

**Saeed Rafee**  
**Security Analysis and Tool Deployment with Kali Linux**  
**05/19/2025**

---

**Objective:**

This lab demonstrates the setup and use of Kali Linux in a controlled virtual environment to conduct a personal security assessment. The goal was to explore key cybersecurity tools and techniques such as Nmap scanning, packet capture with Wireshark, vulnerability scanning with Legion, and threat simulation using Snort, Aircrack-ng, and Metasploit. Each step reflects practical thinking from the perspective of an analyst and showcases how these tools can be used to evaluate and strengthen system and network defenses.

To begin this project, I set up Kali Linux as a virtual machine using VirtualBox on my local system. This allowed me to safely run penetration testing tools without affecting my host machine. The virtual environment provided an isolated, controlled space where I could simulate network interactions and conduct security assessments. Having Kali running in a VM also gave me the flexibility to take snapshots, roll back changes, and work with various tools like Nmap, Wireshark, and Metasploit throughout this lab. This setup laid the foundation for all the hands-on testing that followed.



Once Kali was up and running, I used Nmap to run a quick scan on my LAMP stack (10.0.2.15) to start building situational awareness around the network. From the results, I could tell the host was responsive based on the low latency and successful return. Ports 22 (SSH) and 80 (HTTP) were open, which tells me that remote shell access and a web server are both available.

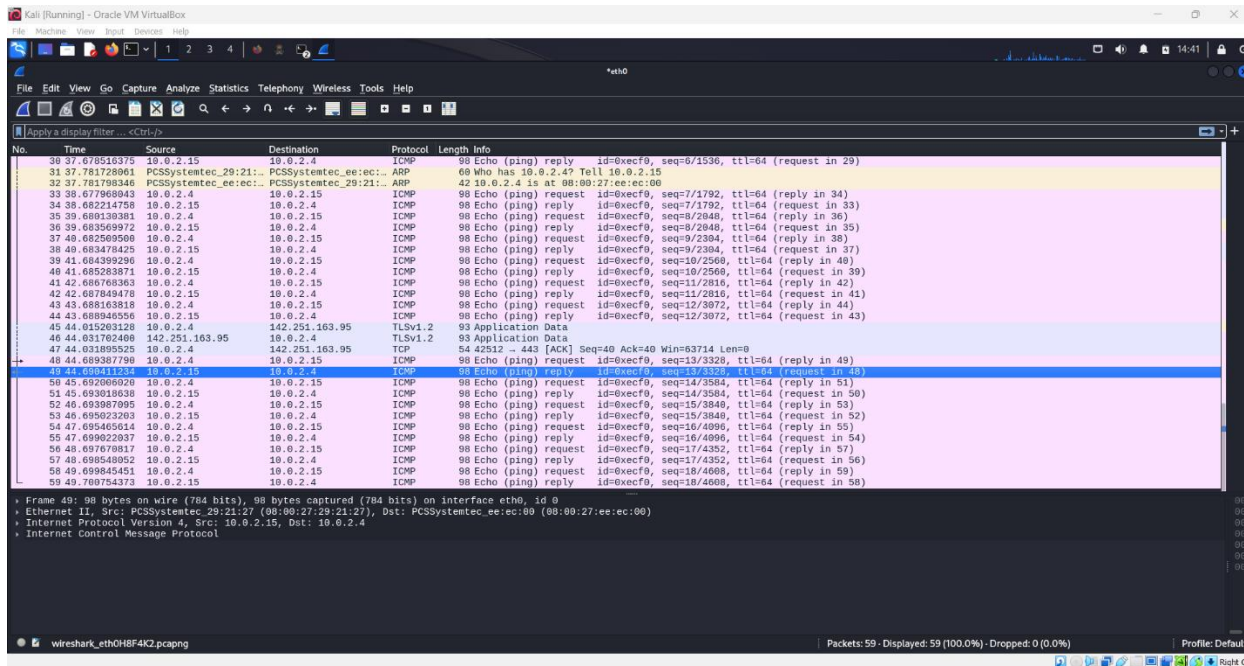
A screenshot of a Kali Linux terminal window. The window title is "Kali [Running] - Oracle VM VirtualBox". The terminal shows the command `nmap 10.0.2.15` being executed. The output is as follows:

```
saeed@kali: ~/Desktop
[saeed@kali]~$ nmap 10.0.2.15
Starting Nmap ( https://nmap.org ) at 2024-03-21 13:44 EDT
Nmap scan report for 10.0.2.15
Host is up (0.000075 latency).
Not shown: 998 closed tcp ports (conn-refused)
PORT      STATE SERVICE
22/tcp    open  ssh
80/tcp    open  http
Nmap done: 1 IP address (1 host up) scanned in 0.30 seconds
```

The terminal background features the Kali Linux logo and the text "KALI LINUX" and "the quieter you become, the more you are able to hear".

While the open ports themselves aren't inherently dangerous, they could represent possible attack surfaces depending on how those services are configured. At this point, I'm noting that the machine is online, reachable, and offering services; but I'd need to dig deeper (e.g., banner grabbing, version enumeration) to evaluate any actual vulnerabilities.

After confirming that my Kali VM was running properly and could reach the target, I opened Wireshark to capture live packet traffic while pinging my LAMP stack. I immediately saw a sequence of ICMP echo requests and replies, which showed successful communication and confirmed that the server was reachable.

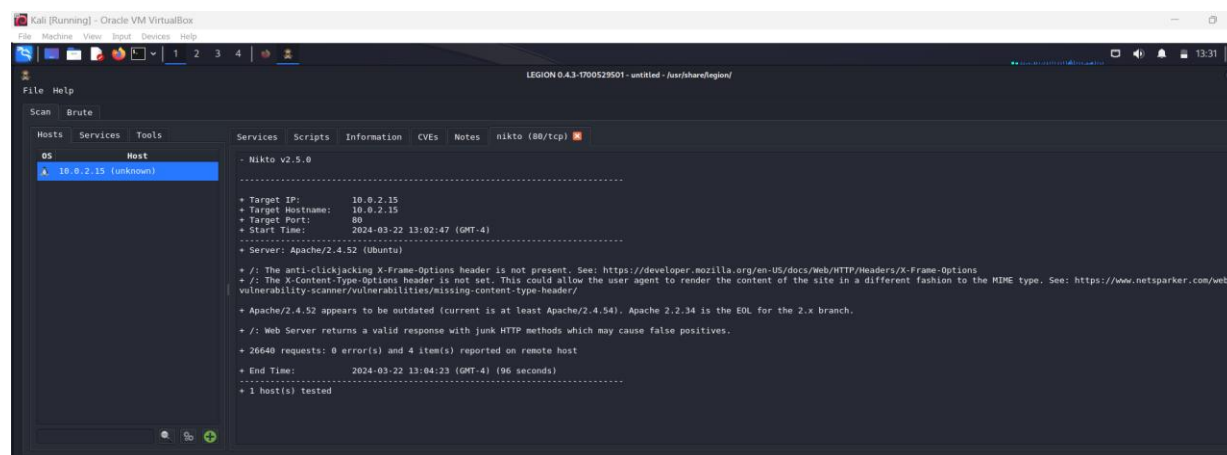
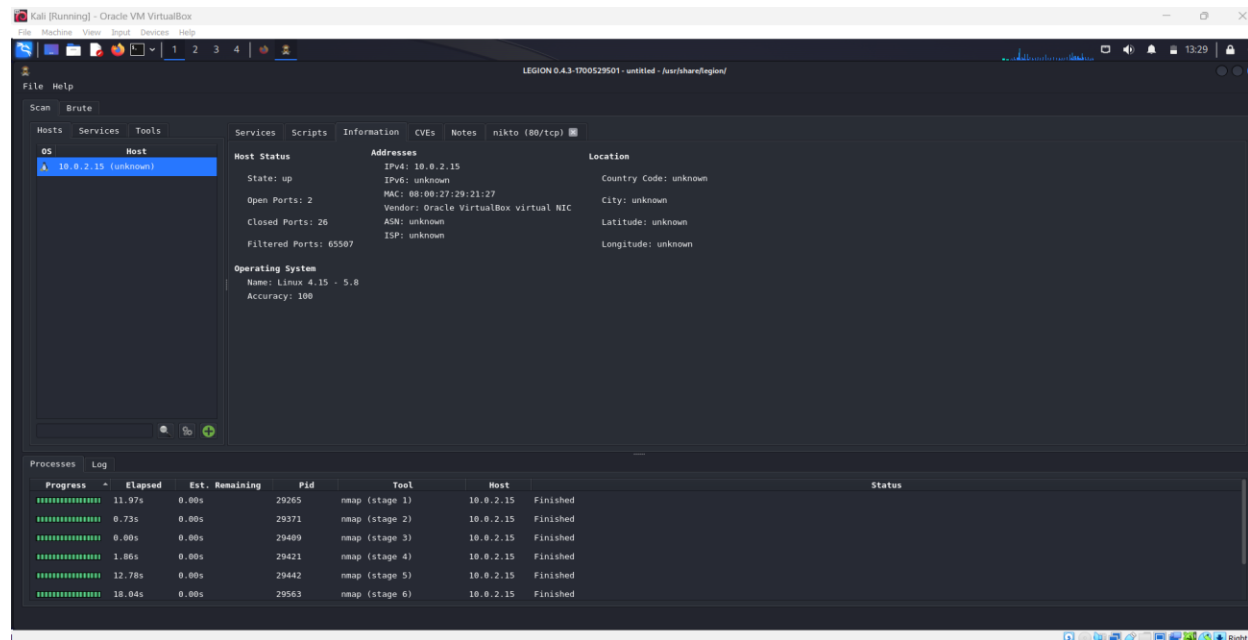


Alongside that, ARP packets were present, helping map IP addresses to MAC addresses on the local network. This is expected whenever two devices try to communicate directly.

I also picked up TCP packets in the capture. Given the ports involved and the timing of the traffic, it appeared to be related to HTTP requests going to the LAMP stack's web server.

Everything I observed reflected typical traffic behavior for a machine interacting with a web server. This gave me confidence that the stack was running correctly and responding to network communication as expected

After verifying connectivity and interaction with the LAMP stack, I used Legion to perform a more automated reconnaissance of the target system. Legion ran a series of modules including Nikto, which scanned the web server hosting the Yoga App. One of the key findings was that the server was running an outdated version of Apache, along with missing security headers.



This raised red flags from a hardening perspective. Outdated server software can introduce known vulnerabilities, and missing headers can expose the app to risks like clickjacking or cross-site scripting. These observations helped me build a list of next steps, such as updating Apache and applying standard security headers like X-Frame-Options and Content-Security-Policy.

Tools like Legion are great for quickly surfacing surface-level exposures. From here, I'd want to validate these findings manually and dig deeper into any modules or plugins running on the server. Nikto assesses the Yoga App web server it. It identifies potential security weaknesses like outdated server software (Apache) and missing security headers that could leave the app

vulnerable to attacks. By analyzing this output, we can improve the application's security by updating the server software and implementing recommended security headers.

### Cyber Challenge: Securing the Team in a High-Risk Environment

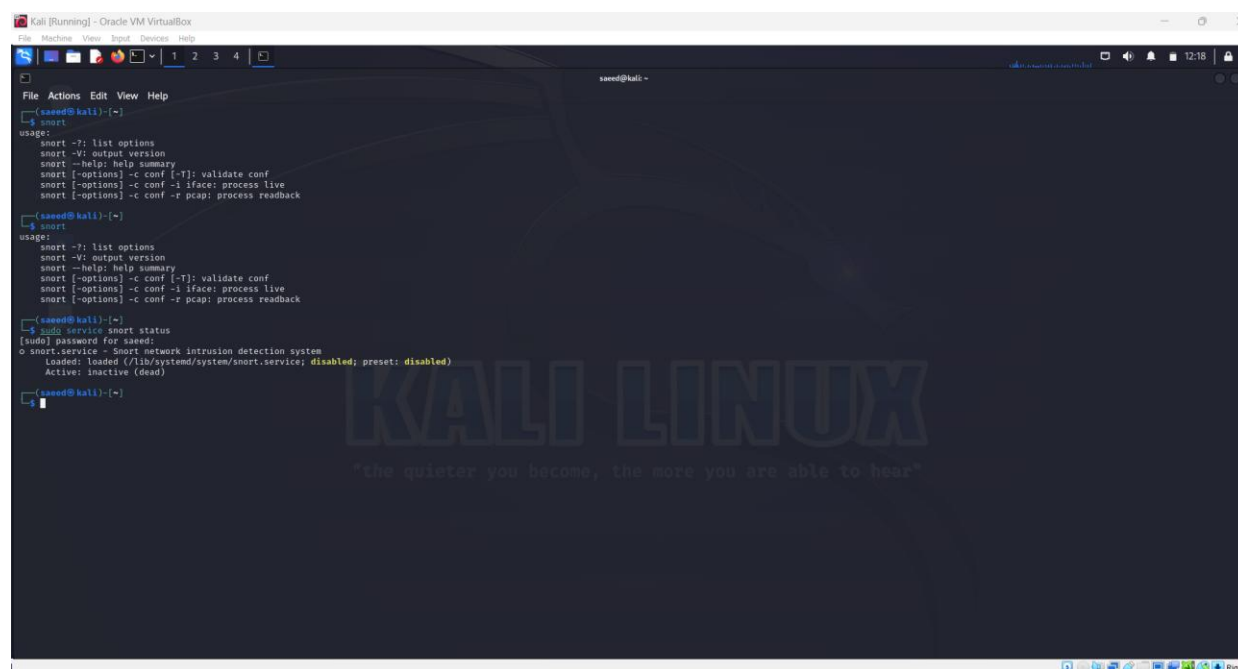
In this scenario, I'm assigned as the cybersecurity analyst responsible for protecting a remote operations team working in a high-risk region. The team is relying on laptops and mobile devices while connected to public Wi-Fi networks, which introduces risks such as interception, device compromise, and unauthorized access.

To help protect the team and ensure communication security, I would deploy the following three tools from my Kali Linux environment. These tools were selected to provide visibility, detection, and proactive defense across our network and endpoints.

#### Tool 1: Snort

##### Capabilities:

Snort is an open-source network intrusion detection system that monitors traffic in real time to detect known threats and suspicious patterns. It can identify activities like Trojan infections, port scans, and DoS attacks using protocol analysis and anomaly detection.



```

Kali [Running] - Oracle VM VirtualBox
File Actions Edit View Help
(saeed@kali)~$ snort
usage:
  snort -f: list options
  snort -V: output version
  snort --help: help summary
  snort [-options] -c conf [-T]: validate conf
  snort [-options] -c conf -i iface: process live
  snort [-options] -c conf -r pcap: process readback

(saeed@kali)~$ snort
usage:
  snort -f: list options
  snort -V: output version
  snort --help: help summary
  snort [-options] -c conf [-T]: validate conf
  snort [-options] -c conf -i iface: process live
  snort [-options] -c conf -r pcap: process readback

(saeed@kali)~$ sudo service snort status
[sudo] password for saeed:
o snort.service - Snort network intrusion detection system
   Loaded: loaded (/lib/systemd/system/snort.service; disabled; preset: disabled)
   Active: inactive (dead)

(saeed@kali)~$
  
```

##### How I used it:

I would deploy Snort on each team laptop to monitor network traffic while connected to public Wi-Fi. Snort would serve as a passive detector, alerting us to suspicious activity like scanning attempts or malicious payloads targeting known vulnerabilities. With this setup, I'd be able to respond quickly if a threat were detected during mission operations.

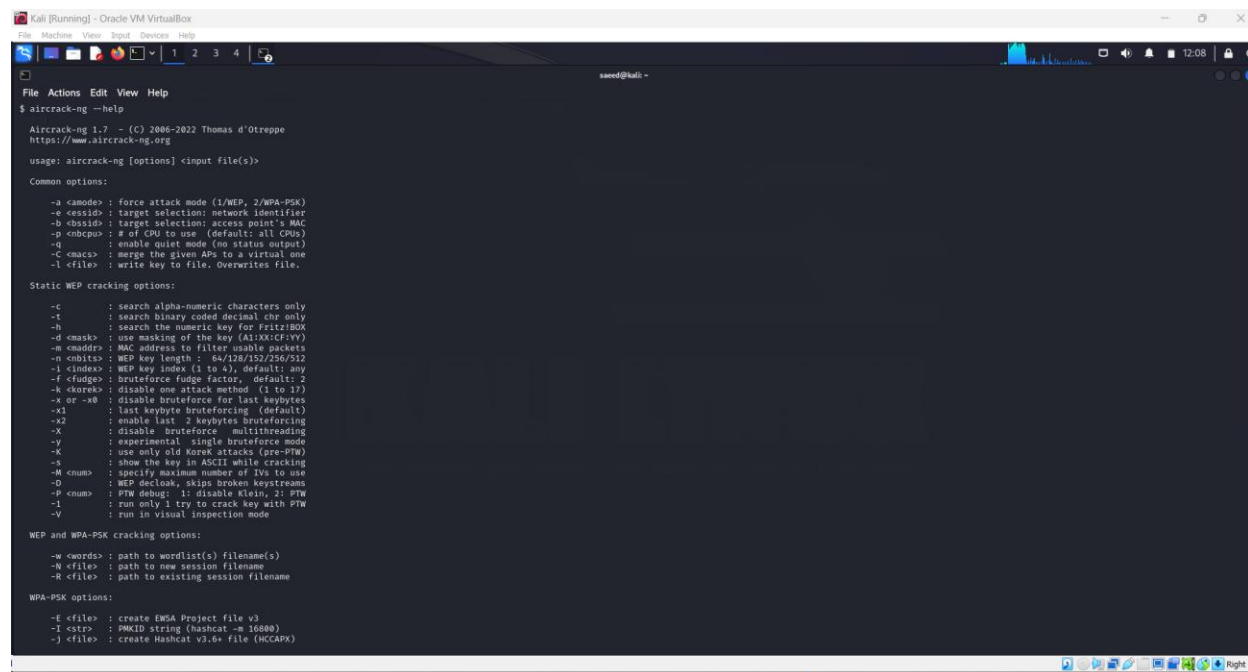
##### Value:

Snort would act as an early warning system, helping the team avoid compromise before an attack escalates. Its logging and alerting would also support post-incident analysis if any network activity looked abnormal.

## Tool 2: Aircrack-ng

### Capabilities:

Aircrack-ng is a suite used to test Wi-Fi network security. It includes tools for packet capture, monitoring, password auditing, and identifying weak encryption configurations.



```

Kali [Running] - Oracle VM VirtualBox
File Machine View Host Devices Help
saaved@kali: ~
$ aircrack-ng --help

Aircrack-ng 1.7 - (C) 2005-2022 Thomas d'Ottreppe
https://www.aircrack-ng.org

usage: aircrack-ng [options] <input file(s)>

Common options:
  -a <mode> : force attack mode (1/WEP, 2/WPA-PSK)
  -e <essid> : target selection: network identifier
  -b <bssid> : target selection: access point's MAC
  -p <nbcpu> : # of CPU to use (default: all CPUs)
  -q : enable quiet mode (no status output)
  -C <macs> : merge the given APs to a virtual one
  -l <file> : write key to file. Overwrites file.

Static WEP cracking options:
  -c : search alpha-numeric characters only
  -t : search binary coded decimal chr only
  -n : search the numeric key for FFSIDBM
  -d <mask> : use masking of the key (A15X1C1F:VV)
  -m <macaddr> : MAC address to filter usable packets
  -m <bits> : WEP key length : 64/128/152/256/512
  -i <index> : WEP key index (1 to 4), default: any
  -f <fudge> : bruteforce fudge factor, default: 2
  -k <disable> : disable one attack method (1 to 12)
  -x or -x8 : disable bruteforce for last keybytes
  -x1 : last keybyte bruteforcing (default)
  -x2 : enable last 2 keybytes bruteforcing
  -X : disable bruteforce multithreading
  -y : experimental single bruteforce mode
  -k : use only old Korek attacks (pre-PTW)
  -s : show the key in ASCII while cracking
  -m <num> : specify maximum number of IVs to use
  -D : WEP deconv, skip broken keystreams
  -p <num> : PTW debug: 1: disable Klein, 2: PTW
  -l : run only 1 try to crack key with PTW
  -V : run in visual inspection mode

WEP and WPA-PSK cracking options:
  -w <words> : path to wordlist(s) filename(s)
  -B <file> : path to new session filename
  -B <file> : path to existing session filename

WPA-PSK options:
  -t <file> : create EWSA Project file v3
  -l <str> : PMKID string (hashcat => 10daae)
  -j <file> : create Hashcat v3.6+ file (HCCAPX)
  
```

### How I used it:

I would use Aircrack-ng to audit the security of any Wi-Fi networks the team needs to connect to. Before joining unfamiliar networks, I'd assess the type of encryption in place and test for common vulnerabilities. If needed, I'd simulate password attacks to determine if the network could be easily breached — not to attack, but to assess whether it's safe to use.

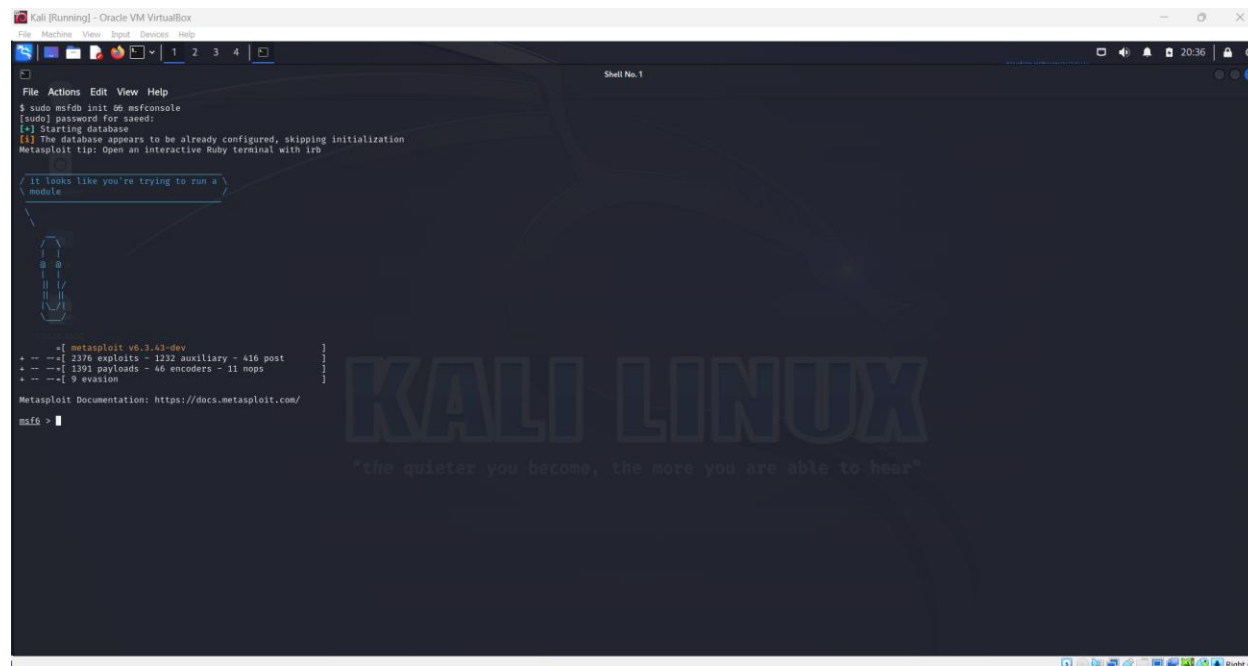
### Value:

Aircrack-ng would help me identify weak or unsafe Wi-Fi networks before the team connects to them. This proactive check reduces the risk of data interception or man-in-the-middle attacks in high-risk environments.

## Tool 3: Metasploit Framework

### Capabilities:

Metasploit is a comprehensive penetration testing platform that simulates attacks using built-in exploits and payloads. It helps security teams uncover and validate vulnerabilities in systems and applications.



### How I used it:

I would use Metasploit to simulate real-world cyberattacks against our own devices before deployment. For example, I might run a scan on each laptop or phone to check for unpatched vulnerabilities. By simulating an exploit attempt, I could evaluate whether a device's configuration would hold up in the field — and take corrective steps before it's too late.

### Value:

Metasploit would allow me to test the strength of our devices under realistic attack conditions and patch weaknesses before bad actors find them. It's a proactive way to reinforce the security of our team's equipment before and during the mission.