Dominique Brown

CSCI 6397

Network Security

Md Sharif Ullah

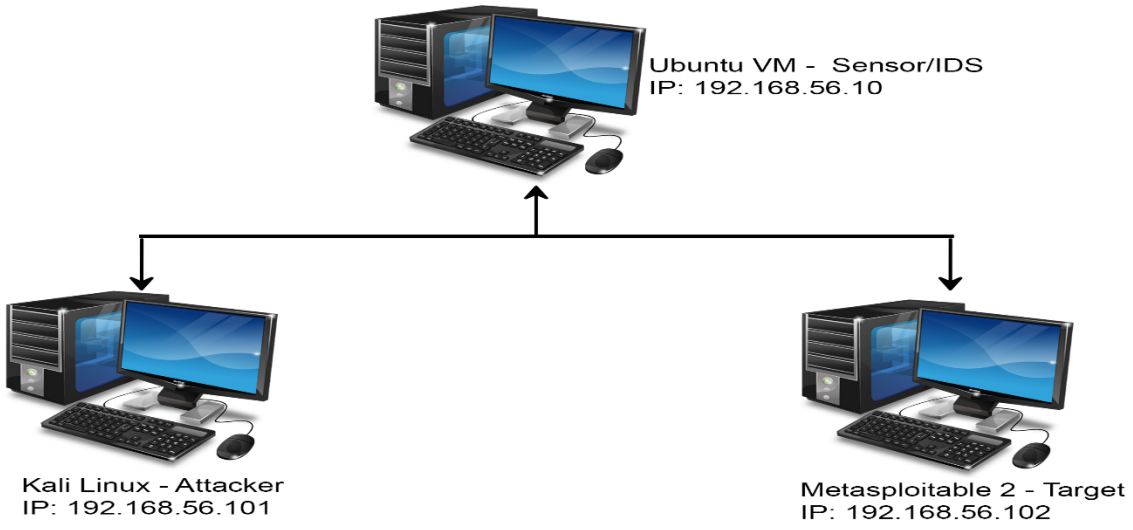
Topic: "Analysis of Network footprints from a VM"

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

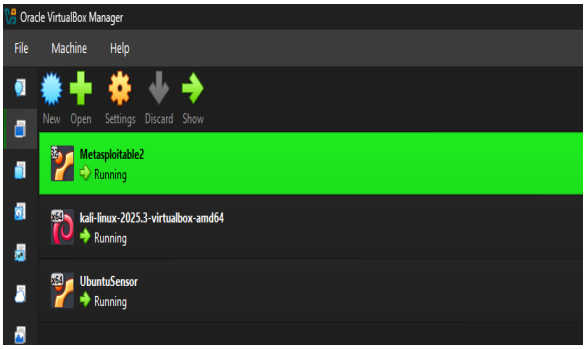
Virtual Environment Architecture:



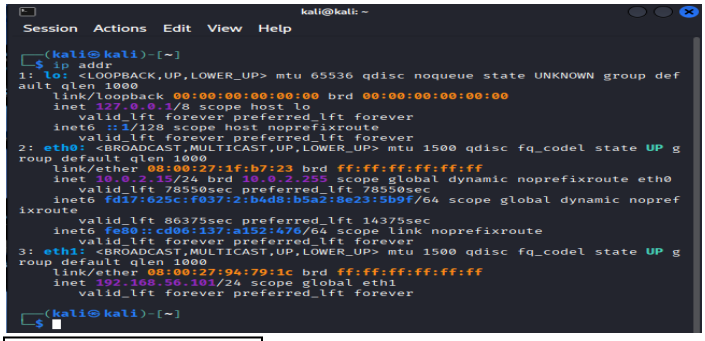
Environment Setup:

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.

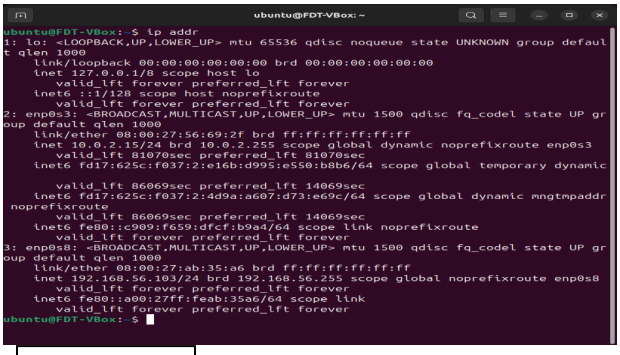
Screenshots



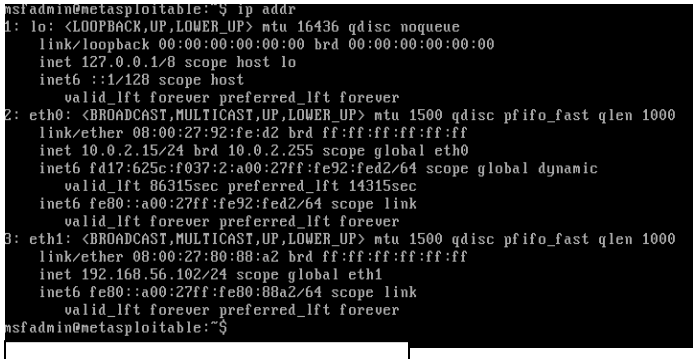
List of Virtual Machines



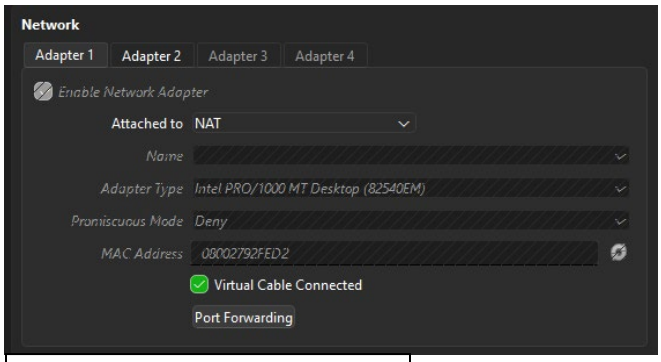
Kali IP Address



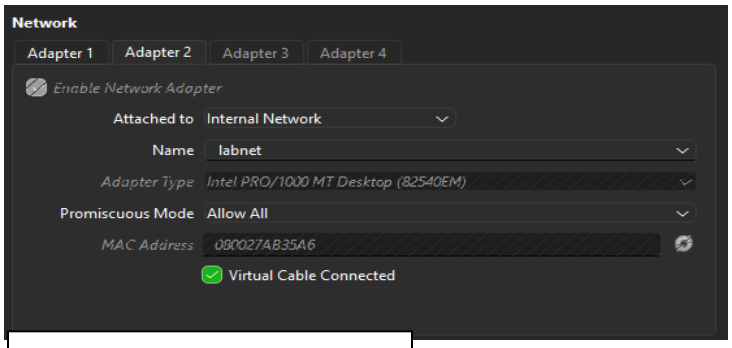
Ubuntu IP



Metaspoitable IP Address



Adapter 1 attached to NAT



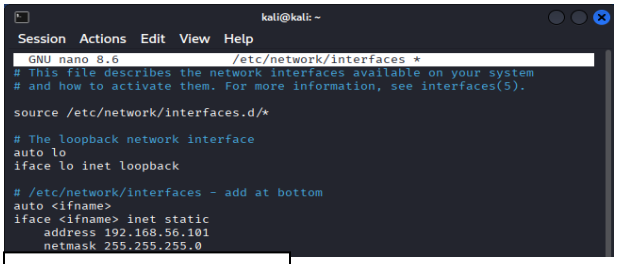
Internal Network “labnet”

Network Configuration Challenges:

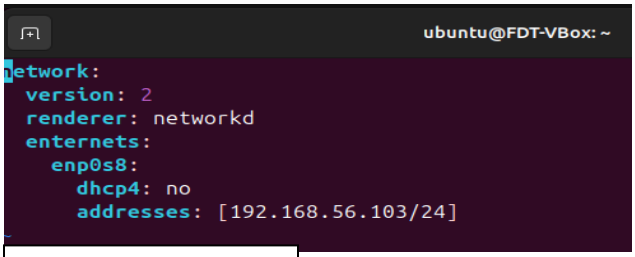
Another challenge emerged when I attempted to use ifup, ifdown, and other legacy networking commands on Metasploitable-2. The system responded with errors such as “unknown interface eth1” and “couldn’t read /etc/network/interfaces.” These errors were due to the fact that newer distributions—and sometimes 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.

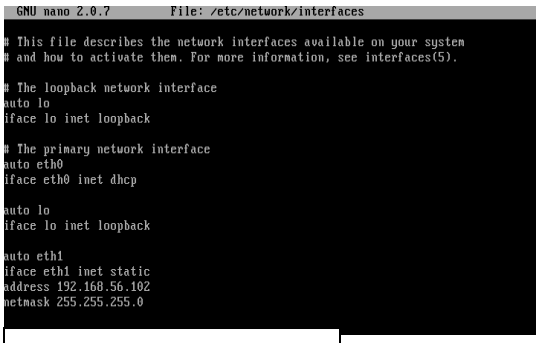
Screenshots



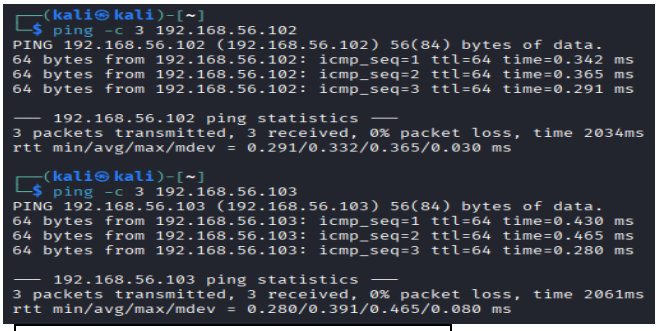
Kali Linux netplan



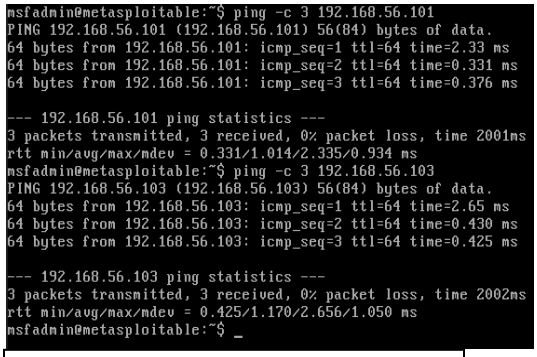
Ubuntu netplan



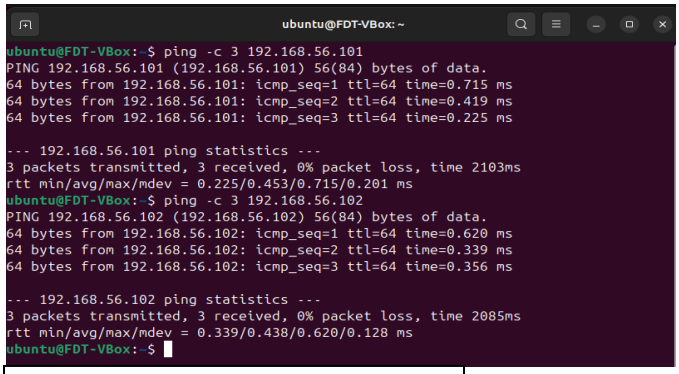
Metasploitable netplan



Testing connection to each VM from Kali using ping



Testing connection to each VM from metasploitable using ping

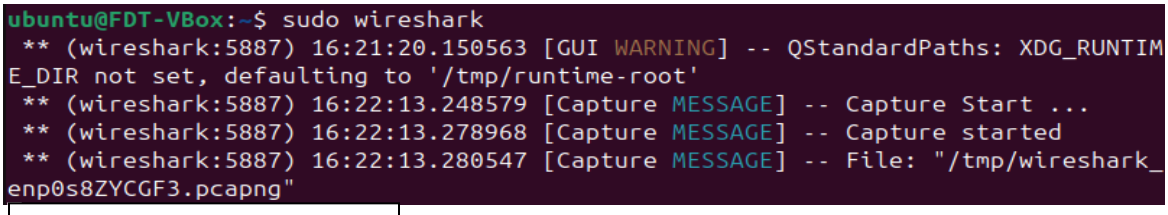


Testing connection to each VM from ubuntu using ping

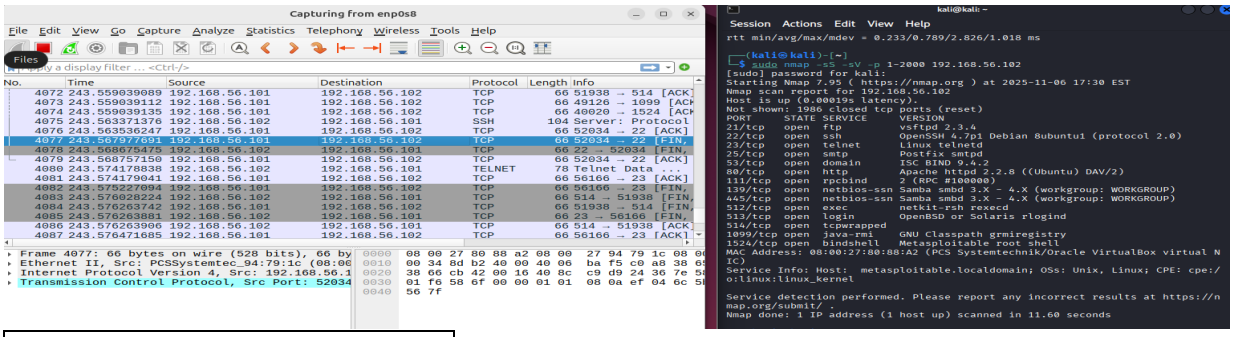
Baseline Traffic Capture:

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.

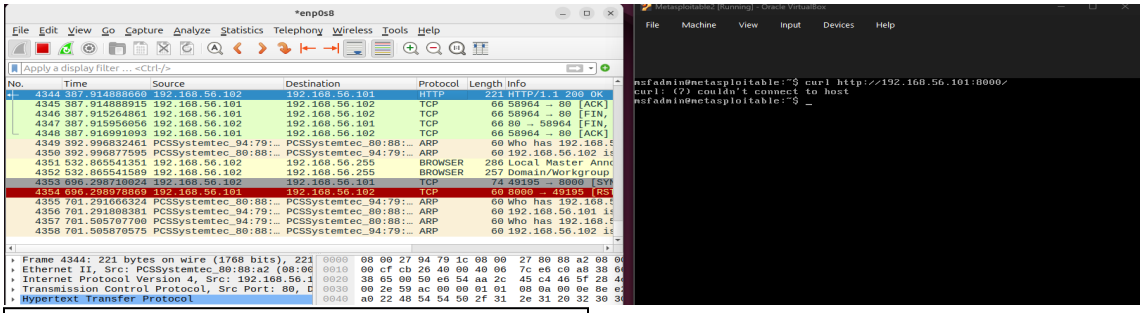
Screenshots



Starting wireshark capture



packet traffic capture from Kali

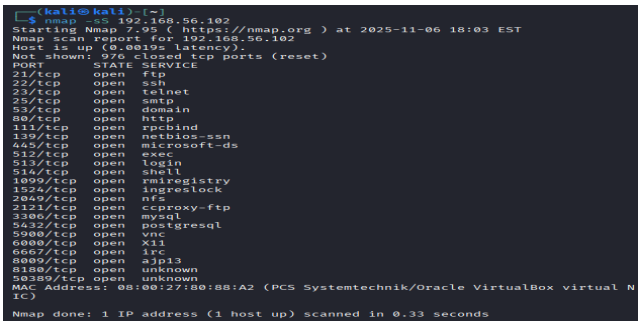


packet traffic capture from Metasploitable

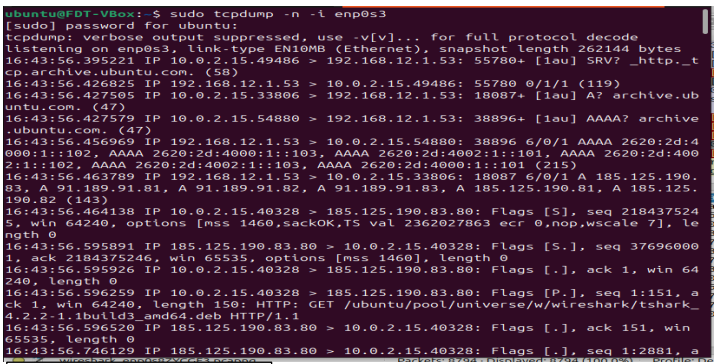
Reconnaissance Phase:

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:



Performing Reconnaissance



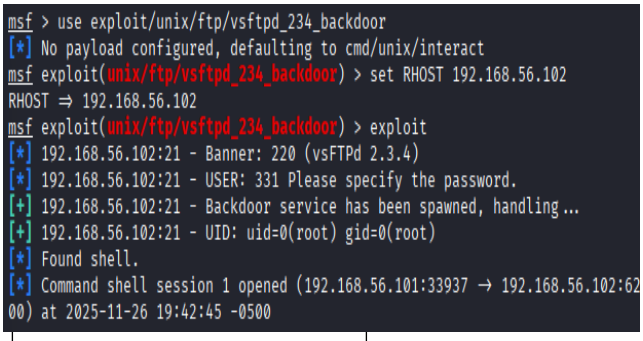
Scanning Target

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.

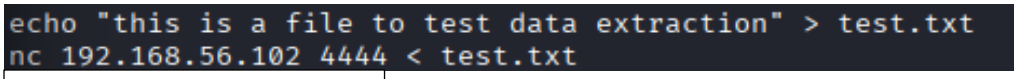
Screenshots

Performing backdoor exploit

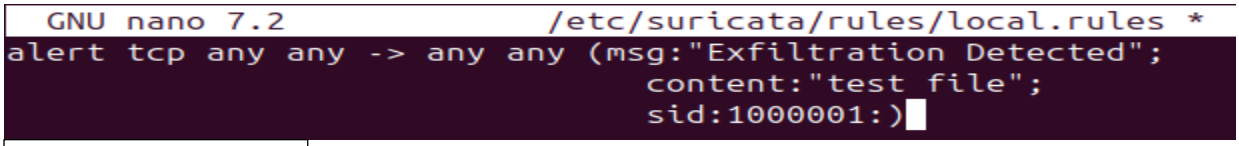


Starting Metasploitable

set listener on target



Simulating data extraction



Setting Suricata rule

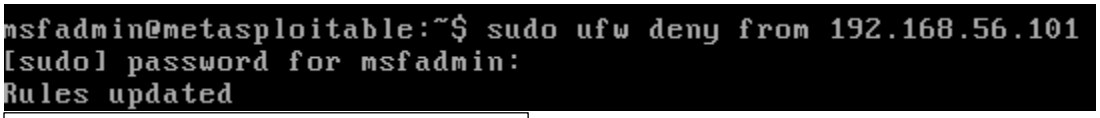
Runnung suricata on attack pcap

check suricata alerts

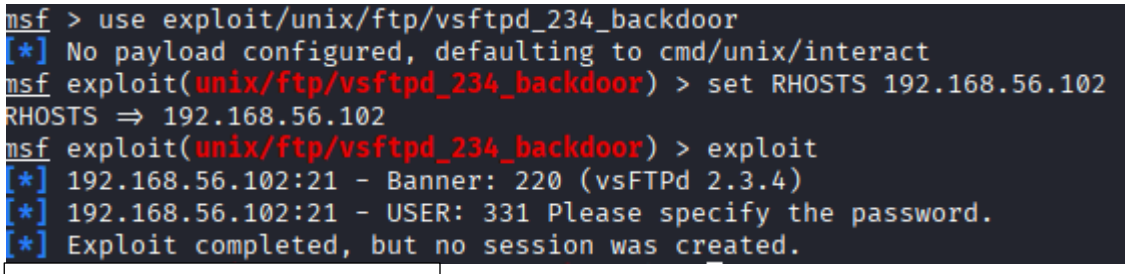
Mitigation

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



Apply Mitigation on Metasploitable



Attacker Exploitation Blocked

Automation and Scalability for Real-World use:

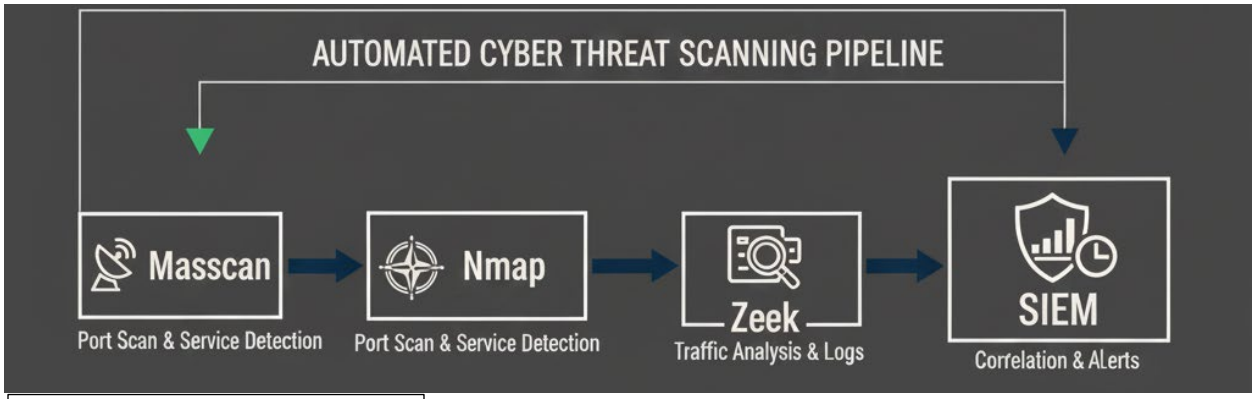
As I reflected on the penetration-testing workflow I used in this project, I realized that nearly every step—from reconnaissance to mitigation—can be scaled and automated for real-

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



Automated scanning pipeline

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       resp_ip_bytes   tunnel_parents
#types time     string  addr    port    addr    port    enum    string  interval        count   count   string  bool    bool    count   string  count   count   count   count   set[string]

1733330111.234156       CXbAqK2HOL0eTh9oB       192.168.56.101  51392   192.168.56.105  443     tcp     https   4.22    1420    3280    SF      -       -       0       ShADadf 8       512
        10      660     -
1733330114.983210       Cua9Zp1eYnDQ7Hx 192.168.56.101  46433   192.168.56.102  21      tcp     ftp     0.09    120     64      S1      -       -       0       S       2       120     1
        64      -
1733330116.552891       CnBv2kQx7aP0sE  192.168.56.105  3389    10.0.2.15       50110   tcp     rdp     10.28   4990    12000   ESTAB   -       -       0       ShADadFf        45      3600
        60      5400    -
1733330119.882578       CJt07qxHhmDueg  192.168.56.102  49221   192.168.56.105  80      tcp     http    2.15    820     2350    Fs      -       -       0       ShADf   7       380     9
        560     -
1733330122.332745       CopkQ1bFnWQs5F  192.168.56.105  54418   8.8.8.8 53      udp     dns     0.01    58      120     SF      -       -       0       D       1       58      1       120
        -
1733330125.111943       CH2qvYeG31nBFk  192.168.56.102  37341   192.168.56.101  22      tcp     ssh     25.89   1920    2560    SF      -       -       0       ShAdDf  32      2080    42
        3020    -
1733330127.994201       CvNxLQ2bDjcLzp  192.168.56.101  40011   192.168.56.105  3306    tcp     mysql   1.45    740     1330    SF      -       -       0       ShAd    14      980     20
        1580    -
1733330131.004992       CQFn7RczGHS28a  192.168.56.105  51200   192.168.56.102  139     tcp     netbios 0.66    200     180     REJ     -       -       0       R       3       210     3
        190     -
1733330133.287001       CZx91FedpJkaPx  192.168.56.101  38777   192.168.56.105  111     tcp     rpc     0.32    160     305     SF      -       -       0       ShAD    5       250     7
        340     -
1733330136.441788       CxLLm4Pdq0w9Sh  192.168.56.105  60222   192.168.56.101  445     tcp     smb     4.45    2900    4800    SF      -       -       0       ShADadF 26      1900    33
        2700    -
```

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.