto riguarda il flag ECN, la cui differenza è stata spiegata nel capitolo precedente. Questo riguarda i test della quarta riga, in cui quello specifico per la notifica esplicita di congestione ha un valore di 100 punti.

```
1 net.ipv4.tcp_ecn=0
```

Listing 4.3: Modifica al campo ECN nel file sysctl.conf

Il valore 0 significa che l'host non può accettare nè inizializzare l'utilizzo del flag ECN. Il comportamento in risposta a determinati pacchetti "patologici" è ora simile a quello di Windows.

Si può inoltre procedere modificando il valore della Windows Size massima, causando di conseguenza un cambiamento del valore del Window Scaling Factor, da 7 a 8. La riga scritta nel file è la seguente:

```
1 net.core.rmem_max = 8388608
```

Listing 4.4: Modifica alla Windows Size massima nel file sysctl.conf

Questa modifica, presa singolarmente, non influenza il risultato del fingerprinting in quanto questo campo è contenuto all'interno delle TCP options, che vengono valutate in maniera aggregata. Servono quindi ulteriori modifiche che verranno illustrate successivamente.

4.3 Modifiche tramite nftables

La modifica dei pacchetti in uscita dall'host di cui si vuole effettuare il fingerprinting tramite nftables consente il cambiamento del comportamento solo in determinate situazioni; di particolare interesse sono quelle riguardanti i probe di Nmap.

La differenza mostrata alla tabella 3.2 è oggetto di uno dei test effettuati da Nmap. Quest'ultimo, infatti, invia un pacchetto *echo request* con campo *code* impostato a 9, e valuta la risposta ricevuta. Questa dovrebbe essere zero nel caso di un *echo reply*, ma questo comportamento è in realtà dipendente dal sistema operativo in utilizzo. Modificare i pacchetti echo reply in uscita per modificare il campo code nel caso questo sia 9 varia il risultato del test effettuato da Nmap. L'alto valore in termini di punteggio che questo test ha rende questa modifica molto importante ai fini del fingerprinting.

Il comando dato per ottenere questo comportamento è nel file nftables.conf, nella tabella output.


```
1 icmp type 0 icmp code 9 icmp code set 0
```

Listing 4.5: Modifica del campo code in caso di test Nmap

Esaminando il database di Nmap, ci si accorge che vi è una differenza nel Window Size tra Kali e Windows 10. Modificare questo campo permette quindi di falsificare un altro test eseguito da Nmap, contribuendo quindi alla realizzazione dell'offuscamento. Si procede dunque ad impostare la Window Size come descritta nel database.

```
1 tcp window != 0x2000 tcp window set 0x2000
```

Listing 4.6: listingsize

Attuando tutte le modifiche elencate fino a questo punto, l'output di Nmap risulta essere il seguente:

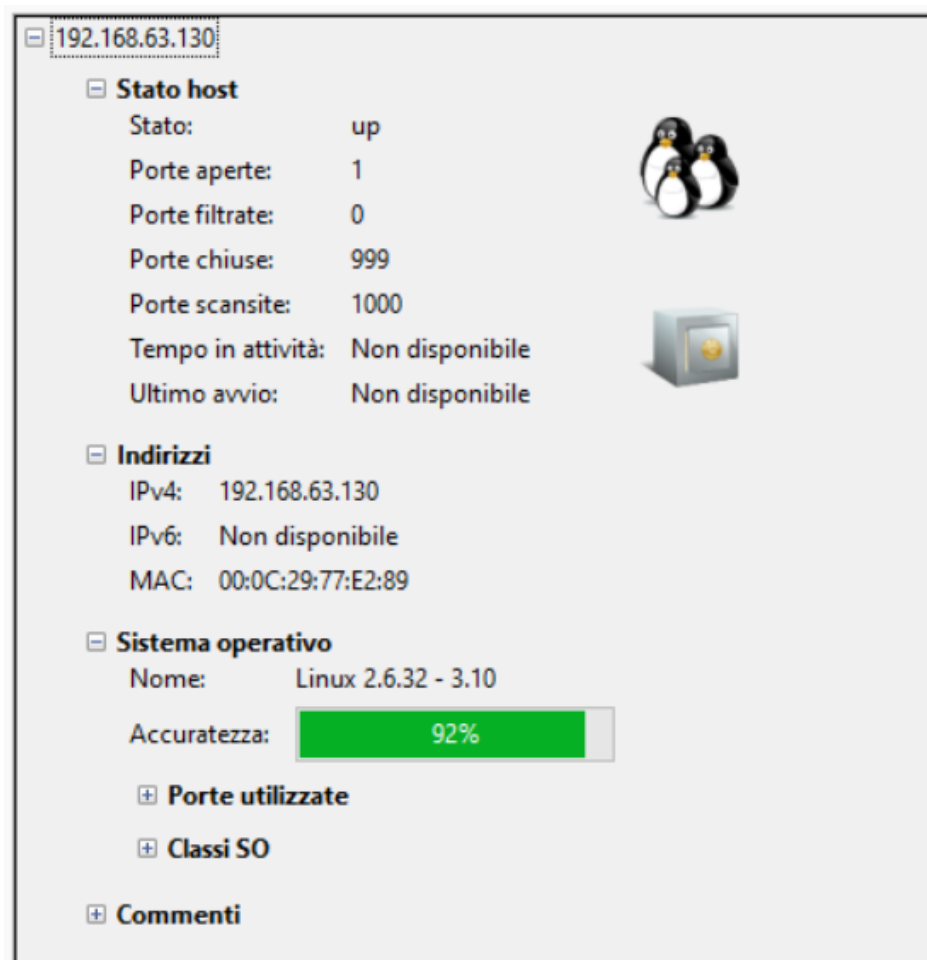


Figura 4.1: Risultato di Nmap

Come si può evincere dall'immagine, viene riconosciuta una versione di Linux non corrispondente a quella attualmente in uso. Visionando l'elenco completo dei test che vengono effettuati, si può notare che questi vengono preceduti da un port scanning, una procedura che ha l'obiettivo di esaminare l'apertura o meno di determinate porte dell'host. Questa è fondamentale ai fini del fingerprinting perché l'individuazione di porte aperte o chiuse permette di effettuare test differenti, in grado di massimizzare il numero di informazioni ricavabili dalle risposte. Di default, Nmap scansiona 1000 porte.

Nel risultato appena mostrato si può osservare come il tool abbia rilevato una sola porta aperta (la porta 80, su cui è attivo il server Apache) e 999 porte chiuse. I test eseguiti su quelle porte influenzano il risultato finale del fingerprinting, pertanto la possibilità di poterli escludere renderebbe molto più complessa l'individuazione del sistema operativo. Per poter realizzare quanto descritto, occorre che l'host rifiuti tutti i pacchetti ricevuti che non abbiano come porta destinazione 80 (Well-Known port corrispondente ad HTTP).

Si procede quindi inserendo una semplice regola nella tabella input delle nftables:

```
1 tcp dport != 80 drop
```

Listing 4.7: Regola per il blocco di tutti i pacchetti ricevuti non diretti alla porta 80

Con l'utilizzo di questa regola, tutte le porte che prima risultavano *chiuse* ora risultano *filtrate*; la mancanza di test per pacchetti inviati alle porte filtrate influenza pesantemente il risultato del fingerprinting.



Figura 4.2: Risultato di Nmap dopo il filtraggio dei pacchetti non diretti alla porta 80

Si tratta di un fingerprinting che Nmap definisce "aggressivo", dovuto al fatto che parecchi test non è stato possibile eseguirli; anche per questa ragione la percentuale di sicurezza di Nmap è 89%, un valore elevato ma che testimonia l'assenza di una certezza assoluta.

4.4 Offuscamento da fingerprinting passivo

Le tecniche seguite per ingannare i tool di fingerprinting passivo non presentano numerose differenze rispetto a quelle della sezione 4.3; vi è però da considerare il fatto che i pacchetti analizzati non sono ricevuti in risposta a determinati probe. Questo rende alcune regole impostate precedentemente non più funzionanti, come ad esempio la 4.5; in questo caso, infatti, l'host non riceve alcun pacchetto echo request con campo code impostato a 9.

Segue l'analisi del database di p0f, presente nel file al percorso `/etc/p0f/p0f.fp`, con i risultati dei test e le possibilità per un eventuale offuscamento.

```

1 [ tcp : request ]
2
3 sig = ver : ittl : olen : mss : wsize , scale : olayout : quirks : pclass
4
5 label = s : unix : Linux : 3.11 and newer
6 sig   = * : 64 : 0 : * : mss * 20 , 10 : mss , sok , ts , nop , ws : df , id + : 0
7 sig   = * : 64 : 0 : * : mss * 20 , 7 : mss , sok , ts , nop , ws : df , id + : 0
8
9 label = s : win : Windows : XP
10 sig   = * : 128 : 0 : * : 16384 , 0 : mss , nop , nop , sok : df , id + : 0
11 sig   = * : 128 : 0 : * : 65535 , 0 : mss , nop , nop , sok : df , id + : 0
12 sig   = * : 128 : 0 : * : 65535 , 0 : mss , nop , ws , nop , nop , sok : df , id + : 0
13 sig   = * : 128 : 0 : * : 65535 , 1 : mss , nop , ws , nop , nop , sok : df , id + : 0
14 sig   = * : 128 : 0 : * : 65535 , 2 : mss , nop , ws , nop , nop , sok : df , id + : 0

```

Listing 4.8: Fingerptinting TCP con p0f

Il primo valore di cui si può notare la differenza si trova in corrispondenza del campo *ittl*, corrispondente al Time To Live iniziale. Si tratta che è possibile modificare per ottenere un offuscamento modificando il file `sysctl.conf` come descritto al listing ???. Per quanto riguarda il campo *wsize* (riferito al Window Size), anche qui vi sono delle diversità, così come per il valore del Window Scaling; entrambi questi valori sono modificabili tramite le regole descritte al listing ?? e 4.4. Infine, si può riscontrare una differenza nelle opzioni TCP (campo *olayout*); queste, infatti, non vengono valutate solamente sul loro utilizzo ma anche nell'ordine in cui queste vengono presentate. È

evidente che l'ordinamento utilizzato dai due sistemi operativi sia differente, ma purtroppo non è stato trovato alcun modo per modificare quest'ultime.

Nonostante ciò, p0f non possiede un meccanismo di punteggio come Nmap, e non mostra il sistema operativo più simile a quello effettivamente testato. Questa caratteristica consente quindi di ottenere un offuscamento anche apportando modifiche minime al fine di determinare un nuovo comportamento del sistema operativo che non sia riconducibile a nessuna riga del database.

P0f analizza inoltre anche il fingerprinting del secondo pacchetto di un handshake TCP (SYN+ACK).

```

1  sig = ver:ittl:olen:mss:wscale:olayout:quirks:pclass
2
3  label = s:unix:Linux:3.x
4  sig   = *:64:0:*:mss*10,0:mss:df:0
5  sig   = *:64:0:*:mss*10,0:mss,sok,ts:df:0
6  sig   = *:64:0:*:mss*10,0:mss,nop,nop,ts:df:0
7  sig   = *:64:0:*:mss*10,0:mss,nop,nop,sok:df:0
8  sig   = *:64:0:*:mss*10,*:mss,nop,ws:df:0
9  sig   = *:64:0:*:mss*10,*:mss,sok,ts,nop,ws:df:0
10 sig   = *:64:0:*:mss*10,*:mss,nop,nop,ts,nop,ws:df:0
11 sig   = *:64:0:*:mss*10,*:mss,nop,nop,sok,nop,ws:df:0
12
13 label = s:win:Windows:7 or 8
14 sig   = *:128:0:*:8192,0:mss:df,id+:0
15 sig   = *:128:0:*:8192,0:mss,sok,ts:df,id+:0
16 sig   = *:128:0:*:8192,8:mss,nop,ws:df,id+:0
17 sig   = *:128:0:*:8192,0:mss,nop,nop,ts:df,id+:0
18 sig   = *:128:0:*:8192,0:mss,nop,nop,sok:df,id+:0
19 sig   = *:128:0:*:8192,8:mss,nop,ws,sok,ts:df,id+:0
20 sig   = *:128:0:*:8192,8:mss,nop,ws,nop,nop,ts:df,id+:0
21 sig   = *:128:0:*:8192,8:mss,nop,ws,nop,nop,sok:df,id+:0

```

Listing 4.9: Database fingerprinting per pacchetti SYN+ACK dell'handshake TCP

Osservando il database si può immediatamente notare come le differenze siano molto simili, in parte uguali, a quelle descritte nel listing precedente. Le strategie per ottenere l'offuscamento rimangono dunque le medesime.

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 viene posizionato tra il livello 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. Per poter decifrare i dati è necessaria la conoscenza di una chiave, la quale ovviamente è circoscritta ai due host. Solitamente avviene in forma simmetrica, ovvero la chiave per la cifratura è la medesima per la decifratura ed è identica per entrambi i componenti della comunicazione.

5.1 Handshake TLS

La presenza di una chiave comune ad entrambi i partecipanti pone il problema di come questi ne vengano a conoscenza, non potendo 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 interpretando uno dei due host a loro insaputa (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) permettendo 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 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.

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

5.2 Browser fingerprinting

L'analisi delle differenze nel contenuto dei pacchetti client hello inviati da differenti client permette di evidenziare differenze peculiari di quest'ultimi, consentendone la loro individuazione; in questo documento saranno analizzati i browser più popolari, installati su Windows con l'eccezione di Safari:

- Chrome versione 105.0.5195.127

- Mozilla Firefox versione 105.0
- Microsoft Edge versione 105.0.1343.42
- Opera versione 91.0.4516.20
- Safari versione 15.4

Esaminando i cifrari supportati, si può notare la loro differenza in numero:

Chrome	16
Firefox	17
Edge	17
Opera	16
Safari	21

Tabella 5.2: Cifrari supportati dai vari browser

Questo rende Safari differente rispetto a tutti gli altri browser, in quanto supportano un numero di cifrari diverso.

Si può inoltre notare come l'ordine dei cifrari elencati da Firefox sia diverso da quello degli altri browser: questo comporta un'ulteriore informazione utile al fingerprinting.

Mozilla Firefox	Altri browser
0x1301	0x1301
0x1303	0x1302
0x1302	0x1303
0xc02b	0xc02b
...	...

Tabella 5.3: Ordine dei primi cifrari dell'elenco

Ulteriori differenze si possono ottenere analizzando altri campi come per esempio il supporto ad un maggior numero di algoritmi di hashing per la firma digitale e l'assenza

del GREASE nei cifrari. Per quanto riguarda il primo valore, Firefox e Safari sono gli unici a presentare un numero diverso rispetto agli altri:

Chrome	8
Firefox	11
Edge	8
Opera	8
Safari	11

Tabella 5.4: Algoritmi di hashing supportati

Un'ulteriore informazione importante che si può ricavare dall'analisi degli algoritmi per la firma digitale consiste nell'ordine differente con cui questi vengono presentati. In particolare, confrontando la lista di Firefox con quella di Safari una disuguaglianza illustrata nella tabella seguente.

Mozilla Firefox	Safari
0x0403	0x0403
0x0503	0x0804
0x0603	0x0401
0x0804	0x0503
0x0805	0x0203
0x0806	0x0805
0x0401	0x0805
0x0501	0x0501
0x0601	0x0806
0x0203	0x0601
0x0201	0x0201

Tabella 5.5: Ordine con cui vengono presentati gli algoritmi di hashing

Si noti come un algoritmo di hashing venga ripetuto due volte all'interno dell'elenco di Safari, delineando di fatto un comportamento peculiare solo di Safari; anche nel caso in cui venisse rimossa questa duplicazione, si otterrebbe comunque una differenza in numero rispetto agli algoritmi elencati da Firefox (10 anziché 11).

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. [3] È quindi logico che il GREASE venga posizionato in cima alla lista dei cifrari, in modo che sia sempre valutato dal server.

Chrome	Sì
Firefox	No
Edge	Sì
Opera	Sì
Safari	Sì

Tabella 5.6: Presenza del GREASE nella lista dei cifrari supportati

In questo caso, l'unico browser a non utilizzare quest'opzione è Firefox; ciò significa solamente avendo quest'informazione si è in grado di individuare quale tra i cinque browser in esame si sta utilizzando.

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 ovviamente non ha alcuna utilità, in quanto in caso di rifiuto del server per quello specifico cifrario si procederebbe a valutarlo nuovamente, producendo conseguentemente un nuovo rifiuto. Si tratta dell'unico browser dei cinque che ha questo comportamento che lo rende molto vulnerabile al fingerprinting. Di seguito la schermata di Wireshark che mostra il pacchetto catturato con questa caratteristica.

Length: 508		
Version: TLS 1.2 (0x0303)		
Random: 046dc3bb9ba055417618f53e7ffcd2ec4d02c3fd88b3dc362834c0b2a522fde1		
Session ID Length: 32		
Session ID: 392ec5a99b647849e2d03680230a8501f2a81e1da5f6f26bda57f43d9624560e		
Cipher Suites Length: 34		
v Cipher Suites (17 suites)		
Cipher Suite: Reserved (GREASE) (0x0a0a)		
Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)		
Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)		
Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)		
Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)		
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)		
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)		
0000	00 0c 29 80 b5 e6 00 50 56 c0 00 08 08 00 45 00	--.)---P V-----E-
0010	02 2d b5 db 40 00 80 06 43 1d c0 a8 3f 01 c0 a8	---@---C---?---
0020	3f 80 08 d2 01 bb bc bb a8 f0 55 94 ee ce 50 18	?-----U---P-
0030	02 01 5d 8f 00 00 16 03 01 02 00 01 00 01 fc 03	--]-----
0040	03 04 6d c3 bb 9b a0 55 41 76 18 f5 3e 7f fc d2	--m---U Av-->---
0050	ec 4d 02 c3 fd 88 b3 dc 36 28 34 c0 b2 a5 22 fd	-M-----6(4---"
0060	e1 20 39 2e c5 a9 9b 64 78 49 e2 d0 36 80 23 0a	- 9....d xI--6-#-
0070	85 01 f2 a8 1e 1d a5 f6 f2 6b da 57 f4 3d 96 24	-----k-W--=\$
0080	56 0e 00 22 0a 0a 13 01 13 02 13 02 13 03 c0 2b	V--"-----+
0090	c0 2f c0 2c c0 30 cc a9 cc a8 c0 13 c0 14 00 9c	-/.-0-----
00a0	00 9d 00 2f 00 35 01 00 01 91 8a 8a 00 00 00 17	---/-5-----
00b0	00 00 ff 01 00 01 00 00 0a 00 0a 00 08 aa aa 00	-----#---
00c0	1d 00 17 00 18 00 0b 00 02 01 00 00 23 00 00 00	-----#---
00d0	10 00 0e 00 0c 02 68 32 08 68 74 74 70 2f 31 2e	-----h2 -http/1.
00e0	31 00 05 00 05 01 00 00 00 00 00 0d 00 12 00 10	1-----
00f0	04 03 08 04 04 01 05 03 08 05 05 01 08 06 06 01	-----
0100	00 12 00 00 00 33 00 2b 00 29 aa aa 00 01 00 00	-----3+ -)-
0110	1d 00 20 26 1a 79 44 2f 95 c9 ba 34 c3 c5 74 54	-- &-yD/ -4--tT

Figura 5.1: Pacchetto client hello inviato tramite Microsoft Edge

Nonostante i cifrari utilizzati siano uguali in numero, questo delinea una differenza di comportamento tra Edge e Firefox agevolando quindi il loro riconoscimento. Inoltre, l'eliminazione del cifrario ripetuto porterebbe Edge ad una situazione uguale a quella di Chrome e Opera in quanto a numero di cifrari, rendendoli quindi indistinguibili: l'ordine dei cifrari e le altre caratteristiche, infatti, sarebbero a quel punto identiche.

Ciò è probabilmente dovuto al fatto che questi tre browser si basano tutti su Chromium¹, un web browser open source sul quale essi sono basati. Per questa ragione sono stati esclusi dalla trattazione Brave e Vivaldi, anch'essi basati su Chromium, in quanto dopo essere stati analizzati non differivano rispetto a Chrome e Opera.

In sintesi, si può affermare che tre dei cinque browser analizzati si possano distingue-

¹<https://www.chromium.org/chromium-projects/>

re utilizzando soltanto dal pacchetto client hello inviato durante l'handshake TLS nel seguente modo:

1. L'assenza del GREASE implica l'utilizzo di Firefox.
2. Il numero di cifrari supportati permette la distinzione di Safari.
3. La ripetizione di un algoritmo per la firma digitale consente un'ulteriore elemento di distinzione di Safari.
4. La presenza di un cifrario duplicato consente di individuare Edge.
5. La differenza in numero algoritmi di hashing comporta una distinzione di Firefox e Safari rispetto ai tre browser rimanenti.

Capitolo 6

Conclusioni

Queste sono le conclusioni della tesi.

Elenco delle figure

2.1	Incapsulamento nel modello ISO/OSI	5
2.2	Header TCP	6
4.1	Risultato di Nmap	14
4.2	Risultato di Nmap dopo il filtraggio dei pacchetti non diretti alla porta 80	16
5.1	Pacchetto client hello inviato tramite Microsoft Edge	24

Bibliografia

- [1] Infodoc. Sistemi e reti: incapsulamento.
<http://infodoc.altervista.org/sistemi-e-reti/incapsulamento/>, visitato in Settembre 2022.

- [2] Ionos. Presentazione tcp.
<https://www.ionos.it/digitalguide/server/know-how/presentazione-tcp/>, visitato in Settembre 2022.

- [3] Ionos. Rfc 8701. <https://www.rfc-editor.org/rfc/rfc8701.html>, visitato in Settembre 2022.