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

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.

L'obiettivo di questa tesi 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.
Quest'operazione trova la sua utilità nel fatto che l'individuazione di un sistema operativo da remoto sia cruciale ai fini della sicurezza, dal momento che quando viene effettuato un attacco (da una persona malintenzionata oppure dall'amministratore di sistema per una verifica della sicurezza), l'attaccante potrebbe avere un solo tentativo a disposizione [7]. Ogni attacco è infatti preceduto da una fase di ricognizione, e

l'OS fingerprinting può essere un ottimo metodo per ottenere informazioni utili circa il dispositivo della vittima [1]. Per l'ottenimento dell'offuscamento, si è proceduto in primis con l'identificazione delle differenze tra i sistemi analizzando i pacchetti che venivano inviati in risposta a richieste GET HTTP; a tale scopo sono stati installati server Apache su Windows 11 e Kali. I tentativi di offuscamento sono stati effettuati tramite modifiche ad alcuni file di configurazione e la manipolazione di pacchetti in uscita.

Infine, sono state valutate le differenze tra i principali browser web rispetto all'handshake TLS; l'analisi di quest'ultimo può infatti portare all'individuazione del browser in utilizzo.

Capitolo 2

Base knowledge

Per poter 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 (ISO/OSI) 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	Host-to-network

Tabella 2.1: Livelli dello stack TCP/IP

2.1 Header per il fingerprinting

Gli header aggiunti ad ogni livello tramite il meccanismo dell'incapsulamento 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 TCP, un protocollo del livello 4 dello stack:

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 2.1: Header TCP

[3]

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 dunque particolare importanza ai fini del fingerprinting. Esempi analoghi si possono riscontrare 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. I test condotti vanno eseguiti successivamente ad alcune operazioni preliminari con lo scopo di effettuare una ricognizione dell'host. Ai fini del fingerprinting attivo è infatti fondamentale effettuare prima un *port scanning*, ovvero un'operazione che consente di individuare le porte aperte e quelle chiuse utile per i test riguardanti il quarto livello dello stack. Nmap effettua di default la scansione di 1000 porte.

2.2 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. Come specificato nella sezione precedente, consente anche di effettuare altre operazioni 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.

- **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.

[1] `https://nmap.org`
[2] `https://www.kali.org/tools/p0f/`
[3] `https://www.wireshark.org/download.html`
[4] `https://httpd.apache.org`
[5] `https://www.nginx.com`
[6] `https://scapy.net`
[7] `https://manpages.debian.org/testing/nftables/nft.8.en.html`

Capitolo 3

Analisi

3.1 Procedura

L'analisi dei pacchetti per l'individuazione delle differenze è stata effettuata sulle risposte ricevute da server Apache installati su Windows 11 e su Kali. L'attività si analisi è stata svolta in due differenti modalità:

1. Nella prima sono state analizzate le risposte ricevute in seguito a richieste GET HTTP inviate tramite browser web; la cattura e l'osservazione di queste è stata effettuata tramite Wireshark.

2. Nella seconda si è proceduto con l'ispezione delle risposte a specifici pacchetti inviati utilizzando la funzione *sr1* della libreria Python Scapy, che permette la cattura della risposta al pacchetto inviato. Le risposte sono state esaminate con la funzione *show*.

Di seguito vengono mostrate le principali differenze notate tra i due sistemi operativi durante l'intera attività di analisi.

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	64

Tabella 3.1: TTL iniziale

Questa differenza è molto importante in quanto è presente in ogni pacchetto inviato in rete e, se non vengono apportate modifiche, non varia tra un pacchetto e l'altro. Solitamente i valori iniziali sono potenze di due (64 o 128); l'eventuale utilizzo di valori parecchio inusuali denota un comportamento molto riconoscibile.

Inoltre, è 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

La differenza è molto peculiare perchè riconoscibile utilizzando solamente il protocollo ICMP (lo stesso del comando *ping*) e non ne influenza il normale utilizzo dal momento che la differenza si presenta solo inviando determinati pacchetti *echo request*.

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

Si tratta di una differenza importante, la quale però per essere rilevata necessita del livello 4 dello stack; inoltre, si può osservare solo se si utilizza il protocollo TCP, essendo UDP privo di handshake.

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()
```

Listing 3.2: Comando Python per l'invio del pacchetto

Windows 11	0
Kali	1

Tabella 3.4: ECN in risposta a specifico pacchetto

Questa diversità è molto importante per il fingerprinting di Nmap che, come verrà meglio specificato nella sezione 4.1, assegna una punteggio elevato al risultato di questo test.

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*.

Inoltre, a questo livello dello stack si può 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 in determinati bit per ogni funzione. L'header, infatti, 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; ne consegue che analizzare l'ordinamento è molto importante se si vuole tentare di individuare, ad esempio, il tipo di client in utilizzo.

Osservando i pacchetti HTTP inviati dai principali browser web si possono notare delle differenze sull'ordinamento per quanto riguarda Firefox e altri browser basati su Chromium. L'ordine dei campi nell'header è infatti il seguente:

Firefox	**Chromium-based**
Host	Host
User-Agent	Connection
Accept	Upgrade-Insecure-Requests
Accept-language	User-Agent
Accept-Encoding	Accept
Connection	Accept-Encoding
Upgrade-Insecure-Requests	Accept-Language

Tabella 3.5: Ordine campi header HTTP

Inoltre, analizzando i linguaggi accettati si possono osservare ulteriori disuguaglianze riguardanti il campo *Quality Value*. Si tratta di un valore indicato tramite la lettera q (case insensitive) che esprime la preferenza a determinati linguaggi tramite un numero compreso tra 0 e 1, avente massimo tre cifre decimali; il valore di default è 1 ovvero la massima preferenza [6].

Segue la stringa di quello specifico campo riguardante Chrome e Firefox:

```
1 Acceept-Language: it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7
```

Listing 3.3: Campo Accept-Language di Chrome

```
1 Acceept-Language: it-IT,it;q=0.8,en-US;q=0.5,en;q=0.3
```

Listing 3.4: Campo Accept-Language di Firefox

Si possono dunque distinguere delle differenze tra i due browser, che ovviamente possono rivestire un ruolo importante in fase di fingerprinting dello User-Agent utilizzato.

È possibile, inoltre, osservare ulteriori differenze che vi sono tra i due browser analizzando il campo *Accept* dell'Header HTTP. Si osservino infatti i due campi nei due browser web menzionati precedentemente:

```
1 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
2 image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
```

Listing 3.5: Campo *Accept* di richiesta GET di Chrome

```
1 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,
2 image/webp,*/*;q=0.8
```

Listing 3.6: Campo *Accept* di richiesta GET di Firefox

Capitolo 4

Offuscamento

In questa capitolo si ha l'obiettivo di ingannare i tool di figerprinting, 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 :

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

Listing 4.1: Punteggi che Nmap attrribuisce ad ogni test [5]

Come si può notare, alcuni test hanno un valore maggiore rispetto ad altri per la frequenza con cui vengono svolti. Ad esempio il test T, che controlla il valore del TTL iniziale, viene effettuato nove volte mentre il test CD che verifica il campo code nel protocollo ICMP viene effettuato una sola volta. Questa è una delle ragioni che spiega la discrepanza tra i punteggi attribuiti ai diversi test che vengono svolti. Un ulteriore esempio di quanto appena affermato si può trovare tra i test della riga OPS (effettuati sulle opzioni TCP) e i test sulla riga U1, riguardanti il protocollo UDP. [1]

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 T, che calcola il valore iniziale di questo campo. La modifica vale per tutti i pacchetti che verranno

[1] `https://nmap.org/book/osdetect-methods.html`

inviati, quindi questo consente di spostare un grande quantità di punti a favore del riconoscimento di Windows, essendo il test ripetuto per molti pacchetti.

```
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 nella sezione 3.4. Questo riguarda i test della quarta riga, in cui quello specifico per la notifica esplicita di congestione ha un valore di 100 punti.

```
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 né 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:

```
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. Manipolare i pacchetti echo reply in uscita per modificare il campo code nel caso questo sia 9 consente la variazione del risultato del test effettuato da Nmap. L'alto valore in termini di punteggio assegnato a questo test rende questa modifica molto importante ai fini dell'offuscamento.
Il comando per ottenere questo comportamento è contenuto 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: Modifica della Window Size

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

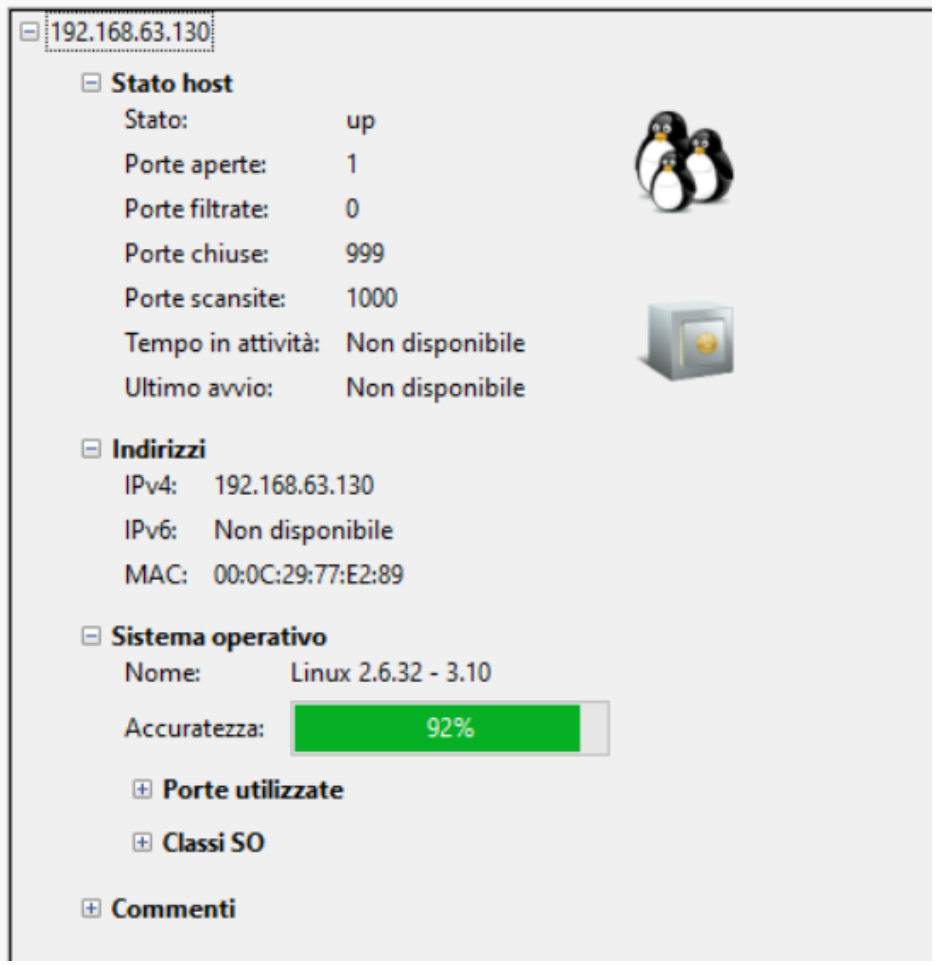

Figura 4.1: Risultato di Nmap

Come si può evincere dall'immagine, viene riconosciuta una versione di Linux non corrispondente a quella attualmente in uso. Questo controllo è possibile effettuarlo eseguendo il seguente comando sul terminale:

```
1 hostnamectl
```

Listing 4.7: Comando per mostrare la versione attualmente in uso

Visionando ulteriormente il risultato, si può notare come siano state rilevate sia porte aperte che porte chiuse. Questo consente una maggiore precisione nel fingerprinting perché i test vengono effettuati su entrambe le condizioni delle porte, individuate precedentemente tramite port scanning; è quindi logico che evitare questa situazione consente di influenzare parecchio il risultato del fingerprinting.
Per poter evitare la situazione appena descritta, occorre quindi che l'host rifiuti tutti i pacchetti ricevuti che non abbiano come porta destinazione 80 (Well-Known port

corrispondente al protocollo applicativo HTTP).

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

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

Listing 4.8: 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.

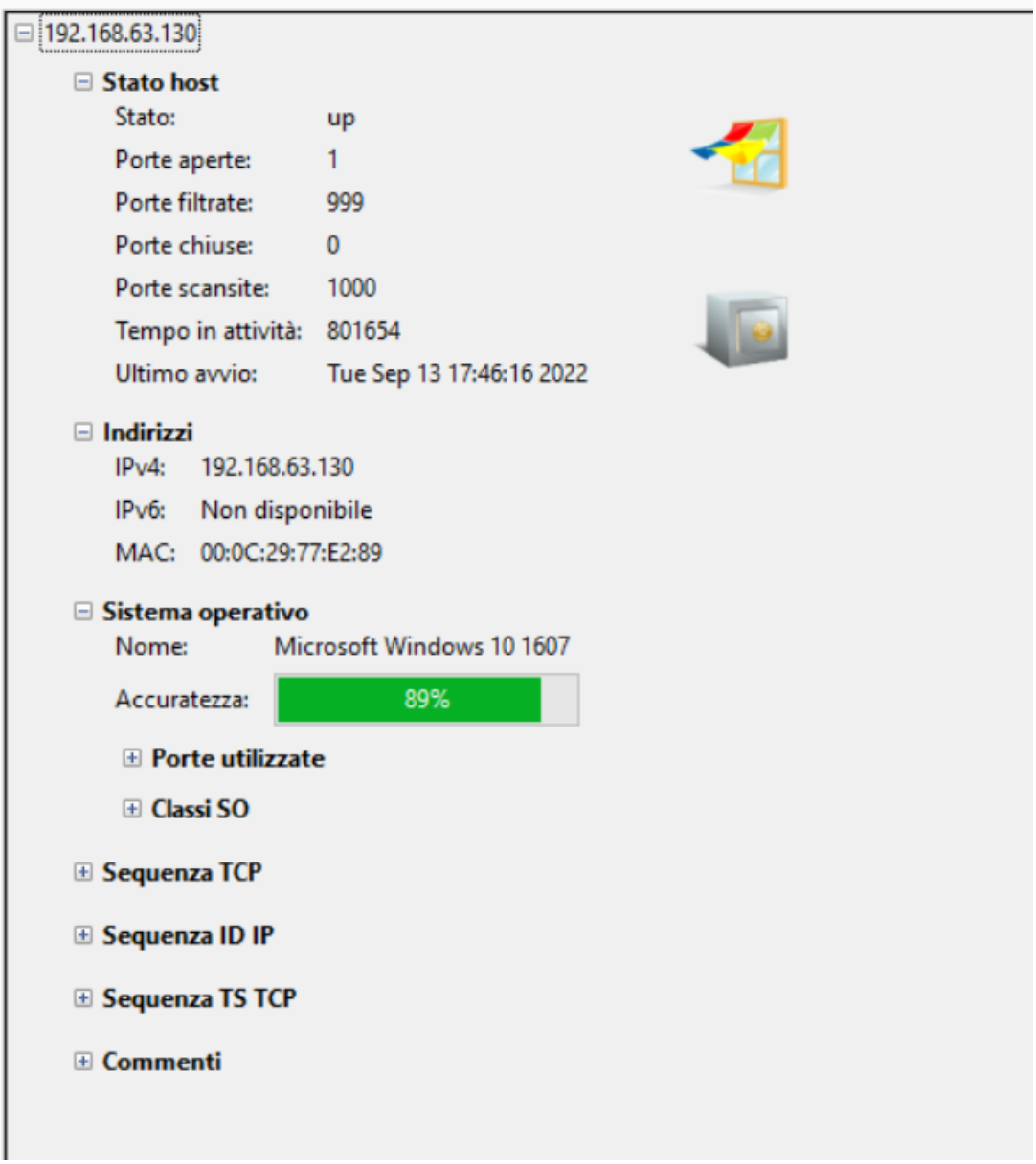

Figura 4.2: Risultato di Nmap dopo il filtraggio dei pacchetti non diretti alla porta 80

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.
Si può quindi affermare che la mancanza di porte chiuse in favore di quelle filtrate sia una strategia utile a contrastare il fingerprinting attivo, diminuendo di fatto i controlli che possono essere effettuati sui pacchetti.

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. L'elenco dei campi è specificato nella documentazione offerta da p0f [4], e si basa sui pacchetti SYN dell'handshake TCP.

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

Listing 4.9: 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 di un campo che è possibile modificare per ottenere un offuscamento modificando il file sysctl.conf come descritto al listing 4.2. 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 4.6 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:wsize,scale: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 4.10: 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.

Per una questione di spazio nel documento, i cifrari verranno riportati utilizzando il loro codice e non il loro nome completo

Mozilla Firefox	**Chromium-based**
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 algotirmi 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 [2]. 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 opacchetto catturato con questa caratteristica.

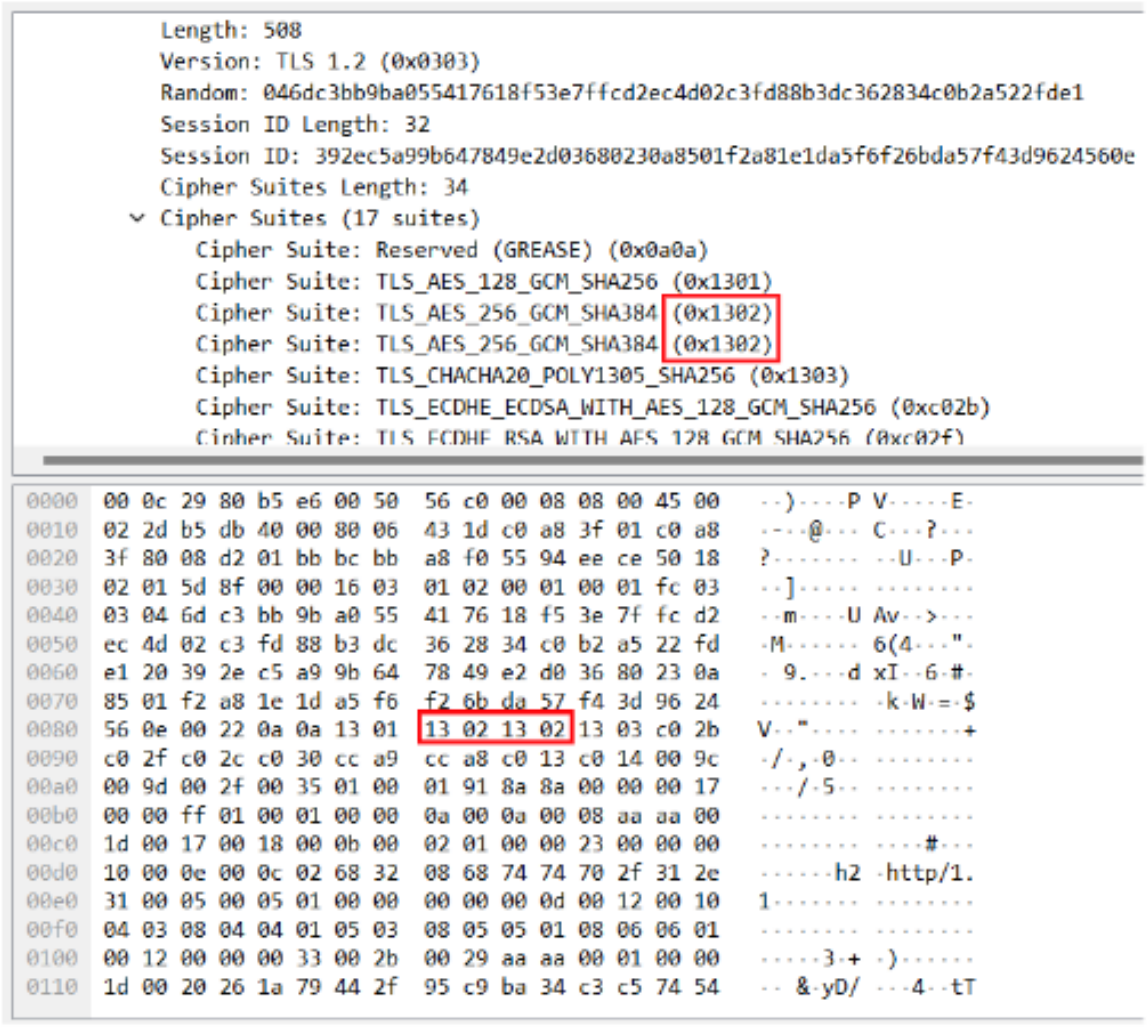

Figura 5.1: Pacchetto client hello inviato tramite Microsoft Edge

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 Chromium [1], 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

Sintetizzando quanto descritto nei capitoli precedenti, si può affermare che il fingerprinting attivo sia relativamente semplice da offuscare se si conosce il tool in utilizzo; questo perchè si conoscerebbero in anticipo i test e i probe inviati dal tool permettendo quindi una difesa mirata ad ingannare solo aspetti specifici di un protocollo di rete, come avvenuto nel listing 4.5.
È inoltre possibile effettuare un offuscamento modificando parametri che intaccano in misura trascurabile il funzionamento della rete come ad esempio il campo TTL; la modifica di questo valore, se impostato a valori alti (solitamente 64 o 128), permette di raggiungere comunque host che distano parecchi hop.
Il fingerprinting passivo richiede invece una modifica del normale comportamento del sistema operativo, non essendoci l'invio di determinati pacchetti da parte del tool, rendendo più complesse le operazioni di offuscamento.

L'individuazione del browser in utilizzo è molto peculiare in quanto questa è possibile analizzando un solo pacchetto: la prima richiesta GET se si vuole analizzare il protocollo HTTP o il client hello se si vuole ispezionare il protocollo TLS. Nel secondo caso, ottenere l'offuscamento mediante la modifica dei cifrari utilizzati può comportare delle problematiche, ovvero:

- Diminuire la lista dei cifrari può inficiare la compatibilità con alcuni server, limitando di fatto la scelta di quest'ultimi aumentando il rischio che non vi siano

cifrari supportanti sia dal client che dal server,

- Aumentare la lista consente la scelta di cifrari potenzialmente più deboli e attaccabili, con conseguenze sulla confidenzialità dei dati inviati in caso di attacco MITM passivo.

Al momento l'unica modifica ai cifrari che previene il fingerprinting senza influenzare la sicurezza o la compatibilità consiste nella rimozione del cifrario ripetuto in Edge mostrato nell'immagine 5.1.

Il medesimo ragionamento si può applicare agli algoritmi per la firma digitale.

Elenco delle figure

Bibliografia

[1] Jon Mark Allen. *OS and Application Fingerprinting Techniques.* visitato in Settembre 2022.

[2] David Benjamin. Applying generate random extensions and sustain extensibility (grease) to tls extensibility. https://www.rfc-editor.org/rfc/rfc8701.html, visitato in Settembre 2022.

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

[4] Marek. p0f v3: passive fingerprinter. https://github.com/p0f/p0f/blob/master/docs/README, visitato in Settembre 2022.

[5] Nmap. Nmap os fingerprinting 2nd generation db. https://svn.nmap.org/nmap/nmap-os-db, visitato in Giugno 2022.

[6] Julian Reschke Roy T. Fielding, Mark Nottingham. Http semantics. https://www.rfc-editor.org/rfc/rfc9110.html, visitato in Settembre 2022.

[7] Riaan Stopforth. *Techniques and Countermeasures of TCP/IP OS Fingerprinting on Linux Systems.* visitato in Ottobre 2022.