UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

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

Network Security

Moving Target Defence with Software Defined Networks

Anno Accademico 2023/2024

Professore: **Simon Pietro Romano**
Membri del Team:
Giampetraglia Federica M63001358
Vallefuoco Roberto M63001443
Carillo Raffaele M63001321
De Simone Anna M551278

Contents

Introduzione

Introduzione

Negli ultimi anni, la crescente complessità delle architetture di rete e la necessità di garantire elevati livelli di sicurezza hanno spinto verso lo sviluppo di soluzioni innovative come il Software Defined Networking (SDN). Questo progetto mira a esplorare l'implementazione di una rete SDN utilizzando il controller Ryu, integrato con Snort, un noto sistema di rilevamento e prevenzione delle intrusioni (IDS/IPS). Le tecnologie sono implementate all'interno di un ambiente Docker, utilizzando Containernet, una variante di Mininet, per simulare una rete complessa che permette di testare efficacemente le configurazioni di sicurezza.

L'obiettivo principale del progetto è dimostrare come la flessibilità di SDN possa essere utilizzata per migliorare la sicurezza delle reti tramite strategie di Moving Target Defense e l'impiego di honeypots, complicando gli schemi di attacco degli aggressori e riducendo così le vulnerabilità note.

In questa documentazione, descriveremo dettagliatamente l'ambiente di sviluppo, la configurazione e l'implementazione della soluzione, i test condotti e le potenziali aree di sviluppo futuro. La struttura del documento segue una progressione logica che va dalla teoria alla pratica, delineando chiaramente i passaggi implementativi e le verifiche delle soluzioni proposte.

Chapter 1

Background

1.1 Software Defined Networking (SDN) e Moving Target Defense (MTD)

Il Software Defined Networking (SDN) è un'architettura di rete che separa il piano di controllo dal piano di dati, aumentando la flessibilità e la gestibilità delle reti. Questa separazione permette agli amministratori di centralizzare e semplificare la gestione delle politiche di traffico, riducendo gli errori e migliorando la reattività ai cambiamenti.

Le tecniche di Moving Target Defense (MTD) consistono nel modificare la topologia di rete col fine di disturbare e interrompere l'azione di attacco. In particolare è possibile modificare i flussi di traffico in maniera automatizzata permettendo operazioni di analisi e di raccolta di Cyber Threat Intelligence (CTI). Le SDN possono essere utilizzate come strumento per implementare una MTD reindirizzando il traffico malevolo verso Honeypot, variando gli indirizzi IP degli host attaccati oppure come semplice strumento per creare confusione all'interno della rete bersaglio mascherando e modificando gli indirizzi IP degli host. Gli honeypot servono quindi come trappole che non solo deviano e rallentano gli attacchi, ma forniscono anche preziose informazioni sugli attaccanti, migliorando così le strategie di difesa.

1.2 Il Controller Ryu

Ryu é un controller SDN open-source che offre una piattaforma programmabile, per lo sviluppo di reti complessi. É popolare per la sua compatibilitá con vari standard di rete tra cui OpenFlow, uno standard che permette l'interazione diretta tra controller e switch di rete (viene utilizzato dal controller per la gestione del flusso di pacchetti nella rete, tramite la modifica dinamica delle tabelle di flusso degli switch). Il software del controller risulta facilmente programmabile tramite il linguaggio python, che permette agli sviluppatori di definire tramite dei semplici script tutti i comportamenti che deve avere la rete in base ad un'analisi dei pacchetti in ingresso.

1.3 Snort come IDS/IPS

Snort é un sistema di prevenzione e rilevamento delle intrusioni utilizzato per il monitoraggio del traffico di rete e l'analisi dei pacchetti in ingresso/uscita per il riconoscimento di firme di attacchi. L'analisi del traffico di rete é gestito in tempo reale, infatti il programma decodifica ogni pacchetto in ingresso, attraverso l'interpretazione dei portocolli e delle varie strutture dati del pacchetto e dopo questa prima fase di decodifica viene applicato al pacchetto decodificato un confronto con diverse regole definite dall'utente (utile per l'aggiornamento delle firme di nuovi attacchi) per il riconoscimento di un pattern d'attacco. Snort puó essere configurato in tre modi

- Sniffer, dove il componente viene utilizzato solo per la cattura e log dei pacchetti, per permettere successivamente un'analisi offline del traffico di rete;
- IDS (Intrusion Detection System), dove al riconoscimento di determinati pattern di attacco (definiti all'interno di un database di regole) vengono alzati degli alert che dovranno poi essere eventualmente gestiti da un controller di rete (nel nostro caso gli alert vengono inviati tramite un socket da snort al controller ryu);
- IPS (intrusion Prevention System), dove sará snort stesso a bloccare il traffico stesso al riconoscimento di un intrusione (in questo caso il componente snort risulta avere una funzione particolarmente piú attiva rispetto alla semplice analisi del traffico).

1.4 Containernet

Containernet é una versione modificata di Mininet (un simulatore di rete), che aggiunge il supporto per la creazione di container docker all'interno di topologie di rete simulate. Come con Mininet tramite questo componente é stato possibile creare una topologia di rete abbastanza complessa (abbiamo simulato una rete aziendale e dei nodi esterni) tramite uno script python, dove é stato possibile definire switch, vlan, nodi e link tra i vari dispositivi.

1.5 Heralding

Il primo container utilizzato è Heralding, si tratta di un honeypot a bassa interazione, infatti i vari servizi non sono implementati completamente ma vengono emulate solo le risposte dei vari software e non accettano mai una connessione in ingresso, ció permette quindi di leggere e loggare tutti i tentativi di accesso da parte di un utente malevolo senza peró il rischio che l'utente possa effettivamente sfruttare il bug del software, visto che il servizio di rete non é effettivamente installato e funzionante nell'honeypot. I servizi emulati sono:

- FTP;
- Telnet;
- SSH;
- HTTP;
- POP3;
- IMAP;
- SMTP.

1.6 Dionaea

Dionaea è un honeypot a bassa interazione. A differenza di Heralding il suo livello di interattività è comunque molto più elevato, infatti non solo permette a un eventuale attaccante l'accesso a dei servizi mediante credenziali vulnerabili ma gli permette anche di effettuare alcune delle

operazioni previste dal vettore d'attacco utilizzato e ove previsto simula la presenza di un piccolo filesystem. Dionaea effettua il logging di queste attività anche all'interno di un database relazionale sqlite. I servizi emulati sono:

- EPMAP
- FTP
- HTTP
- DNS
- Telnet
- NTP
- Memcache
- Mirror
- MongoDB
- MQTT
- MsSQL
- MySQL
- PPTP
- SIP
- SMB
- TFTP
- UPNP

Oltre a questa vasta collezione di protocolli offre una serie di handler automatici tra cui l'invio di richieste HTTP, lo storing dei campioni catturati su Amazon AWS S3 e su HPFEEDS, l'analisi su VirusTotal nonché il logging degli eventi in vari formati.

Chapter 2

Implementazione

Lo scenario ipotizzato è quello di una classica rete aziendale. Abbiamo diverse subnet adibite a usi diversi, in genere una tipica rete aziendale separa gli host funzionali ai servizi esterni e la rete dedicata alle macchine del personale aziendale. Chiaramente è in genere presente anche una serie di host dedicati al management della suddetta rete. Nel nostro caso poi è prevista anche una sottorete dedicata alla raccolta di CTI. Viene fatta un'ipotesi semplificativa riguardo agli attaccanti che sono connessi mediante uno switch direttamente al router principale.

2.1 Topologia della Rete

La topologia della rete consiste in 3 VLAN connesse a un OpenVSwitch a sua volta connesso a un router secondo il classico schema del router-on-stick. Il router svolge la funzione di gateway predefinito per tutti gli host della topologia. Sul router sono state definite delle rotte statiche per ognuna delle sottoreti ad esso collegate. I dettagli di configurazione degli host vengono riportati in Tab. 2.1.

Una delle porte dello switch è dedicata a Snort, il quale funziona come IDS e qualora dovesse individuare un'anomalia manderà un alert al controller Ryu. Il controller a sua volta modificherà le regole di instadamento dei flussi sullo switch in modo tale che il traffico malevolo venga inidirizzato agli honeypot.

La rete aziendale possiede una sottomaschera di rete /16, pur trattandosi di una topologia le cui necessità sono molto più piccole, la rete utilizza comunque un range di indirizzi riservati all'uso

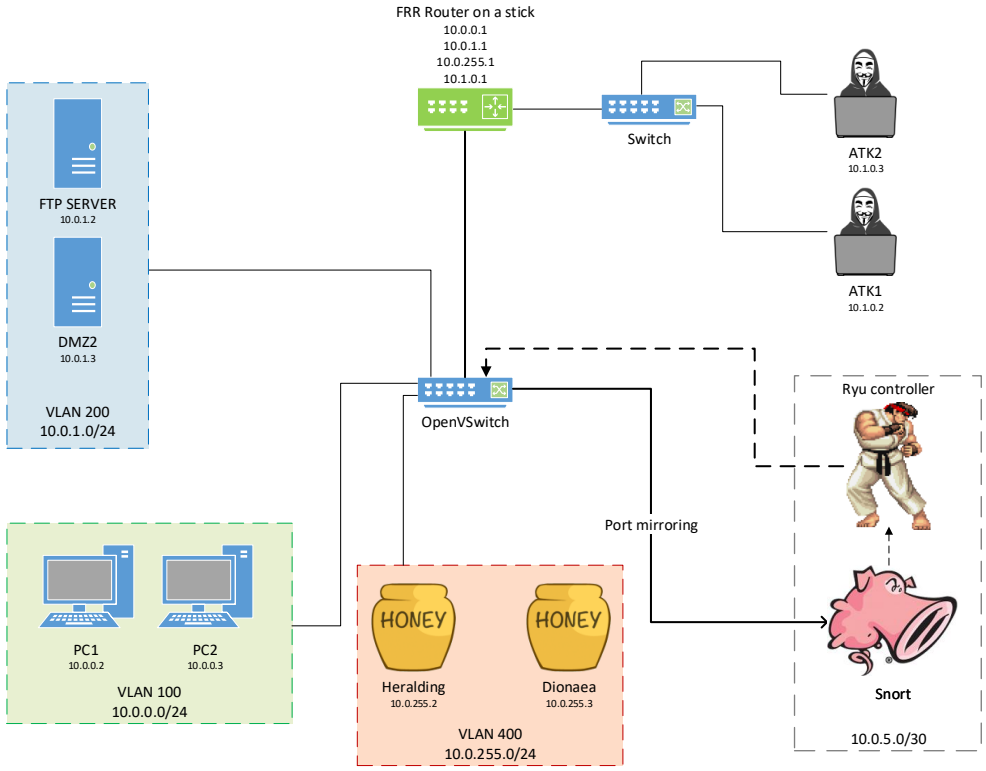


Figure 2.1: Topologia della rete simulata.

in una rete privata. Come anticipato, i due attaccanti sono collegati a uno switch semplice che a sua volta è collegato al router aziendale.

È presente una VLAN dedicata ai PC utilizzati dai dipendenti e un'altra VLAN dedicata a una pseudo-DMZ in cui sono presenti gli host che forniscono servizi all'esterno. I due honeypot hanno anche loro una propria VLAN. Il controller Ryu e Snort sono invece connessi mediante una loro LAN dedicata, questo previene che i pacchetti tra questi due host vengano a loro volta analizzati tramite il port mirroring.

L'host che contiene il controller Ryu in realtà comunica con l'OpenVSwitch mediante l'interfaccia di rete della macchina virtuale che esegue Containernet. Una limitazione di quest'ultimo infatti non permette di assegnare come controller un container connesso alla rete che viene simulata.

Host	Mac Address	IP	Subnet mask	VLAN	Port
R1		10.0.0.1	255.255.255.0	Trunk	S1-eth1
		10.0.1.1			
		10.0.255.1			
		10.1.0.1	255.255.255.0		S2-eth1
Snort	Port mirroring (Promiscous mode)				S1-eth2
		10.0.5.2	255.255.255.252		C0-eth0
Ryu controller		10.0.5.1	255.255.255.252		IDS-eth1
PC1		10.0.0.2	255.255.255.0	100	S1-eth3
PC2		10.0.0.3	255.255.255.0	100	S1-eth4
FTP server		10.0.1.2	255.255.255.0	200	S1-eth5
DMZ2		10.0.1.3	255.255.255.0	200	S1-eth6
Heralding	00:00:00:00:00:02	10.0.255.2	255.255.255.0	400	S1-eth7
Dionaea	00:00:00:00:00:03	10.0.255.3	255.255.255.0	400	S1-eth8
ATK1		10.1.0.2	255.255.255.0		S2-eth2
ATK2		10.1.0.3	255.255.255.0		S2-eth3

Table 2.1: Tabella riassuntiva della topologia. Gli indirizzi MAC ove non specificati sono assegnati casualmente. Nel caso dell'IDS con Snort i valori sono assegnati ma non rilevanti.

2.2 Configurazione di Snort come NIDS e relay degli alert

L'host con Snort installato ha il compito di ispezionare i pacchetti che vengono ricevuti tramite port mirroring e verificare la presenza di determinati pattern definiti dalle regole Snort. Più nel dettaglio sono state implementate regole per verificare la presenza di alcuni tipi di scanning TCP e di attacchi bruteforce su FTP. Inoltre è stata implementata anche una regola contro l'accesso a SSH da parte di un host esterno, nonché un pattern per verificare se un attaccante cerca di sfruttare una vulnerabilità di vsFTPd 2.3.4. In questo modo è stato possibile verificare il funzionamento delle regole contro una varietà di attacchi.

```
1 alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[1000] SCAN NULL"; flow:
    stateless; ack:0; flags:0; seq:0; classtype:attempted-recon; sid:1000;)
2 alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[1001] SCAN SYN FIN"; flow:
    stateless; flags:SF,12; classtype:attempted-recon; sid:1001;)
3 alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[1002] SCAN XMAS"; flow:
    stateless; flags:SRAFPU,12; classtype:attempted-recon; sid:1002;)
4 alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[1003] SCAN nmap XMAS"; flow:
    stateless; flags:FPU,12; classtype:attempted-recon; sid:1003;)
5 alert tcp $EXTERNAL_NET any -> $HOME_NET 22 (msg:"[1004] SSH port scan attempt from
     external net"; flow:stateless; classtype:attempted-recon; sid:1004;)
```

```
6 alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[1005] Special vsftpd backdoor
    exploit characters used for login"; content:"USER"; content:"|3A 29|"; classtype
    :suspicious-login; sid:1005;)
7 alert tcp $HOME_NET 21 -> $EXTERNAL_NET any (msg:"[1006] FTP Brute force attempt";
    pcre:"/500|530/i";threshold:type both, track by_src, count 5, seconds 10;  sid
    :1006; rev:5;)
```

Il messaggio della regola include il SID della regola stessa, questo è necessario per comunicare il tipo di violazione (e quindi l'azione necessaria) al controller Ryu. Per effettuare la comunicazione Snort logga il suo output su un file e uno script Python esterno (*pigrelay.py*) invia il contenuto di questo file al controller tramite socket TCP.
A livello di rete l'aspetto più importante da evidenziare è l'abilitazione della modalità promiscua sull'interfaccia di rete connessa allo switch. Ciò consente il passaggio alla CPU di tutto il traffico che viene osservato dall'interfaccia e quindi lo sniffing da parte di Snort.

2.3 Configurazione del Controller Ryu

Il controller Ryu deve assolvere principalmente tre compiti.

1. Effettuare il port mirroring del traffico verso il container di Snort.
2. Gestire le operazioni del tagging VLAN secondo il protocollo 802.1q.
3. Impostare eventuali regole di instradamento al ricevere di alert da parte di Snort realizzando la MTD.

2.3.1 Port mirroring

Il primo compito viene assolto abbastanza agevolmente, infatti basta aggiungere alle azioni dello switch OpenFlow l'invio del pacchetto verso la porta 2 ove è presente l'IDS.

2.3.2 VLAN 802.1q

A seconda della presenza dell'header 802.1q e della porta in ingresso vengono previste azioni diverse per la gestione della VLAN. Il comportamento complessivo viene riassunto in Fig. 2.2.

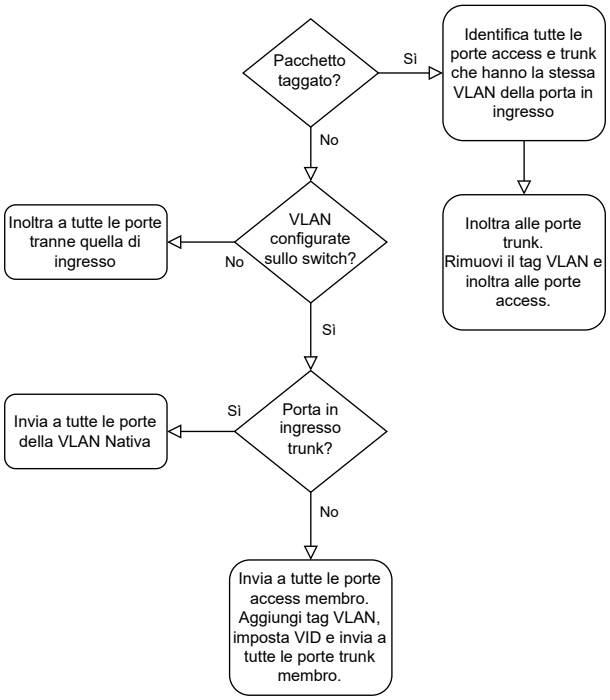


Figure 2.2: Gestione dei pacchetti VLAN.

Il comportamento illustrato è possibile grazie alla capacità dello switch OpenFlow di stabilire regole di matching (es.: presenza di un header, o di un determinato valore in un campo) e azioni corrispondenti.

2.3.3 Moving Target Defense Forwarding Rules

```
        self.honeypots = {
            "heralding": {
                'ip':'10.0.255.2',
                'mac':'00:00:00:00:00:02',
                'port': 7,
                'busy_services': [],
                'snort_sids': {
                    1004: {
                        'reverse': False,
                        'service': 'ssh'
                    },
                    1005: {
                        'reverse': False,
                        'service': 'ftp'
                    },
                    1006: {
                        'reverse': True,
                        'service': 'ftp'
                    }
                },
                'protected_routes': dict()
            },
            "dionaea": {
```

```
            'ip':'10.0.255.3',
            'mac':'00:00:00:00:00:03',
            'port': 8,
            'busy_services': [],
            'snort_sids': {
                1000: {
                    'reverse': False,
                    'service': 'all'
                },
                1001: {
                    'reverse': False,
                    'service': 'all'
                },
                1002: {
                    'reverse': False,
                    'service': 'all'
                },
                1003: {
                    'reverse': False,
                    'service': 'all'
                },
                1005: {
                    'reverse': False,
                    'service': 'ftp'
                },
                1006: {
                    'reverse': True,
                    'service': 'ftp'
                }
            },
            'protected_routes': dict()
        }
    }
```

La struttura dati fondamentale che permette la gestione degli attacchi è un dizionario che contiene un elenco degli honeypot con le loro informazioni (IP, MAC e porta), i servizi attualmente occupati, i SID di Snort ai quali reagire e quali servizi vengono occupati per ognuno di essi. È presente anche un campo reverse per quei casi in cui la regola per Snort è di più semplice ed efficace implementazione se monitora il traffico dalla rete interna verso quelle esterne.

Si è deciso che per un attacco di tipo scanning viene assegnato l'intero honeypot, in questo modo la fase di ricognizione di un attaccante avverrà su queste macchine e sarà monitorata. Se l'attaccante prende di mira un servizio specifico invece allora gli altri potranno accogliere ulteriori connessioni da parte di altri flussi malevoli.

```
def add_flow(self, datapath, priority, match, actions):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]

```

```
    mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=match, instructions=inst)
    datapath.send_msg(mod)
```

```
# ATTACKER -> HONEYPOT
#Match dei pacchetti con vlan_vid = vlan_victim e ip_src = attacker_ip e ip_dst = victim_ip
#Action: setta ip_dst = dst_pot["ip"] e mac_dst = dst_pot["mac"] e porta di output = dst_pot["port"]
match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,vlan_vid=vlan_victim|ofproto_v1_3.OFPVID_PRESENT,ipv4_src=
    attacker_ip, ipv4_dst=victim_ip)
actions = []
actions = [
    parser.OFPActionSetField(ipv4_dst=dst_pot["ip"]),
    parser.OFPActionSetField(eth_dst=dst_pot["mac"]),
    parser.OFPActionPopVlan(),
    parser.OFPActionOutput(dst_pot["port"])
    ]
self.add_flow(datapath, 102, match, actions)

#HONEYPOT -> ATTACKER
parser = datapath.ofproto_parser
self.vlan_members(datapath.id,port_victim,vlan_victim)
#Match dei pacchetti con ip_src = dst_pot["ip"] e ip_dst = attacker_ip
#Action: setta ip_src = victim_ip e mac_src = victim_mac e porta di output = 1
match = parser.OFPMatch(eth_type=ether_types.ETH_TYPE_IP,ipv4_src=dst_pot["ip"], ipv4_dst=attacker_ip)
actions = [
    parser.OFPActionSetField(eth_src = victim_mac),
    parser.OFPActionSetField(ipv4_src = victim_ip),
    parser.OFPActionOutput(1)
    ]
self.add_flow(datapath, 101, match, actions)
```

All'interno della libreria di Ryu è disponibile un modulo chiamato *snortlib* che permette la definizione della funzione *_dump_alert* che in ingresso riceve un evento ricevuto da Snort. Quando un alert viene ricevuto, il controller cerca un honeypot disponibile in cui uno dei SID gestiti corrisponda a quello dell'alert inviato da Snort. Fatto ciò vengono impostate due regole:

- **Attacker -> Honeypot:** in questa regola viene fatto il matching sull'IP di sorgente che è quello dell'attaccante e su quello di destinazione (della vittima), viene inoltre fatto un matching sul VID della VLAN (il pacchetto inviato a Snort è quello che ha appena attraversato la porta trunk e che proviene dal router-on-stick e che quindi possiede un tag VLAN). Come azioni vengono cambiati l'IP e il MAC di destinazione con quelli dell'honeypot, rimosso il tag VLAN e inoltrato il pacchetto alla porta dell'honeypot.
- **Honeypot -> Attacker:** in questa regola vogliamo mascherare il traffico di risposta dell'honeypot come una risposta della macchina vittima. Per fare ciò viene effettuato il match sull'IP sorgente che è quello dell'honeypot e su quello di destinazione che è quello

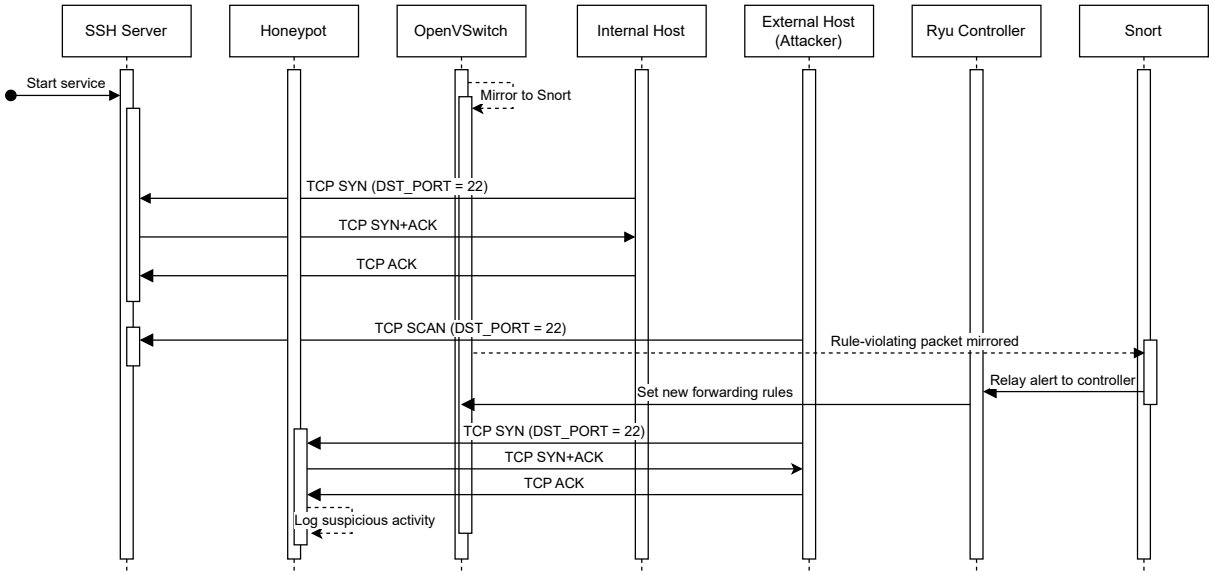


Figure 2.3: Funzionamento complessivo dello schema di MTD in caso d'attacco.

dell'attaccante. Come azione viene cambiati il MAC e l'ip sorgente con quelli della vittima e il pacchetto viene instradato verso il router.

2.4 Funzionamento complessivo

Nella Fig. 2.3 è possibile vedere un'illustrazione del funzionamento complessivo del sistema. Nel momento in cui un host esterno alla rete invia uno o più pacchetti che violano una regola di Snort, questo inoltra al controller Ryu l'alert generato. A sua volta il controller Ryu comunica con lo switch OpenFlow e imposta le due regole di forwarding viste precedentemente. Da questo punto in poi l'attaccante comunicherà con l'honeypot il quale terrà tracce delle attività dell'attaccante.

Chapter 3

Test e Analisi

Sono stati presi in esame i seguenti scenari per verificare il funzionamento della rete.

- **Traffico benigno locale e verso l'esterno della rete.**
- **Port scanning da un host esterno alla rete.**
- **Brute-force attack verso il server FTP.**
- Tentativo di accesso di un servizio privato (SSH) dall'esterno della rete.
- Sfruttamento di un exploit FTP.

Per ognuno di questi scenari vengono verificate le regole di instradamento presenti sullo switch OpenFlow e gli effetti di tale regole.

3.1 Traffico benigno locale

In questo caso si tratta di effettuare un semplice ping tra due host, ipotiziamo che $PC1$ effettui un ping verso $DMZ2$. In seguito al ping vediamo (col comando `ovs-ofctl dump-flows s1`) che sono state aggiunte quattro regole di instradamento sullo switch (Fig. 3.1). In particolare esse corrispondono al traffico dagli host ai rispettivi gateway sulle interfacce virtuali del router R1 e da quest'ultimo verso gli host. Per le prime viene fatto il matching in base alla porta in ingresso e al MAC di destinazione e come azioni viene aggiunto il tag VLAN e il pacchetto viene inoltrato verso il router oltre che verso Snort. Per i pacchetti che attraversano R1 e vanno

```
containernet> p1 ping 10.0.1.3
PING 10.0.1.3 (10.0.1.3) 56(84) bytes of data.
64 bytes from 10.0.1.3: icmp_seq=1 ttl=63 time=8.95 ms
64 bytes from 10.0.1.3: icmp_seq=2 ttl=63 time=0.342 ms
64 bytes from 10.0.1.3: icmp_seq=3 ttl=63 time=0.071 ms
^C
--- 10.0.1.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2032ms
rtt min/avg/max/mdev = 0.071/3.119/8.946/4.121 ms
containernet> sh ovs-ofctl dump-flows s1
 cookie=0x0, duration=281.234s, table=0, n_packets=4, n_bytes=352, priority=1,in_port="s1-eth1",dl_vlan=100,dl_dst=f6:e2:b7:39:99:7e
 actions=strip_vlan,output:"s1-eth3",output:"s1-eth2"
 cookie=0x0, duration=281.228s, table=0, n_packets=3, n_bytes=250, priority=1,in_port="s1-eth1",dl_vlan=200,dl_dst=66:f5:ba:46:c1:86
 actions=strip_vlan,output:"s1-eth6",output:"s1-eth2"
 cookie=0x0, duration=281.233s, table=0, n_packets=3, n_bytes=238, priority=1,in_port="s1-eth3",dl_dst=3a:97:1f:e2:b2:6c actions=mod
_vlan_vid:100,output:"s1-eth1",output:"s1-eth2"
 cookie=0x0, duration=281.230s, table=0, n_packets=4, n_bytes=336, priority=1,in_port="s1-eth6",dl_dst=3a:97:1f:e2:b2:6c actions=mod
_vlan_vid:200,output:"s1-eth1",output:"s1-eth2"
 cookie=0x0, duration=323.003s, table=0, n_packets=6, n_bytes=376, priority=0 actions=CONTROLLER:65535
containernet>
```

Figure 3.1: Regole instaurate in seguito al ping tra PC1 e DMZ2.

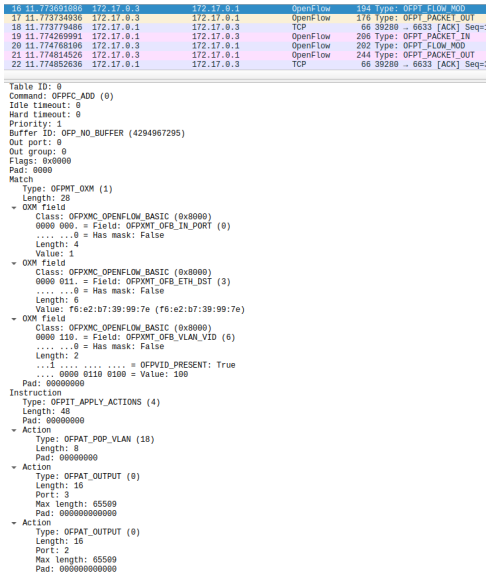


Figure 3.2: Contenuto del pacchetto OFPT_MOD_FLOW catturato da EdgeShark

verso PC1 e DMZ2 invece il matching oltre che sulla porta di ingresso e il MAC di destinazione viene fatto anche sul tag VLAN; in questo caso il tag VLAN viene rimosso. Ai fini di verifica e di debugging è stato utilizzato il tool Edgeshark. Si tratta di una estensione di Wireshark che permette la cattura del traffico generato dai container Docker. Possiamo vedere che in questo scenario viene inviato un pacchetto di tipo OFPT_MOD_FLOW (Fig. 3.2) dal controller verso lo switch con la lista dei match e delle azioni da impostare. Possiamo anche verificare il funzionamento del port mirroring verso Snort (Fig. 3.3).

3.2 Port scanning

```
1 alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[1000] SCAN NULL"; flow:stateless; ack:0; flags:0; seq
      :0; classtype:attempted-recon; sid:1000;)
```

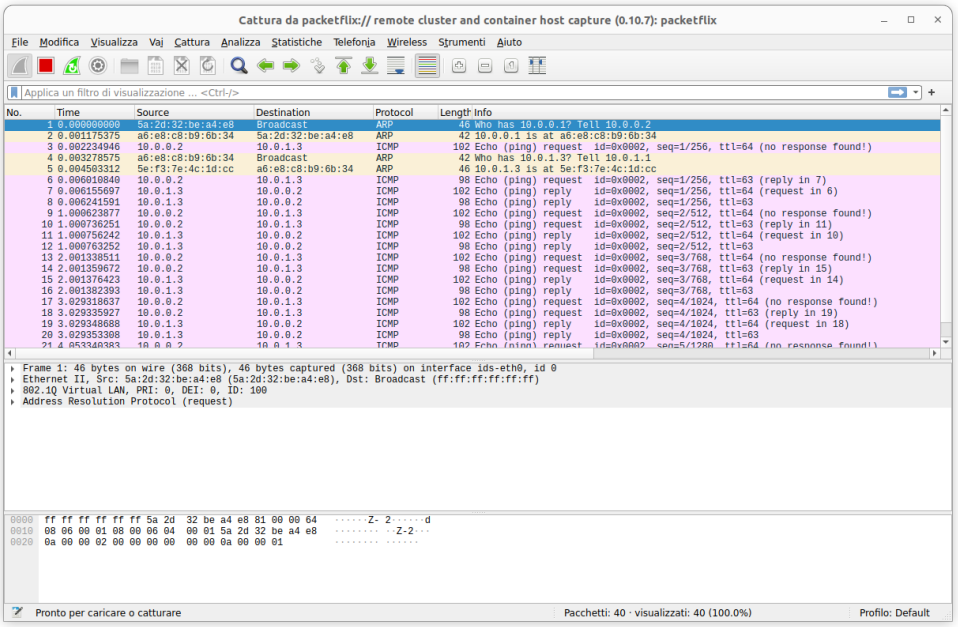


Figure 3.3: Cattura dei pacchetti ICMP scambiati tra PC1 e DMZ2 e inviati a Snort.

```
2 alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[1001] SCAN SYN FIN"; flow:stateless; flags:SF,12;
      classtype:attempted-recon; sid:1001;)
3 alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[1002] SCAN XMAS"; flow:stateless; flags:SRAFPU
      ,12; classtype:attempted-recon; sid:1002;)
4 alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"[1003] SCAN nmap XMAS"; flow:stateless; flags:FPU
      ,12; classtype:attempted-recon; sid:1003;)
```

Come primo tipo di attività malevola cerchiamo di individuare una ricognizione di tipo port scanning. Le regole Snort verificano la presenza di flag TCP anomali e generano un alert che viene inviato al controller Ryu (Fig. 3.4c). In Fig. 3.4a possiamo vedere che nel primo scan effettuato con NMAP la porta SSH individuata è chiusa (DMZ2 infatti non ha un server SSH in ascolto), mentre alla seconda scansione la porta è aperta, questo perché il traffico è ora diretto verso l'honeypot che ha a disposizione un server SSH. In Fig. 3.4b vediamo le regole che sono state impostate come precedentemente menzionato in 2.3.3.

3.3 FTP Bruteforce

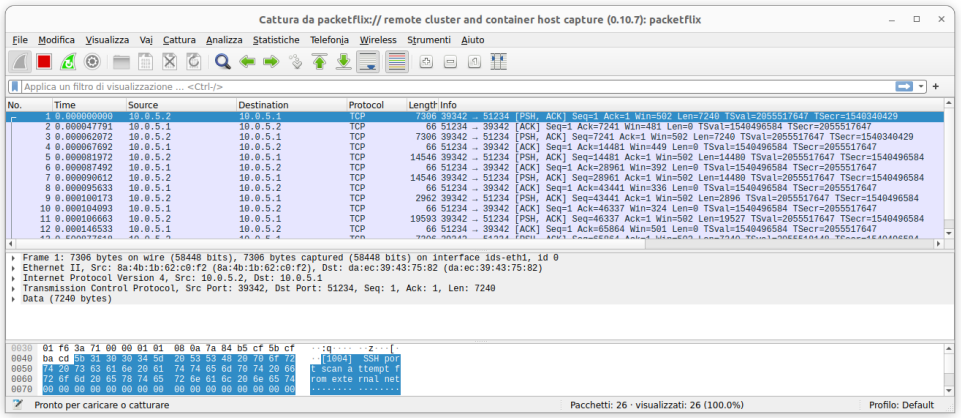
Per determinare una regola Snort efficace contro un attacco bruteforce ne è stata prima catturata una traccia tra un ipotetico attaccante e un server. In particolare in Fig. 3.5 si vede come il server invii errori di tipo 530 e occasionalmente 500 a seguito dei vari tentativi di accesso. Perciò

```
containernet> atk1 nmap -sX -p22 10.0.1.3

Starting Nmap 7.93 ( https://nmap.org ) at 2024-04-28 21:46 UTC
Nmap scan report for 10.0.1.3
Host is up (0.0092s latency).

PORT   STATE  SERVICE
22/tcp closed ssh

Nmap done: 1 IP address (1 host up) scanned in 13.23 seconds
containernet> atk1 nmap -sX -p22 10.0.1.3

Starting Nmap 7.93 ( https://nmap.org ) at 2024-04-28 21:47 UTC
Nmap scan report for 10.0.1.3
Host is up (0.0025s latency).

PORT   STATE         SERVICE
22/tcp open|filtered ssh

Nmap done: 1 IP address (1 host up) scanned in 13.38 seconds
containernet>
```

(a) Console di NMAP.

```
containernet> sh ovs-ofctl dump-flows s1
 cookie=0x0, duration=138.698s, table=0, n_packets=7, n_bytes=398, priority=102,ip,dl_vlan=200,nw_src=10.1.0.2,nw_dst=10.0.1.3 actions=mod_nw_dst:10.0.255.2,mod_dl_dst:00:
00:00:00:00:02,strip_vlan,output:"s1-eth7"
 cookie=0x0, duration=138.698s, table=0, n_packets=4, n_bytes=208, priority=101,ip,nw_src=10.0.255.2,nw_dst=10.1.0.2 actions=mod_dl_src:0e:8a:bb:1a:ef:d2,mod_nw_src:10.0.1
.3,output:"s1-eth1"
 cookie=0x0, duration=152.970s, table=0, n_packets=6, n_bytes=300, priority=1,in_port="s1-eth6",dl_dst=de:c6:2d:59:23:fc actions=mod_vlan_vid:200,output:"s1-eth1",output:"
s1-eth2"
 cookie=0x0, duration=152.967s, table=0, n_packets=2, n_bytes=104, priority=1,in_port="s1-eth1",dl_vlan=200,dl_dst=0e:8a:bb:1a:ef:d2 actions=strip_vlan,output:"s1-eth6",ou
tput:"s1-eth2"
 cookie=0x0, duration=129.288s, table=0, n_packets=0, n_bytes=0, priority=1,in_port="s1-eth1",dl_vlan=400,dl_dst=00:00:00:00:00:02 actions=strip_vlan,output:"s1-eth7",outp
ut:"s1-eth2"
 cookie=0x0, duration=295.756s, table=0, n_packets=8, n_bytes=400, priority=0 actions=CONTROLLER:65535
containernet>
```

(b) Regole di flusso instaurate.



(c) Messaggi inviati da Snort al controller Ryu.

Figure 3.4: Tentativo di XMAS scan su porta 21 da parte di ATK1.

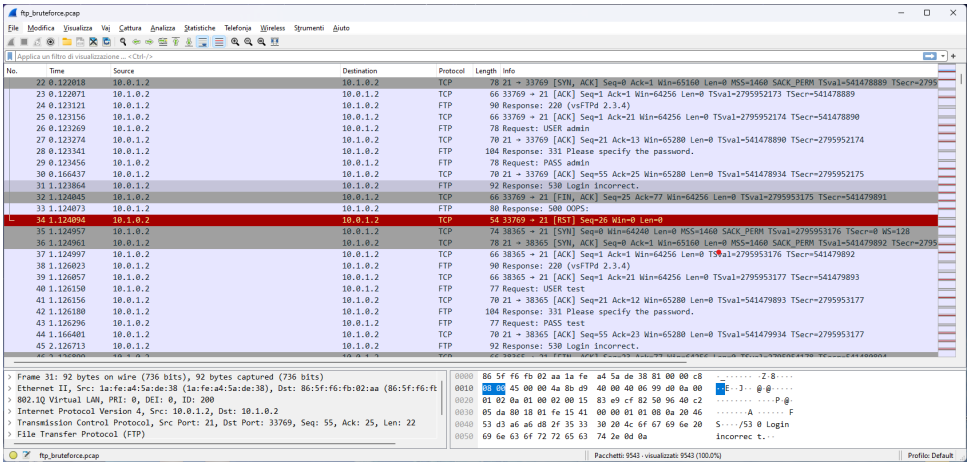


Figure 3.5: Cattura di un attacco bruteforce FTP..

si è deciso di creare una regola che venga azionata nel caso di un numero eccessivo di tentativi di accesso in un determinato intervallo di tempo. In particolare vengono cercati i codici 500 e 530 inviati dal server FTP e se ne vengono individuati più di 5 in 10 secondi allora viene azionato il meccanismo di moving target defence.

```
alert tcp $HOME_NET 21 -> $EXTERNAL_NET any (msg:"[1006] FTP Brute force attempt";
    pcre:"/500|530/i";threshold:type both, track by_src, count 5, seconds 10;  sid
    :1006; rev:5;)
```

Gli attaccanti utilizzano il tool `auxiliary/scanner/ftp/ftp_login` di Metasploit e viene fornito in un file di testo una lista di coppie username-password da usare durante l'attacco.
I due honeypot agiscono in modo diverso ai tentativi di connessione tramite FTP. Heralding effettua il logging (in formato .csv o .json) della sessione e dei dati di accesso utilizzati da un attacco ma non permette a un attaccante di eseguire alcun tipo di operazioni.

```
{
    "timestamp": "2024-04-29 22:29:43.693522",
    "duration": 0,
    "session_id": "472179eb-8b2e-4d05-b655-9696bc4fbd1b",
    "source_ip": "10.1.0.2",
    "source_port": 42581,
    "destination_ip": "10.0.255.2",
    "destination_port": 21,
    "protocol": "ftp",
    "num_auth_attempts": 1,
```

```
    "auth_attempts": [
        {"timestamp": "2024-04-29 22:29:43.694003",
        "username": "clamav1",
        "password": "clamav1"}
        ],
    "session_ended": true,
    "auxiliary_data": {}
}
```

Invece Dionaea non solo permette di effettuare un vero e proprio accesso, ma consente all'attaccante di operare (seppur limitatamente) all'interno di una subdirectory del filesystem dedicata da parte dell'honeypot. I dati vengono registrati in un database sqlite Fig. 3.6.

	conne...	connecti...	connecti...	connecti...	connection_timestamp	connecti...	connecti...	local_host	local_port	remote_h...
1	1	accept	tcp	ftpd	1714600995.709326	1		10.0.255.3	21	10.1.0.3
2	2	accept	tcp	ftpd	1714601011.734727	2		10.0.255.3	21	10.1.0.3
3	3	accept	tcp	ftpd	1714601027.7596989	3		10.0.255.3	21	10.1.0.3
4	4	accept	tcp	ftpd	1714601043.7755094	4		10.0.255.3	21	10.1.0.3
5	5	accept	tcp	ftpd	1714601059.8002608	5		10.0.255.3	21	10.1.0.3
6	6	accept	tcp	ftpd	1714601075.826325	6		10.0.255.3	21	10.1.0.3
7	7	accept	tcp	ftpd	1714601091.8527076	7		10.0.255.3	21	10.1.0.3
8	8	accept	tcp	ftpd	1714601107.8763907	8		10.0.255.3	21	10.1.0.3
9	9	accept	tcp	ftpd	1714601123.8992302	9		10.0.255.3	21	10.1.0.3
10	10	accept	tcp	ftpd	1714601139.9212897	10		10.0.255.3	21	10.1.0.3
11	11	accept	tcp	ftpd	1714601155.933559	11		10.0.255.3	21	10.1.0.3
12	12	accept	tcp	ftpd	1714601179.1627185	12		10.0.255.3	21	10.1.0.3
13	13	accept	tcp	ftpd	1714601195.1888475	13		10.0.255.3	21	10.1.0.3

(a) Sessioni Dionaea.

	login	connection	login_use...	login_pas...
1	1	1	root	sleeper
2	2	2	guest	guest
3	3	3	ghost	123456
4	4	4	magnos	magnos
5	5	5	aaron	aaron
6	6	6	jun	jun
7	7	7	rebecca	rebecca
8	8	8	einstein	einstein
9	9	9	anna	anna
10	10	10	sara	sara
11	11	12	anna	anna
12	12	13	root	root
13	13	15	root	sleeper
14	14	16	guest	guest
15	15	17	ghost	123456
16	16	18	magnos	magnos

(b) Login Dionaea.

Figure 3.6: Attacco bruteforce FTP registrato da Dionaea.

Chapter 4

Conclusioni

Il presente studio ha esplorato l'integrazione e la configurazione di una rete SDN utilizzando il controller Ryu, Snort come IDS/IPS, e l'impiego di honeypots come Heralding e Dionaea, configurati per migliorare la sicurezza di rete mediante tecniche di Moving Target Defense (MTD). Questa configurazione ha permesso di dimostrare l'efficacia delle SDN nel rispondere dinamicamente agli attacchi informatici, orientando il traffico sospetto verso sistemi honeypot specificamente predisposti per l'analisi e la raccolta di informazioni su potenziali minacce.

4.1 Principali risultati e validità dell'approccio

Il setup ha validato l'ipotesi che le SDN, grazie alla loro flessibilità nella gestione dei flussi di dati, permettono una difesa proattiva contro gli attacchi, riducendo la superficie di attacco esposta e confondendo gli attaccanti. La capacità di modificare le rotte di rete in tempo reale si è rivelata fondamentale nel deviare i flussi di traffico malevolo verso gli honeypots, evitando così danni ai sistemi critici. Inoltre, l'uso di Snort ha permesso una dettagliata analisi del traffico per la rilevazione tempestiva di comportamenti sospetti.

4.2 Limitazioni del sistema implementato

Nonostante i successi ottenuti, l'implementazione ha mostrato limitazioni, principalmente legate alla complessità della configurazione e della gestione delle regole di sicurezza, che richiedono una

conoscenza approfondita e specifica. Inoltre, la simulazione ha operato su una scala relativamente ridotta e controllata.

4.3 Implicazioni future e aree di miglioramento

Guardando al futuro, è auspicabile esplorare l'integrazione di soluzioni basate su intelligenza artificiale e machine learning per automatizzare ulteriormente la risposta agli attacchi e migliorare la capacità di prevedere e prevenire minacce emergenti. Un'altra area di miglioramento potrebbe essere l'ottimizzazione delle prestazioni di rete per garantire che le misure di sicurezza non compromettano l'efficienza operativa.

In conclusione, questo progetto ha confermato il potenziale delle SDN e delle tecniche di MTD nel rafforzare la sicurezza di rete. Attraverso continui sviluppi e raffinamenti, tale approccio promette di offrire una strategia di difesa cybernetica altamente efficace e adattabile, capace di rispondere alle mutevoli minacce del panorama informatico contemporaneo.