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

Academic year 2023/2024

Professor: **Simon Pietro Romano**
Team members:
Giampetraglia Federica M63001358
Vallefuoco Roberto M63001443
Carillo Raffaele M63001321
De Simone Anna M551278

Contents

Introduction

In recent years, the increasing complexity of network architectures and the growing need for high-security standards have driven the development of innovative solutions such as Software-Defined Networking (SDN). This project aims to explore the implementation of an SDN-based network using the Ryu controller, integrated with Snort, a well-known Intrusion Detection and Prevention System (IDS/IPS). The technologies are deployed within a Dockerized environment, leveraging Containernet—a variant of Mininet—to simulate a complex network that enables effective security testing.

The primary objective of this project is to demonstrate how the flexibility of SDN can enhance network security through strategies such as Moving Target Defense and the use of honeypots. These techniques increase the complexity of attack vectors for adversaries, thereby reducing known vulnerabilities.

This documentation provides a comprehensive overview of the development environment, configuration, and implementation of the solution, along with the testing process and potential future enhancements. The document follows a logical progression from theory to practice, clearly outlining the implementation steps and validation of the proposed solutions.

Chapter 1

Background

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

Software Defined Networking (SDN) is a network architecture that separates the control plane from the data plane, enhancing the flexibility and manageability of networks. This separation allows administrators to centralize and simplify traffic policy management, thereby reducing errors and improving responsiveness to changes. Moving Target Defense (MTD) techniques involve modifying the network topology to disrupt and hinder attack activities. In particular, these techniques enable the automated modification of traffic flows, facilitating both analysis and the collection of Cyber Threat Intelligence (CTI). SDN can be employed as a tool to implement MTD strategies by redirecting malicious traffic toward honeypots, dynamically altering the IP addresses of targeted hosts, or simply creating confusion within the target network by masking and modifying host IP addresses. Honeypots serve as traps that not only divert and slow down attacks but also provide valuable information about the attackers, thereby enhancing defense strategies.

1.2 The Ryu Controller

Ryu is an open-source SDN controller that offers a programmable platform for the development of complex networks. It is popular for its compatibility with various network standards, including OpenFlow—a protocol that enables direct interaction between the controller and network switches (used by the controller to manage packet flows in the network through the dynamic modification of switch flow tables). The controller software is easily programmable in Python, allowing developers to define network behaviors through simple scripts based on the analysis of incoming packets.

1.3 Snort as IDS/IPS

Snort is an intrusion detection and prevention system used to monitor network traffic and analyze incoming/outgoing packets for attack signatures. Its real-time traffic analysis involves decoding each incoming packet by interpreting protocols and various data structures. After this initial decoding phase, the decoded packet is compared against a set of user-defined rules (which are useful for updating signatures to recognize new attacks) to detect attack patterns. Snort can be configured in three different modes:

- **Sniffer:** In this mode, Snort is used solely for capturing and logging packets, allowing for subsequent offline analysis of the network traffic.
- **IDS (Intrusion Detection System):** In this mode, when specific attack patterns (defined in a rule database) are detected, alerts are generated that can then be managed by a network controller (in our case, alerts are sent via a socket from Snort to the Ryu controller).
- **IPS (Intrusion Prevention System):** In this configuration, Snort actively blocks traffic upon detecting an intrusion, playing a more active role compared to merely analyzing traffic.

1.4 Containernet

Containernet is a modified version of Mininet (a network simulator) that adds support for creating Docker containers within simulated network topologies. Similar to Mininet, this tool allowed us to create a sufficiently complex network topology (simulating a corporate network along with external nodes) using a Python script, in which we defined switches, VLANs, nodes, and links between various devices.

1.5 Heralding

The first container utilized is Heralding, a low-interaction honeypot. In this setup, various services are not fully implemented; instead, only the responses of the different software services are emulated. As a result, incoming connections are never actually accepted, which allows the system to log and record all access attempts by a malicious user without the risk of the user exploiting any software vulnerabilities—since the network service is not genuinely installed or operational on the honeypot. The emulated services include:

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

1.6 Dionaea

Dionaea is a low-interaction honeypot. Unlike Heralding, its level of interactivity is considerably higher. It not only allows an attacker to access services using vulnerable credentials but also permits them to perform certain operations associated with the specific attack vector, and, where applicable, simulates the presence of a small filesystem. Dionaea logs these activities in a relational SQLite database. The emulated services include:

- EPMAP
- FTP
- HTTP

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

In addition to this extensive collection of protocols, Dionaea offers a range of automated handlers, including sending HTTP requests, storing captured samples on Amazon AWS S3 and HPFEEDS, analyzing samples via VirusTotal, and logging events in various formats.

Chapter 2

Background

2.1 Software Defined Networking (SDN) and Moving Target Defense (MTD)

Software Defined Networking (SDN) is a network architecture that separates the control plane from the data plane, enhancing the flexibility and manageability of networks. This separation allows administrators to centralize and simplify traffic policy management, thereby reducing errors and improving responsiveness to changes. Moving Target Defense (MTD) techniques involve modifying the network topology to disrupt and hinder attack activities. In particular, these techniques enable the automated modification of traffic flows, facilitating both analysis and the collection of Cyber Threat Intelligence (CTI). SDN can be employed as a tool to implement MTD strategies by redirecting malicious traffic toward honeypots, dynamically altering the IP addresses of targeted hosts, or simply creating confusion within the target network by masking and modifying host IP addresses. Honeypots serve as traps that not only divert and slow down attacks but also provide valuable information about the attackers, thereby enhancing defense strategies.

2.2 The Ryu Controller

Ryu is an open-source SDN controller that offers a programmable platform for the development of complex networks. It is popular for its compatibility with various network standards, including OpenFlow—a protocol that enables direct interaction between the controller and network switches (used by the controller to manage packet flows in the network through the dynamic modification of switch flow tables). The controller software is easily programmable in Python, allowing developers to define network behaviors through simple scripts based on the analysis of incoming packets.

2.3 Snort as IDS/IPS

Snort is an intrusion detection and prevention system used to monitor network traffic and analyze incoming/outgoing packets for attack signatures. Its real-time traffic analysis involves decoding each incoming packet by interpreting protocols and various data structures. After this initial decoding phase, the decoded packet is compared against a set of user-defined rules (which are useful for updating signatures to recognize new attacks) to detect attack patterns. Snort can be configured in three different modes:

- **Sniffer:** In this mode, Snort is used solely for capturing and logging packets, allowing for subsequent offline analysis of the network traffic.
- **IDS (Intrusion Detection System):** In this mode, when specific attack patterns (defined in a rule database) are detected, alerts are generated that can then be managed by a network controller (in our case, alerts are sent via a socket from Snort to the Ryu controller).
- **IPS (Intrusion Prevention System):** In this configuration, Snort actively blocks traffic upon detecting an intrusion, playing a more active role compared to merely analyzing traffic.

2.4 Containernet

Containernet is a modified version of Mininet (a network simulator) that adds support for creating Docker containers within simulated network topologies. Similar to Mininet, this tool allowed us to create a sufficiently complex network topology (simulating a corporate network along with external nodes) using a Python script, in which we defined switches, VLANs, nodes, and links between various devices.

2.5 Heralding

The first container utilized is Heralding, a low-interaction honeypot. In this setup, various services are not fully implemented; instead, only the responses of the different software services are emulated. As a result, incoming connections are never actually accepted, which allows the system to log and record all access attempts by a malicious user without the risk of the user exploiting any software vulnerabilities—since the network service is not genuinely installed or operational on the honeypot. The emulated services include:

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

2.6 Dionaea

Dionaea is a low-interaction honeypot. Unlike Heralding, its level of interactivity is considerably higher. It not only allows an attacker to access services using vulnerable credentials but also permits them to perform certain operations associated with the specific attack vector, and, where applicable, simulates the presence of a small filesystem. Dionaea logs these activities in a relational SQLite database. The emulated services include:

- EPMAP
- FTP
- HTTP

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

In addition to this extensive collection of protocols, Dionaea offers a range of automated handlers, including sending HTTP requests, storing captured samples on Amazon AWS S3 and HPFEEDS, analyzing samples via VirusTotal, and logging events in various formats.

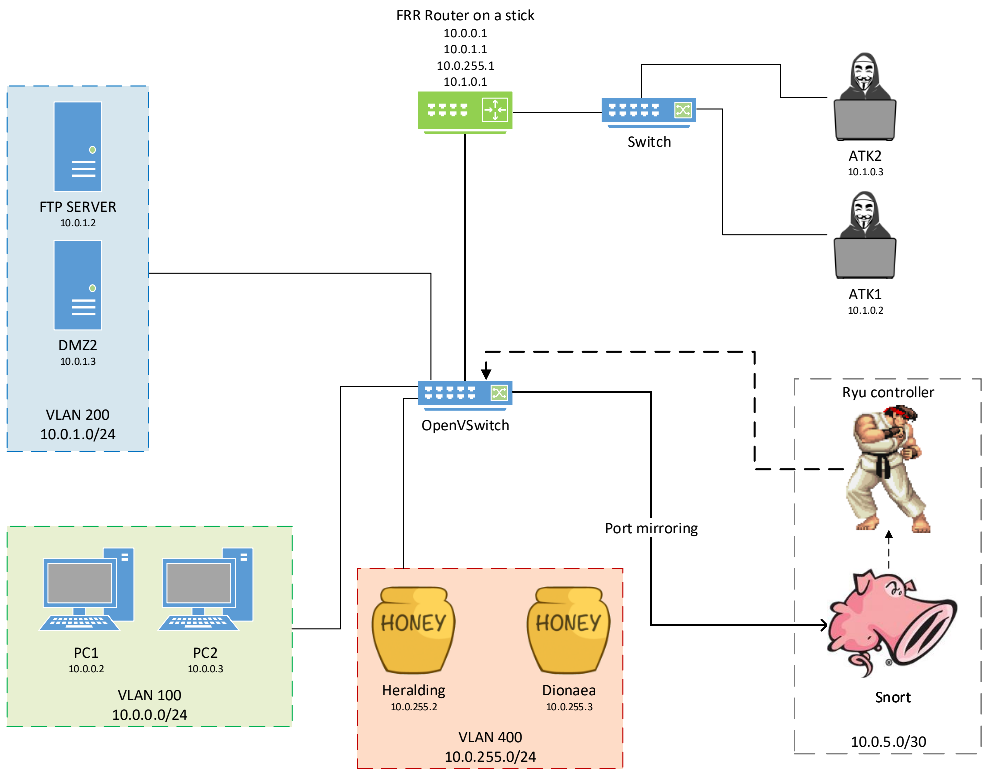


Figure 2.1: Simulated network topology.

The envisioned scenario is that of a typical corporate network. Several subnets are allocated for different purposes; in a standard enterprise network, hosts providing external services are usually separated from those used by internal personnel. In addition, a set of hosts dedicated to network management is typically present. In our setup, there is also a dedicated subnet for collecting Cyber Threat Intelligence (CTI). For simplicity, we assume that the attackers are connected via a switch directly to the main router.

2.7 Network Topology

The network topology consists of three VLANs connected to an OpenVSwitch, which in turn is connected to a router following the classic router-on-a-stick design. The router acts as the default gateway for all hosts in the topology. Static routes have been defined on the router for each of the connected subnets. The host configuration details are provided in Table 2.1.

One of the switch ports is dedicated to Snort, which operates as an IDS. Upon detecting an

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: Summary table of the topology. MAC addresses not specified are assigned randomly. In the case of the IDS with Snort, the values are assigned but not relevant.

anomaly, Snort sends an alert to the Ryu controller. The controller then modifies the flow installation rules on the switch so that malicious traffic is redirected to the honeypots.

The corporate network uses a /16 network mask; although the topology's requirements are much smaller, the network still utilizes an address range reserved for private network use. As mentioned earlier, the two attackers are connected to a simple switch that is in turn connected to the corporate router.

There is a VLAN dedicated to employee PCs and another VLAN assigned to a pseudo-DMZ, where hosts providing external services are located. The two honeypots have their own VLAN as well. The Ryu controller and Snort are connected via their own dedicated LAN, which prevents packets exchanged between these two hosts from being analyzed by port mirroring.

The host running the Ryu controller actually communicates with the OpenVSwitch through the network interface of the virtual machine running Containernet. A limitation of Containernet is that it does not allow assigning a controller that is hosted on a container connected to the simulated network.

2.8 Configuration of Snort as NIDS and Alert Relay

The host running Snort is responsible for inspecting packets received via port mirroring and checking for specific patterns defined by Snort rules. More specifically, rules have been implemented to detect certain types of TCP scans and FTP brute-force attacks. Additionally, there is a rule that prevents SSH access from external hosts, as well as a pattern to detect attempts to exploit the vsFTPd 2.3.4 vulnerability. This setup allows us to verify the functionality of the rules against a variety of attack types.

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

The rule message includes the SID (Snort ID) of the rule itself, which is necessary to communicate the type of violation—and therefore the required action—to the Ryu controller. To facilitate communication, Snort logs its output to a file, and an external Python script (*pigrelay.py*) sends the file's contents to the controller via a TCP socket.

At the network level, it is important to highlight that the network interface connected to the switch is set to promiscuous mode. This configuration allows the CPU to process all traffic observed by the interface, enabling Snort to perform its packet sniffing effectively.

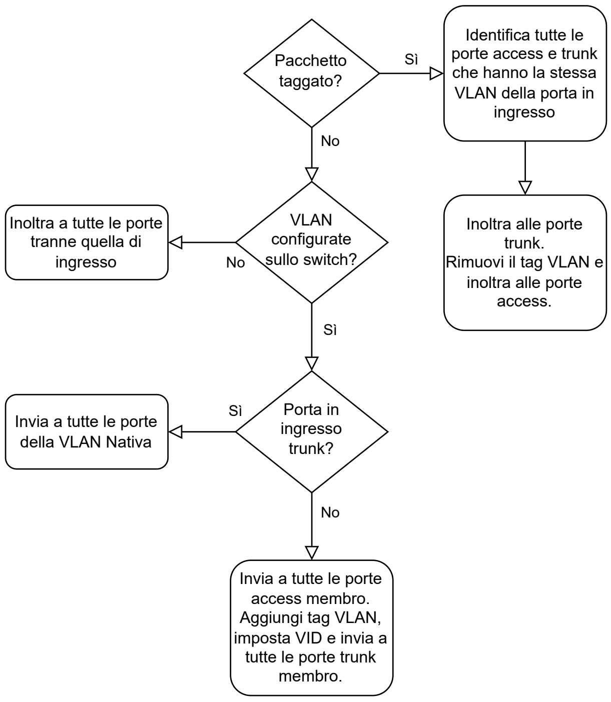

Pacchetto taggato?
Sì
Identifica tutte le porte access e trunk che hanno la stessa VLAN della porta in ingresso
No
Inoltra alle porte trunk. Rimuovi il tag VLAN e inoltra alle porte access.
VLAN configurate sullo switch?
No
Inoltra a tutte le porte tranne quella di ingresso
Sì
Porta in ingresso trunk?
Sì
Invia a tutte le porte della VLAN Nativa
No
Invia a tutte le porte access membro. Aggiungi tag VLAN, imposta VID e invia a tutte le porte trunk membro.

Figure 2.2: VLAN packet handling.

2.9 Ryu Controller Configuration

The Ryu controller is tasked with three primary functions:

1. Implementing port mirroring of traffic to the Snort container.
2. Managing VLAN tagging operations according to the 802.1q protocol.
3. Setting up appropriate forwarding rules upon receiving alerts from Snort to implement the Moving Target Defense (MTD) strategy.

2.9.1 Port Mirroring

The first function is straightforward. It involves adding an action in the OpenFlow switch to send a copy of each packet to port 2, where the IDS is connected.

2.9.2 802.1q VLAN

Depending on the presence of the 802.1q header and the ingress port, different actions are taken to manage the VLAN. The overall behavior is summarized in Fig. 2.2. This functionality is made possible by the OpenFlow switch's ability to establish matching rules (e.g., checking for the presence of a header or a specific field value) and to apply corresponding actions.

2.9.3 Moving Target Defense Forwarding Rules

```
'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() } }
```

The core data structure that enables attack management is a dictionary containing a list of honeypots with their information (IP, MAC, and port), the services currently in use, the Snort SIDs to which to react, and the services affected by each SID. A field named `reverse` is included for cases in which monitoring traffic from the internal network to external networks is more effective for a given Snort rule.

For scanning-type attacks, the entire honeypot is assigned, ensuring that the attacker's reconnaissance phase takes place on these monitored machines. For attacks targeting a specific service, other honeypots may still be available to handle additional malicious connections.

```

lua
Copia
Modifica
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 packets with vlan_vid = vlan_victim, ipv4_src = attacker_ip, and ipv4_dst = victim_ip
Action: set ipv4_dst = dst_pot["ip"], set eth_dst = dst_pot["mac"], remove VLAN tag, and output packet to 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 = [ 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
Reconfigure VLAN membership for the victim port
parser = datapath.ofproto_parser self.vlan_members(datapath.id, port_victim, vlan_victim)

Match packets with ipv4_src = dst_pot["ip"] and ipv4_dst = attacker_ip
Action: set eth_src = victim_mac, set ipv4_src = victim_ip, and output packet to port 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)
```

Within the Ryu library, a module called `snortlib` is available, which defines the function `_dump_alert` that receives an event from Snort. When an alert is received, the controller

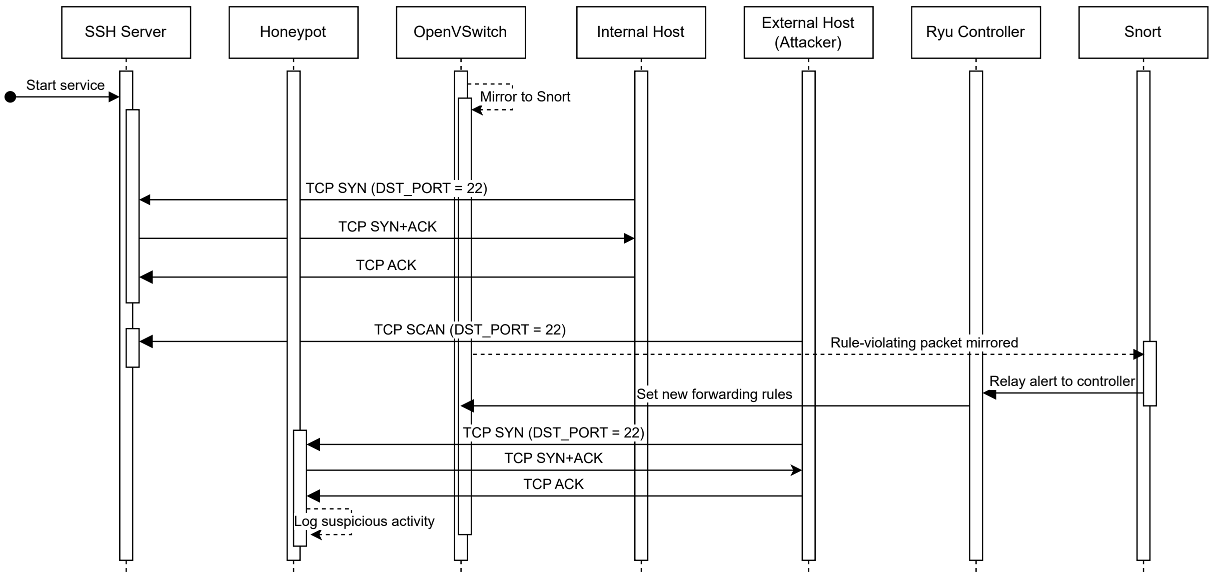


Figure 2.3: Overall operation of the MTD scheme during an attack.

searches for an available honeypot with a managed SID that matches the alert sent by Snort. Once a match is found, two rules are installed:

- **Attacker → Honeypot:** This rule matches packets based on the source IP (the attacker) and destination IP (the victim), along with the VLAN ID (the packet sent to Snort is one that has just passed through the trunk port and originates from the router-on-a-stick, and thus carries a VLAN tag). The actions performed include replacing the destination IP and MAC with those of the honeypot, removing the VLAN tag, and forwarding the packet to the honeypot's port.
- **Honeypot → Attacker:** This rule masks the honeypot's response to appear as if it is coming from the victim. It matches packets with the source IP of the honeypot and the destination IP of the attacker. The actions include changing the source MAC and IP to those of the victim and routing the packet to the router.

2.10 Overall Functionality

Figure 2.3 illustrates the complete operation of the system. When an external host sends one or more packets that violate a Snort rule, Snort sends the generated alert to the Ryu controller. In turn, the Ryu controller communicates with the OpenFlow switch to install the two forwarding rules described above. From that point onward, the attacker communicates with the honeypot, which logs all the attacker's activities for further analysis.

Chapter 3

Tests and Analysis

The following scenarios were examined to verify the proper functioning of the network:

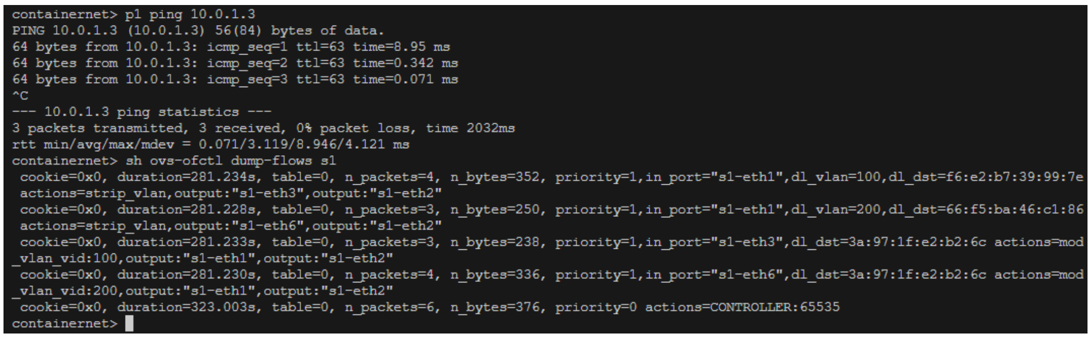
- **Local and external benign traffic.**
- **Port scanning from an external host.**
- **Brute-force attack against the FTP server.**
- **Unauthorized access attempt to a private service (SSH) from outside the network.**
- **Exploitation of an FTP exploit.**

For each of these scenarios, the OpenFlow switch's routing rules and their effects were carefully verified.

3.1 Local Benign Traffic

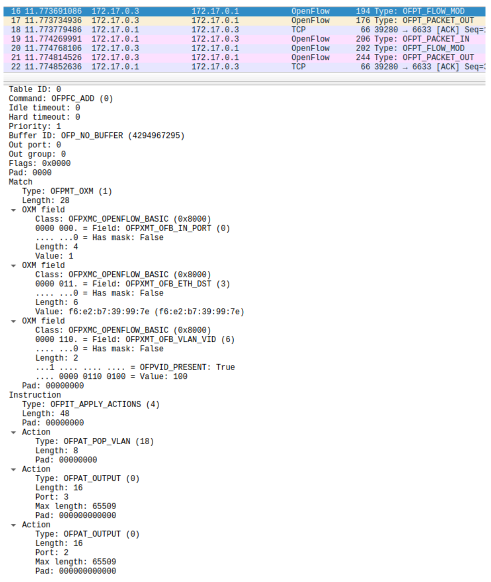
In this test, a simple ping is executed between two hosts; for instance, assume that `PC1` pings `DMZ2`.

After the ping, using the command `ovs-ofctl dump-flows s1`, we can see that four forwarding rules have been added to the switch (see Fig. 3.1). In particular, these rules correspond to traffic from the hosts to their respective gateways via the virtual interfaces of router R1, and from R1 back to the hosts. For the first set, matching is based on the ingress port and the



```
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: Flow rules installed as a result of the ping between `PC1` and `DMZ2`.



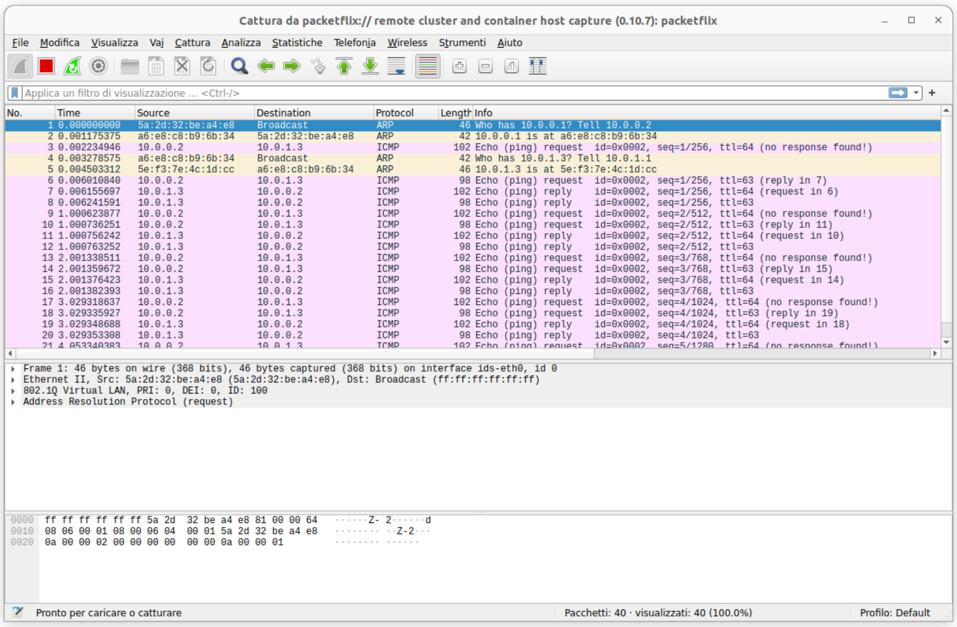
```
16 11.773691086  172.17.0.3    172.17.0.1    OpenFlow   194 Type: OFPT_FLOW_MOD
17 11.773734936  172.17.0.3    172.17.0.1    OpenFlow   176 Type: OFPT_PACKET_OUT
18 11.773779486  172.17.0.1    172.17.0.3    TCP         66 39280 → 6633 [ACK] Seq=
19 11.774269991  172.17.0.1    172.17.0.3    OpenFlow   206 Type: OFPT_PACKET_IN
20 11.774768106  172.17.0.3    172.17.0.1    OpenFlow   202 Type: OFPT_FLOW_MOD
21 11.774814526  172.17.0.3    172.17.0.1    OpenFlow   244 Type: OFPT_PACKET_OUT
22 11.774852636  172.17.0.1    172.17.0.3    TCP         66 39280 → 6633 [ACK] Seq=

Table ID: 0
Command: OFPFC_ADD (0)
Idle timeout: 0
Hard timeout: 0
Priority: 1
Buffer ID: OFP_NO_BUFFER (4294967295)
Out port: 0
Out group: 0
Flags: 0x0000
Pad: 0000
Match
   Type: OFPMT_OXM (1)
   Length: 28
 ▾ OXM field
      Class: OFPXMC_OPENFLOW_BASIC (0x8000)
      0000 000. = Field: OFPXMT_OFB_IN_PORT (0)
      .... ...0 = Has mask: False
      Length: 4
      Value: 1
 ▾ OXM field
      Class: OFPXMC_OPENFLOW_BASIC (0x8000)
      0000 011. = Field: OFPXMT_OFB_ETH_DST (3)
      .... ...0 = Has mask: False
      Length: 6
      Value: f6:e2:b7:39:99:7e (f6:e2:b7:39:99:7e)
 ▾ OXM field
      Class: OFPXMC_OPENFLOW_BASIC (0x8000)
      0000 110. = Field: OFPXMT_OFB_VLAN_VID (6)
      .... ...0 = Has mask: False
      Length: 2
      ...1 .... .... .... = OFPVID_PRESENT: True
      .... 0000 0110 0100 = Value: 100
   Pad: 00000000
Instruction
   Type: OFPIT_APPLY_ACTIONS (4)
   Length: 48
   Pad: 00000000
 ▾ Action
      Type: OFPAT_POP_VLAN (18)
      Length: 8
      Pad: 00000000
 ▾ Action
      Type: OFPAT_OUTPUT (0)
      Length: 16
      Port: 3
      Max length: 65509
      Pad: 000000000000
 ▾ Action
      Type: OFPAT_OUTPUT (0)
      Length: 16
      Port: 2
      Max length: 65509
      Pad: 000000000000
```

Figure 3.2: OFPT_MOD_FLOW packet captured by EdgeShark.

destination MAC address; the actions include adding a VLAN tag and forwarding the packet to the router as well as to Snort. For packets that traverse R1 and are destined for `PC1` or `DMZ2`, the matching is done based on the ingress port, destination MAC address, and the VLAN tag; in these cases, the VLAN tag is removed.

For verification and debugging purposes, the tool EdgeShark was used. This is an extension of Wireshark that enables capturing traffic generated by Docker containers. In this scenario, we can observe an OFPT_MOD_FLOW packet (Fig. 3.2) being sent from the controller to the switch, containing the list of match fields and actions to be installed.

Figure 3.3 also verifies that the port mirroring to Snort is functioning correctly.


Cattura da packetflix:// remote cluster and container host capture (0.10.7): packetflix
File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonia Wireless Strumenti Aiuto
Applica un filtro di visualizzazione ... <Ctrl-/>

```
No.  Time         Source             Destination        Protocol Length Info
   1 0.000000000  5a:2d:32:be:a4:e8  Broadcast          ARP       46 Who has 10.0.0.1? Tell 10.0.0.2
   2 0.001175375  a6:e8:c8:b9:6b:34  5a:2d:32:be:a4:e8  ARP       42 10.0.0.1 is at a6:e8:c8:b9:6b:34
   3 0.002234946  10.0.0.2           10.0.1.3           ICMP     102 Echo (ping) request  id=0x0002, seq=1/256, ttl=64 (no response found!)
   4 0.003278575  a6:e8:c8:b9:6b:34  Broadcast          ARP       42 Who has 10.0.1.3? Tell 10.0.1.1
   5 0.004503312  5e:f3:7e:4c:1d:cc  a6:e8:c8:b9:6b:34  ARP       46 10.0.1.3 is at 5e:f3:7e:4c:1d:cc
   6 0.006010840  10.0.0.2           10.0.1.3           ICMP      98 Echo (ping) request  id=0x0002, seq=1/256, ttl=63 (reply in 7)
   7 0.006155697  10.0.1.3           10.0.0.2           ICMP     102 Echo (ping) reply    id=0x0002, seq=1/256, ttl=64 (request in 6)
   8 0.006241591  10.0.1.3           10.0.0.2           ICMP      98 Echo (ping) reply    id=0x0002, seq=1/256, ttl=63
   9 1.000623877  10.0.0.2           10.0.1.3           ICMP     102 Echo (ping) request  id=0x0002, seq=2/512, ttl=64 (no response found!)
  10 1.000736251  10.0.0.2           10.0.1.3           ICMP      98 Echo (ping) request  id=0x0002, seq=2/512, ttl=63 (reply in 11)
  11 1.000756242  10.0.1.3           10.0.0.2           ICMP     102 Echo (ping) reply    id=0x0002, seq=2/512, ttl=64 (request in 10)
  12 1.000763252  10.0.1.3           10.0.0.2           ICMP      98 Echo (ping) reply    id=0x0002, seq=2/512, ttl=63
  13 2.001338511  10.0.0.2           10.0.1.3           ICMP     102 Echo (ping) request  id=0x0002, seq=3/768, ttl=64 (no response found!)
  14 2.001359672  10.0.0.2           10.0.1.3           ICMP      98 Echo (ping) request  id=0x0002, seq=3/768, ttl=63 (reply in 15)
  15 2.001376423  10.0.1.3           10.0.0.2           ICMP     102 Echo (ping) reply    id=0x0002, seq=3/768, ttl=64 (request in 14)
  16 2.001382393  10.0.1.3           10.0.0.2           ICMP      98 Echo (ping) reply    id=0x0002, seq=3/768, ttl=63
  17 3.029318637  10.0.0.2           10.0.1.3           ICMP     102 Echo (ping) request  id=0x0002, seq=4/1024, ttl=64 (no response found!)
  18 3.029335927  10.0.0.2           10.0.1.3           ICMP      98 Echo (ping) request  id=0x0002, seq=4/1024, ttl=63 (reply in 19)
  19 3.029348688  10.0.1.3           10.0.0.2           ICMP     102 Echo (ping) reply    id=0x0002, seq=4/1024, ttl=64 (request in 18)
  20 3.029353308  10.0.1.3           10.0.0.2           ICMP      98 Echo (ping) reply    id=0x0002, seq=4/1024, ttl=63
  21 4.053340383  10.0.0.2           10.0.1.3           ICMP     102 Echo (ping) request  id=0x0002, seq=5/1280, ttl=64 (no response found!)

▸ Frame 1: 46 bytes on wire (368 bits), 46 bytes captured (368 bits) on interface ids-eth0, id 0
▸ Ethernet II, Src: 5a:2d:32:be:a4:e8 (5a:2d:32:be:a4:e8), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 100
▸ Address Resolution Protocol (request)

0000  ff ff ff ff ff ff 5a 2d  32 be a4 e8 81 00 00 64   ······Z- 2·····d
0010  08 06 00 01 08 00 06 04  00 01 5a 2d 32 be a4 e8   ········ ··Z-2···
0020  0a 00 00 02 00 00 00 00  00 00 0a 00 00 01         ········ ······

Pronto per caricare o catturare        Pacchetti: 40 · visualizzati: 40 (100.0%)        Profilo: Default
```

Figure 3.3: Capture of ICMP packets exchanged between `PC1` and `DMZ2` that are mirrored to Snort.

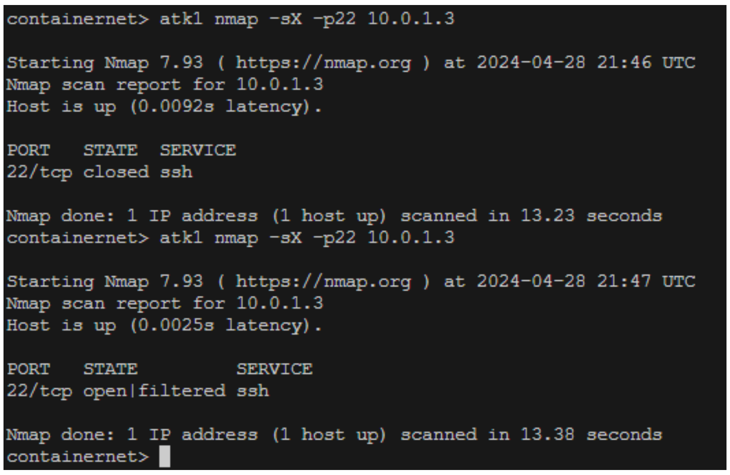
3.2 Port Scanning

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

The first malicious activity tested was a port scanning reconnaissance. The Snort rules check for anomalous TCP flags and generate alerts that are sent to the Ryu controller (see Fig. 3.4c). In Fig. 3.4a, the first scan performed using NMAP shows that the SSH port is closed (since `DMZ2` does not have an SSH server running). However, in the second scan, the port appears open because the traffic is now being redirected to a honeypot that has an SSH service available. Figure 3.4b displays the flow rules that have been installed, as described in Section 2.9.3.

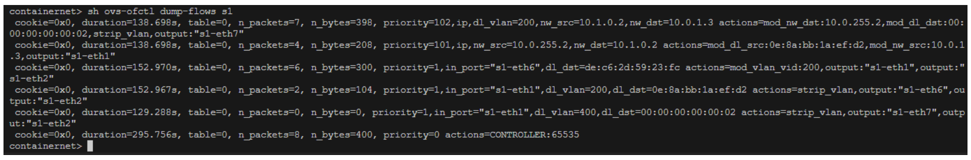
3.3 FTP Brute-force Attack

To create an effective Snort rule against a brute-force attack, a traffic capture was first obtained between a hypothetical attacker and an FTP server. As shown in Fig. 3.5, the server responds



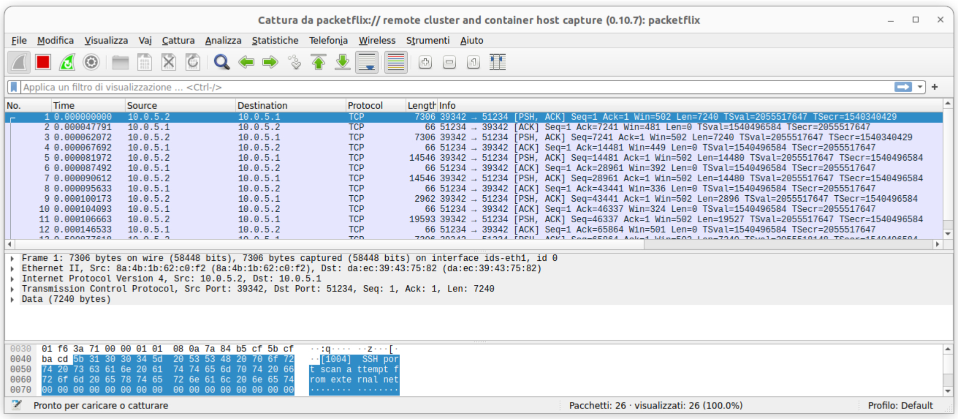
```
containernet> atk1 nmap -sX -p22 10.0.1.3

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) NMAP console output.



```
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>
```

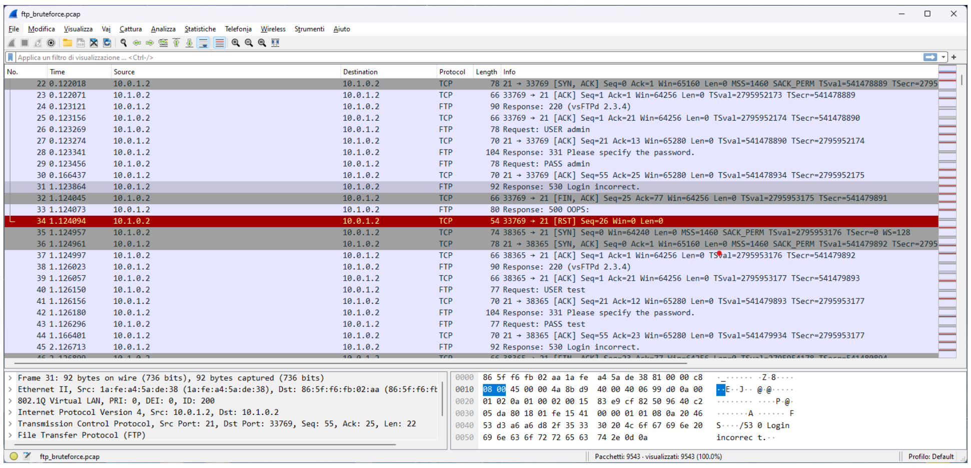
(b) Installed flow rules.


Cattura da packetflix:// remote cluster and container host capture (0.10.7): packetflix
File Modifica Visualizza Vai Cattura Analizza Statistiche Telefonia Wireless Strumenti Aiuto
Applica un filtro di visualizzazione ... <Ctrl-/>

```
No.  Time         Source     Destination  Protocol  Length Info
   1 0.000000000  10.0.5.2   10.0.5.1     TCP        7306 39342 → 51234 [PSH, ACK] Seq=1 Ack=1 Win=502 Len=7240 TSval=2055517647 TSecr=1540340429
   2 0.000047791  10.0.5.1   10.0.5.2     TCP          66 51234 → 39342 [ACK] Seq=1 Ack=7241 Win=481 Len=0 TSval=1540496584 TSecr=2055517647
   3 0.000062072  10.0.5.2   10.0.5.1     TCP        7306 39342 → 51234 [PSH, ACK] Seq=7241 Ack=1 Win=502 Len=7240 TSval=2055517647 TSecr=1540340429
   4 0.000067692  10.0.5.1   10.0.5.2     TCP          66 51234 → 39342 [ACK] Seq=1 Ack=14481 Win=449 Len=0 TSval=1540496584 TSecr=2055517647
   5 0.000081972  10.0.5.2   10.0.5.1     TCP       14546 39342 → 51234 [PSH, ACK] Seq=14481 Ack=1 Win=502 Len=14480 TSval=2055517647 TSecr=1540496584
   6 0.000087492  10.0.5.1   10.0.5.2     TCP          66 51234 → 39342 [ACK] Seq=1 Ack=28961 Win=392 Len=0 TSval=1540496584 TSecr=2055517647
   7 0.000090612  10.0.5.2   10.0.5.1     TCP       14546 39342 → 51234 [PSH, ACK] Seq=28961 Ack=1 Win=502 Len=14480 TSval=2055517647 TSecr=1540496584
   8 0.000095633  10.0.5.1   10.0.5.2     TCP          66 51234 → 39342 [ACK] Seq=1 Ack=43441 Win=336 Len=0 TSval=1540496584 TSecr=2055517647
   9 0.000100173  10.0.5.2   10.0.5.1     TCP        2962 39342 → 51234 [PSH, ACK] Seq=43441 Ack=1 Win=502 Len=2896 TSval=2055517647 TSecr=1540496584
  10 0.000104093  10.0.5.1   10.0.5.2     TCP          66 51234 → 39342 [ACK] Seq=1 Ack=46337 Win=324 Len=0 TSval=1540496584 TSecr=2055517647
  11 0.000106663  10.0.5.2   10.0.5.1     TCP       19593 39342 → 51234 [PSH, ACK] Seq=46337 Ack=1 Win=502 Len=19527 TSval=2055517647 TSecr=1540496584
  12 0.000146533  10.0.5.1   10.0.5.2     TCP          66 51234 → 39342 [ACK] Seq=1 Ack=65864 Win=501 Len=0 TSval=1540496584 TSecr=2055517647

▸ Frame 1: 7306 bytes on wire (58448 bits), 7306 bytes captured (58448 bits) on interface ids-eth1, id 0
▸ Ethernet II, Src: 8a:4b:1b:62:c0:f2 (8a:4b:1b:62:c0:f2), Dst: da:ec:39:43:75:82 (da:ec:39:43:75:82)
▸ Internet Protocol Version 4, Src: 10.0.5.2, Dst: 10.0.5.1
▸ Transmission Control Protocol, Src Port: 39342, Dst Port: 51234, Seq: 1, Ack: 1, Len: 7240
▸ Data (7240 bytes)

0030  01 f6 3a 71 00 00 01 01  08 0a 7a 84 b5 cf 5b cf   ··:q···· ··z···[·
0040  ba cd 5b 31 30 30 34 5d  20 53 53 48 20 70 6f 72   ··[1004]  SSH por
0050  74 20 73 63 61 6e 20 61  74 74 65 6d 70 74 20 66   t scan a ttempt f
0060  72 6f 6d 20 65 78 74 65  72 6e 61 6c 20 6e 65 74   rom exte rnal net
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········

Pronto per caricare o catturare        Pacchetti: 26 · visualizzati: 26 (100.0%)        Profilo: Default
```

(c) Messages sent from Snort to the Ryu controller.

Figure 3.4: XMAS scan attempt on port 21 by ATK1.



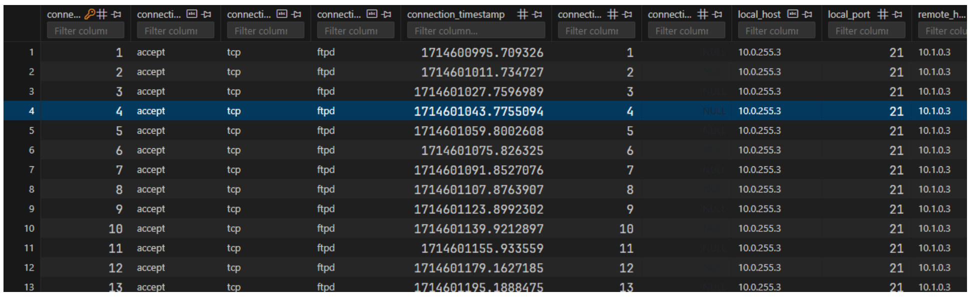
```
No.  Time      Source    Destination  Protocol Length Info
  22 0.122018  10.0.1.2  10.1.0.2     TCP       78 21 → 33769 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=541478889 TSecr=2795
  23 0.122071  10.1.0.2  10.0.1.2     TCP       66 33769 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2795952173 TSecr=541478889
  24 0.123121  10.0.1.2  10.1.0.2     FTP       90 Response: 220 (vsFTPd 2.3.4)
  25 0.123156  10.1.0.2  10.0.1.2     TCP       66 33769 → 21 [ACK] Seq=1 Ack=21 Win=64256 Len=0 TSval=2795952174 TSecr=541478890
  26 0.123269  10.1.0.2  10.0.1.2     FTP       78 Request: USER admin
  27 0.123274  10.0.1.2  10.1.0.2     TCP       70 21 → 33769 [ACK] Seq=21 Ack=13 Win=65280 Len=0 TSval=541478890 TSecr=2795952174
  28 0.123341  10.0.1.2  10.1.0.2     FTP      104 Response: 331 Please specify the password.
  29 0.123456  10.1.0.2  10.0.1.2     FTP       78 Request: PASS admin
  30 0.166437  10.0.1.2  10.1.0.2     TCP       70 21 → 33769 [ACK] Seq=55 Ack=25 Win=65280 Len=0 TSval=541478934 TSecr=2795952175
  31 1.123864  10.0.1.2  10.1.0.2     FTP       92 Response: 530 Login incorrect.
  32 1.124045  10.1.0.2  10.0.1.2     TCP       66 33769 → 21 [FIN, ACK] Seq=25 Ack=77 Win=64256 Len=0 TSval=2795953175 TSecr=541479891
  33 1.124073  10.0.1.2  10.1.0.2     FTP       80 Response: 500 OOPS:
  34 1.124094  10.1.0.2  10.0.1.2     TCP       54 33769 → 21 [RST] Seq=26 Win=0 Len=0
  35 1.124957  10.1.0.2  10.0.1.2     TCP       74 38365 → 21 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=2795953176 TSecr=0 WS=128
  36 1.124961  10.0.1.2  10.1.0.2     TCP       78 21 → 38365 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=541479892 TSecr=2795
  37 1.124997  10.1.0.2  10.0.1.2     TCP       66 38365 → 21 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2795953176 TSecr=541479892
  38 1.126023  10.0.1.2  10.1.0.2     FTP       90 Response: 220 (vsFTPd 2.3.4)
  39 1.126057  10.1.0.2  10.0.1.2     TCP       66 38365 → 21 [ACK] Seq=1 Ack=21 Win=64256 Len=0 TSval=2795953177 TSecr=541479893
  40 1.126150  10.1.0.2  10.0.1.2     FTP       77 Request: USER test
  41 1.126156  10.0.1.2  10.1.0.2     TCP       70 21 → 38365 [ACK] Seq=21 Ack=12 Win=65280 Len=0 TSval=541479893 TSecr=2795953177
  42 1.126180  10.0.1.2  10.1.0.2     FTP      104 Response: 331 Please specify the password.
  43 1.126296  10.1.0.2  10.0.1.2     FTP       77 Request: PASS test
  44 1.166401  10.0.1.2  10.1.0.2     TCP       70 21 → 38365 [ACK] Seq=55 Ack=23 Win=65280 Len=0 TSval=541479934 TSecr=2795953177
  45 2.126713  10.0.1.2  10.1.0.2     FTP       92 Response: 530 Login incorrect.

> Frame 31: 92 bytes on wire (736 bits), 92 bytes captured (736 bits)
> Ethernet II, Src: 1a:fe:a4:5a:de:38 (1a:fe:a4:5a:de:38), Dst: 86:5f:f6:fb:02:aa (86:5f:f6:fb
> 802.1Q Virtual LAN, PRI: 0, DEI: 0, ID: 200
> Internet Protocol Version 4, Src: 10.0.1.2, Dst: 10.1.0.2
> Transmission Control Protocol, Src Port: 21, Dst Port: 33769, Seq: 55, Ack: 25, Len: 22
> File Transfer Protocol (FTP)

0000  86 5f f6 fb 02 aa 1a fe  a4 5a de 38 81 00 00 c8   ·_······ ·Z·8····
0010  08 00 45 00 00 4a 8b d9  40 00 40 06 99 d0 0a 00   ··E··J·· @·@·····
0020  01 02 0a 01 00 02 00 15  83 e9 cf 82 50 96 40 c2   ········ ····P·@·
0030  05 da 80 18 01 fe 15 41  00 00 01 01 08 0a 20 46   ·······A ······ F
0040  53 d3 a6 a6 d8 2f 35 33  30 20 4c 6f 67 69 6e 20   S····/53 0 Login
0050  69 6e 63 6f 72 72 65 63  74 2e 0d 0a               incorrec t.··

Pacchetti: 9543 · visualizzati: 9543 (100.0%)    Profilo: Default
```

Figure 3.5: Capture of an FTP brute-force attack.

with error codes such as 530 (and occasionally 500) after several failed login attempts. Therefore, a rule was created to trigger if an excessive number of access attempts is detected within a specified time interval. In particular, the rule monitors for FTP error codes 500 and 530, and if more than five occurrences are detected within 10 seconds, the moving target defense mechanism is activated.

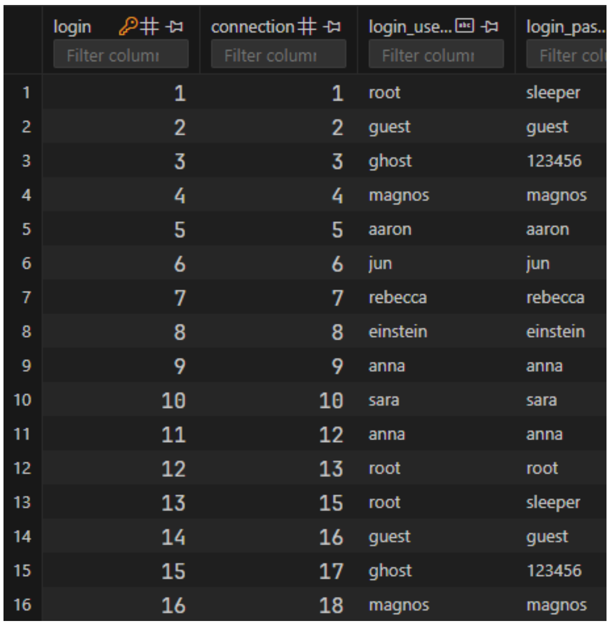
```

The attackers use the Metasploit module \verb|auxiliary/scanner/ftp/ftp_login| and supply a text file containing a list of username-password pairs for the attack.\

The two honeypots react differently to FTP connection attempts: \begin{itemize} \item \textbf{Heralding} logs the session (in either \texttt{.csv} or \texttt{.json} format) along with the login credentials used during the attack, but it does not allow the attacker to perform any operations. \end{itemize}

\begin{lstlisting}
{
    "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": {}
}
```

- **Dionaea** not only permits a real login but also allows the attacker to operate (albeit in a limited fashion) within a designated subdirectory of the honeypot's simulated filesystem. The activity data is recorded in an SQLite database (see 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	NULL	10.0.255.3	21	10.1.0.3
2	2	accept	tcp	ftpd	1714601011.734727	2	NULL	10.0.255.3	21	10.1.0.3
3	3	accept	tcp	ftpd	1714601027.7596989	3	NULL	10.0.255.3	21	10.1.0.3
4	4	accept	tcp	ftpd	1714601043.7755094	4	NULL	10.0.255.3	21	10.1.0.3
5	5	accept	tcp	ftpd	1714601059.8002608	5	NULL	10.0.255.3	21	10.1.0.3
6	6	accept	tcp	ftpd	1714601075.826325	6	NULL	10.0.255.3	21	10.1.0.3
7	7	accept	tcp	ftpd	1714601091.8527076	7	NULL	10.0.255.3	21	10.1.0.3
8	8	accept	tcp	ftpd	1714601107.8763907	8	NULL	10.0.255.3	21	10.1.0.3
9	9	accept	tcp	ftpd	1714601123.8992302	9	NULL	10.0.255.3	21	10.1.0.3
10	10	accept	tcp	ftpd	1714601139.9212897	10	NULL	10.0.255.3	21	10.1.0.3
11	11	accept	tcp	ftpd	1714601155.933559	11	NULL	10.0.255.3	21	10.1.0.3
12	12	accept	tcp	ftpd	1714601179.1627185	12	NULL	10.0.255.3	21	10.1.0.3
13	13	accept	tcp	ftpd	1714601195.1888475	13	NULL	10.0.255.3	21	10.1.0.3

(a) Dionaea Sessions.



	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) Dionaea Login Records.

Figure 3.6: FTP brute-force attack logged by Dionaea.

Chapter 4

Conclusions

This study explored the integration and configuration of an SDN network utilizing the Ryu controller, with Snort deployed as an IDS/IPS, and the use of honeypots—Heralding and Dionaea—to enhance network security through Moving Target Defense (MTD) techniques. This setup successfully demonstrated how SDN's flexibility in managing data flows can dynamically respond to cyberattacks by redirecting suspicious traffic toward honeypot systems designed for in-depth analysis and threat intelligence gathering.

4.1 Key Results and Approach Validity

The experimental setup validated the hypothesis that SDNs, thanks to their dynamic flow management capabilities, enable a proactive defense against attacks by reducing the exposed attack surface and confounding adversaries. The ability to modify network routes in real time proved crucial in diverting malicious traffic toward honeypots, thereby protecting critical systems. Additionally, Snort's deployment facilitated a detailed analysis of network traffic, enabling the timely detection of suspicious behaviors.

4.2 Limitations of the Implemented System

Despite the promising results, the implementation exhibited some limitations. The complexity of configuring and managing security rules requires specialized, in-depth knowledge, which can pose challenges in practical deployments. Moreover, the simulation was conducted on a relatively small and controlled scale, which may not fully capture the complexities of larger, real-world networks.

4.3 Future Implications and Areas for Improvement

Looking ahead, it is worthwhile to explore the integration of artificial intelligence and machine learning solutions to further automate response mechanisms and enhance the prediction and prevention of emerging threats. Another area for improvement is the optimization of network performance to ensure that the security measures do not adversely affect operational efficiency.

In conclusion, this project confirms the potential of SDN and MTD techniques in strengthening network security. With ongoing advancements and refinements, this approach promises to deliver a highly effective and adaptable cybersecurity strategy capable of addressing the evolving threats in today's digital landscape.