SRM Institute of Science and Technology

Department of Computer Applications

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Circular – 2020-21

MCA 1st semester
CYBER SECURITY (PCA20D02J)

Lab Manual

Lab 1: Cyber security attacks case study Submission

Title: Cyber Security Attacks Case Study Submission - Part 1

Aim: To analyze and report on a specific cyber security attack, understanding its methodology, impact, and mitigation strategies.

Procedure:

1. Select a recent or significant cyber security attack (e.g., a major data breach, ransomware attack, or DDoS incident).
2. Research the chosen attack thoroughly, gathering information on:
    - The type of attack.
    - The target(s) and affected parties.
    - The vulnerabilities exploited.
    - The tools and techniques used by the attackers.
    - The impact of the attack (financial, reputational, operational).
    - The detection and response measures taken.
    - Lessons learned and recommended prevention strategies.
3. Structure your findings into a comprehensive report.

Source Code: Not applicable (submission-based).

Input: Research materials, news articles, security reports, and official statements related to the chosen cyber attack.

Expected Output: A detailed written report or presentation summarizing the case study, including analysis of the attack, its impact, and proposed mitigation strategies.

Lab 2: Cyber security attacks case study Submission

Title: Cyber Security Attacks Case Study Submission - Part 2

Aim: To further analyze and present another cyber security attack, focusing on different attack vectors or more in-depth technical aspects.

Procedure:

1. Choose a second distinct cyber security attack, ideally one that highlights different attack methodologies or vulnerabilities compared to Lab 1.
2. Conduct in-depth research on this attack, focusing on:
    - Technical details of the exploit.
    - Indicators of Compromise (IoCs).
    - Forensic analysis findings (if available).
    - Specific security controls that failed or succeeded.
    - Compliance and regulatory implications.
3. Prepare a structured report or presentation detailing your findings.

Source Code: Not applicable (submission-based).

Input: Research materials, technical papers, vulnerability reports, and incident response summaries related to the second chosen cyber attack.

Expected Output: A second comprehensive report or presentation providing a deep dive into the technical aspects and broader implications of the cyber attack.

Lab 3: TCP scanning using NMAP Port scanning using NMAP

Title: TCP and Port Scanning using NMAP

Aim: To understand and practically perform TCP and port scanning using the NMAP (Network Mapper) tool to discover open ports and services on target systems.

Procedure:

1. **Install NMAP:** Ensure NMAP is installed on your system.
2. **Basic TCP Scan:** Use NMAP to perform a basic TCP connect scan on a target IP address or hostname.
    - `nmap <target_IP>`
3. **SYN Stealth Scan:** Perform a SYN (stealth) scan, which is less intrusive and often bypasses basic firewalls.
    - `nmap -sS <target_IP>`
4. **Version Detection Scan:** Scan for open ports and attempt to determine the version of services running on those ports.
    - `nmap -sV <target_IP>`
5. **Specific Port Scan:** Scan only specific ports or a range of ports.
    - `nmap -p 21,22,80,443 <target_IP>`
    - `nmap -p 1-1024 <target_IP>`
6. **Operating System Detection:** Attempt to identify the operating system of the target.
    - `nmap -O <target_IP>`
7. **Aggressive Scan:** Combine several common scan options, including OS detection, version detection, script scanning, and traceroute.
    - `nmap -A <target_IP>`
8. **Output to File:** Save scan results to a file for later analysis.
    - `nmap -oN output.txt <target_IP>`

Source Code:

```
# Basic TCP Connect Scan
nmap 192.168.1.1

# SYN Stealth Scan
nmap -sS 192.168.1.1

# Version Detection Scan
nmap -sV 192.168.1.1

# Scan specific ports
nmap -p 21,22,80,443 192.168.1.1

# Aggressive Scan (includes OS and version detection, script scanning)
nmap -A 192.168.1.1

# Scan and save output to a normal file
nmap -oN nmap_results.txt 192.168.1.1
```

Input: A target IP address or hostname (e.g., `192.168.1.1` or `scanme.nmap.org`).

Expected Output: A list of open, closed, or filtered ports on the target system, along with service names, versions, and potentially OS information, similar to:

```
Starting Nmap 7.91 ( https://nmap.org ) at 2023-10-27 10:00 IST
```

```
Nmap scan report for 192.168.1.1
Host is up (0.002s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu 4ubuntu0.2 (Ubuntu Linux; protocol
2.0)
80/tcp open  http    Apache httpd 2.4.41 ((Ubuntu))

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.23 seconds
```

Lab 4: TCP/UDP connectivity using Net cat

Title: TCP/UDP Connectivity using Netcat - Part 1

Aim: To establish and test basic TCP and UDP network connections using Netcat (nc), demonstrating its utility as a network debugging and data transfer tool.

Procedure:

1. **Install Netcat:** Ensure Netcat is installed on two machines (one acting as a listener, one as a client).
2. **TCP Listener:** On one machine (Server), set up a Netcat listener on a specific TCP port.
    - `nc -lvp <port_number>`
3. **TCP Client:** On the other machine (Client), connect to the server's IP address and port.
    - `nc <server_IP> <port_number>`
4. **Send Data (TCP):** Type messages on either side; they should appear on the other.
5. **UDP Listener:** On one machine (Server), set up a Netcat listener on a specific UDP port.
    - `nc -luvp <port_number>`
6. **UDP Client:** On the other machine (Client), send UDP data to the server's IP address and port.
    - `nc -u <server_IP> <port_number>`
7. **Send Data (UDP):** Type messages on the client; they should appear on the server. Note that UDP is connectionless, so there's no persistent connection.

Source Code:

```
# On Server (listening for TCP on port 12345)
nc -lvp 12345

# On Client (connecting to Server IP 192.168.1.100 on port 12345)
nc 192.168.1.100 12345

# On Server (listening for UDP on port 54321)
nc -luvp 54321

# On Client (sending UDP to Server IP 192.168.1.100 on port 54321)
nc -u 192.168.1.100 54321
```

Input:

- Server IP address (e.g., `192.168.1.100`).
- Port numbers (e.g., `12345` for TCP, `54321` for UDP).
- Text messages to send through the connection.

Expected Output:

- On the server, output indicating that it is listening and then displaying any received messages.
- On the client, output indicating a connection (for TCP) and displaying any received messages.
- Successful transfer of text data between the client and server.

Lab 5: TCP/UDP connectivity using Net cat

Title: TCP/UDP Connectivity using Netcat - Part 2 (Advanced Usage)

Aim: To explore more advanced uses of Netcat, such as file transfer and simple banner grabbing, to demonstrate its versatility in network operations.

Procedure:

1. **File Transfer (TCP - Server sends file):**
    - On Server: `nc -lvp <port> < file_to_send.txt`
    - On Client: `nc <server_IP> <port> > received_file.txt`
2. **File Transfer (TCP - Client sends file):**
    - On Server: `nc -lvp <port> > received_file.txt`
    - On Client: `nc <server_IP> <port> < file_to_send.txt`
3. **Simple Banner Grabbing (HTTP):** Use Netcat to connect to a web server and manually send an HTTP GET request to retrieve the server's banner.
    - `nc <web_server_IP> 80`
    - Then type: `GET / HTTP/1.0` followed by two newlines.

Source Code:

```
# Server sends file 'data.txt' on port 8888
nc -lvp 8888 < data.txt

# Client receives file from Server IP 192.168.1.100 on port 8888
nc 192.168.1.100 8888 > received_data.txt

# Server receives file on port 9999
nc -lvp 9999 > uploaded_file.txt

# Client sends file 'upload.txt' to Server IP 192.168.1.100 on port 9999
nc 192.168.1.100 9999 < upload.txt

# Simple HTTP Banner Grabbing
# Connect to a web server (e.g., example.com) on port 80
nc example.com 80
# After connecting, manually type:
# GET / HTTP/1.0
# (Press Enter twice)
```

Input:

- Server IP address.
- Port numbers.
- Files to send/receive (e.g., `data.txt`, `upload.txt`).
- For banner grabbing: a web server hostname or IP (e.g., `example.com`).

Expected Output:

- Successful transfer of files between machines.
- For banner grabbing, the HTTP response headers from the web server, including the server's banner (e.g., `Server: Apache/2.4.41 (Ubuntu)`).

Lab 6: Perform an experiment to demonstrate sniffing of router traffic by using the tool Wireshark

Title: Sniffing Router Traffic using Wireshark

Aim: To demonstrate the ability to capture and analyze network traffic passing through a router or network interface using the Wireshark network protocol analyzer.

Procedure:

1. **Install Wireshark:** Ensure Wireshark is installed on a machine connected to the network segment you wish to monitor.
2. **Identify Interface:** Open Wireshark and identify the network interface connected to the router or the network segment (e.g., Ethernet, Wi-Fi).
3. **Start Capture:** Select the appropriate interface and start a packet capture.
4. **Generate Traffic:** Generate some network traffic (e.g., browse a website, ping another device, transfer a file).
5. **Apply Filters:** Use Wireshark's display filters to narrow down the captured traffic (e.g., `ip.addr == <router_IP>`, `http`, `dns`, `tcp.port == 80`).
6. **Analyze Packets:** Examine individual packets to understand their headers, protocols, and payloads.
7. **Stop Capture:** Stop the packet capture when sufficient data has been collected.
8. **Save Capture:** Save the captured traffic for later analysis.

Source Code: Not applicable (GUI-based tool). The "source code" here refers to the commands or steps within the Wireshark GUI.

Input:

- A network interface to monitor.
- Network traffic generated by local or remote devices.

Expected Output:

- A live stream of captured network packets displayed in Wireshark.
- Filtered views of traffic, showing specific protocols, source/destination IPs, and port numbers.
- Ability to inspect packet details, revealing information like HTTP requests, DNS queries, TCP handshakes, etc.

Lab 7: Demonstrate how to provide secure data storage , secure data transmission and for creating digital signatures (GnuPG)

Title: Secure Data Operations and Digital Signatures using GnuPG - Part 1

Aim: To practically demonstrate secure data storage (encryption), secure data transmission (encryption for transfer), and the creation of digital signatures using GnuPG (GNU Privacy Guard).

Procedure:

1. **Install GnuPG:** Ensure GnuPG is installed on your system.
2. **Generate Key Pair:** Generate a PGP key pair (public and private keys) for a user.
    - `gpg --full-generate-key`
3. **Export Public Key:** Export the public key to share with others.
    - `gpg --output public_key.asc --armor --export <your_email>`
4. **Encrypt a File (Symmetric):** Encrypt a file using a passphrase (for secure storage on your system).
    - `gpg -c <filename>`
5. **Decrypt a File (Symmetric):** Decrypt the symmetrically encrypted file.
    - `gpg <filename.gpg>`
6. **Encrypt a File (Asymmetric - for transmission):** Encrypt a file for a recipient using their public key.
    - `gpg -e -r <recipient_email> <filename>`
7. **Decrypt a File (Asymmetric):** The recipient decrypts the file using their private key.
    - `gpg <filename.gpg>`

Source Code:

```
# Generate a new PGP key pair (follow prompts)
gpg --full-generate-key

# Export your public key (replace your_email with your key's email)
gpg --output public_key.asc --armor --export your_email@example.com

# Encrypt a file symmetrically (prompts for passphrase)
gpg -c my_secret_data.txt

# Decrypt the symmetrically encrypted file
gpg my_secret_data.txt.gpg

# Encrypt a file for a recipient (replace recipient_email with their key's
email)
gpg -e -r recipient_email@example.com sensitive_document.pdf

# Decrypt the file (recipient uses their private key)
gpg sensitive_document.pdf.gpg
```

Input:

- User details for key generation (name, email).
- Passphrases for key protection and symmetric encryption.
- Files to be encrypted (e.g., `my_secret_data.txt`, `sensitive_document.pdf`).
- Recipient's public key (for asymmetric encryption).

Expected Output:

- Successful generation of a key pair.
- `public_key.asc` file containing the exported public key.
- `.gpg` files created after encryption (e.g., `my_secret_data.txt.gpg`).
- Original files recovered after decryption.
- Confirmation messages from GnuPG for successful operations.

Lab 8: Demonstrate how to provide secure data storage , secure data transmission and for creating digital signatures (GnuPG)

Title: Secure Data Operations and Digital Signatures using GnuPG - Part 2

Aim: To focus on creating and verifying digital signatures using GnuPG, ensuring data integrity and authenticity.

Procedure:

1. **Sign a File:** Create a detached digital signature for a file using your private key.
   - `gpg --output document.sig --detach-sig document.txt`
2. **Verify a Signature:** Verify the digital signature of a file using the signer's public key.
   - `gpg --verify document.sig document.txt`
3. **Sign and Encrypt:** Encrypt a file and simultaneously sign it (for both confidentiality and authenticity).
   - `gpg -se -r <recipient_email> <filename>`
4. **Import Public Key:** Import a public key from another user to your keyring for verification or encryption.
   - `gpg --import public_key_from_other_user.asc`

Source Code:

```
# Create a detached signature for 'report.docx'
gpg --output report.docx.sig --detach-sig report.docx

# Verify the signature for 'report.docx' using 'report.docx.sig'
gpg --verify report.docx.sig report.docx

# Sign and encrypt 'memo.txt' for a recipient
gpg -se -r recipient_email@example.com memo.txt

# Import a public key from a file
gpg --import other_user_public.asc
```

Input:

- Files to be signed (e.g., `document.txt`, `report.docx`, `memo.txt`).
- Recipient's public key (for signing and encrypting).
- Public key files from other users (e.g., `public_key_from_other_user.asc`).

Expected Output:

- `.sig` file created containing the digital signature (e.g., `document.txt.sig`).
- Confirmation message from GnuPG indicating "Good signature" upon successful verification.
- For signed and encrypted files, a `.gpg` file that can be decrypted and whose signature can be verified.
- Confirmation of successful public key import.

Lab 9: Perform an experiment to sniff traffic using ARP Poisoning

Title: Traffic Sniffing using ARP Poisoning

Aim: To understand and demonstrate how ARP (Address Resolution Protocol) poisoning can be used to redirect network traffic, enabling a malicious actor to sniff communications between two hosts on a local network.

Procedure:

1. **Setup Environment:** Set up a controlled lab environment with at least three machines: Attacker, Target A, and Target B, all on the same LAN segment.
2. **Identify IPs:** Determine the IP addresses of all three machines and the MAC addresses of Target A and Target B.
3. **Enable IP Forwarding (Attacker):** Configure the attacker machine to forward IP packets.
4. **Choose ARP Poisoning Tool:** Select an ARP poisoning tool (e.g., `arpspoof` from dsniff suite, `ettercap`).
5. **Start ARP Poisoning:** Use the tool to send forged ARP replies to Target A, claiming to be Target B's MAC address for Target B's IP, and to Target B, claiming to be Target A's MAC address for Target A's IP.
    - `arpspoof -i <interface> -t <target_A_IP> <target_B_IP>`
    - `arpspoof -i <interface> -t <target_B_IP> <target_A_IP>` (run in a separate terminal)
6. **Start Sniffing:** Simultaneously, start a packet sniffer (like Wireshark) on the attacker machine to capture traffic.
7. **Generate Traffic:** Have Target A and Target B communicate (e.g., ping each other, browse a website, transfer a file).
8. **Analyze Captured Traffic:** Observe that traffic between Target A and Target B is now passing through the attacker machine and can be captured by Wireshark.
9. **Stop Poisoning:** Stop the ARP poisoning tool to restore normal network operation.

Source Code:

```
# Enable IP Forwarding on Linux (Attacker machine)
echo 1 > /proc/sys/net/ipv4/ip_forward

# ARP Spoofing using arpspoof (run in two separate terminals)
# In Terminal 1: Poison Target A's ARP cache to think Attacker is Target B
arpspoof -i eth0 -t 192.168.1.100 192.168.1.101

# In Terminal 2: Poison Target B's ARP cache to think Attacker is Target A
arpspoof -i eth0 -t 192.168.1.101 192.168.1.100

# Simultaneously, run Wireshark on the attacker machine to sniff traffic
# (This is a GUI tool, so no command-line code here, but you'd start it and
select the interface)
```

Input:

- IP addresses of Target A and Target B.
- Network interface name on the attacker machine (e.g., `eth0`).
- Traffic generated between Target A and Target B.

Expected Output:

- ARP caches of Target A and Target B showing the attacker's MAC address associated with the other target's IP.
- Wireshark on the attacker machine capturing all traffic flowing between Target A and Target B, even if they are not directly communicating with the attacker.

Lab 10: Perform an experiment how to use DumpSec

Title: Using DumpSec for Security Auditing - Part 1

Aim: To understand and practically use DumpSec, a Windows-based security auditing tool, to extract and analyze security-related information from local or remote Windows systems.

Procedure:

1. **Install DumpSec:** Download and install DumpSec on a Windows machine.
2. **Launch DumpSec:** Run the `DumpSec.exe` executable.
3. **Connect to System:**
    - For local system: Select "Local Machine".
    - For remote system: Select "Remote Machine" and provide credentials if necessary.
4. **Select Information to Dump:** Choose specific categories of security information to extract (e.g., User Accounts, Group Accounts, Permissions on Files/Folders, Registry Permissions, Services).
5. **Start Dump:** Initiate the data extraction process.
6. **Analyze Output:** Review the generated reports, looking for misconfigurations, weak permissions, or unusual accounts.
7. **Save Results:** Save the dumped information to a file for further analysis or reporting.

Source Code: Not applicable (GUI-based tool). The "source code" here refers to the steps within the DumpSec GUI.

Input:

- Local Windows system or remote Windows system's IP/hostname.
- Credentials for remote access (if applicable).
- Selection of security categories to dump.

Expected Output:

- Detailed reports listing:
    - User accounts (including last logon, password age, account status).
    - Group memberships.
    - Permissions (ACLs) on files, folders, and registry keys.
    - Service configurations.
    - Password policies.
- Identification of potential security weaknesses based on the extracted data.

Lab 11: Perform an experiment how to use DumpSec

Title: Using DumpSec for Security Auditing - Part 2 (Advanced Analysis)

Aim: To delve deeper into the analysis capabilities of DumpSec, focusing on identifying specific security vulnerabilities or compliance issues from the extracted data.

Procedure:

1. **Review Saved Data:** Load a previously saved DumpSec report or perform a new dump.
2. **Focus on Permissions:** Analyze file and folder permissions, looking for:
    - "Everyone" or "Authenticated Users" having Write/Full Control permissions on critical system directories.
    - Inherited permissions that grant unintended access.
3. **Examine User Accounts:** Look for:
    - Accounts with blank or easily guessable passwords (though DumpSec doesn't show passwords, it shows password age/expiration).
    - Enabled guest accounts.
    - Accounts with elevated privileges that are not standard.
4. **Audit Services:** Check service configurations for:
    - Services running under highly privileged accounts unnecessarily.
    - Services that are not required but are running.
5. **Export and Filter:** Export the data to a text file or CSV for easier filtering and searching using external tools (e.g., text editor, spreadsheet software).
6. **Report Findings:** Document any identified security issues and propose remediation steps.

Source Code: Not applicable (GUI-based tool). The "source code" here refers to the analytical steps performed on the DumpSec output.

Input:

- DumpSec reports (live or saved).
- Knowledge of common Windows security best practices and vulnerabilities.

Expected Output:

- A list of identified security vulnerabilities or misconfigurations (e.g., overly permissive ACLs, dormant privileged accounts).
- Recommendations for hardening the Windows system based on the audit findings.

Lab 12: Implementing the Secure Sockets Layer (SSL v2/v3) and Transport Layer Security(TLS v1) network protocols

Title: Implementing and Demonstrating SSL/TLS Protocols

Aim: To understand the principles of SSL/TLS and demonstrate their implementation in securing network communication, focusing on how they provide confidentiality, integrity, and authentication.

Procedure:

1. **Understand Concepts:** Review the handshake process, cryptographic algorithms, and certificate usage in SSL/TLS.
2. **Setup a Simple Web Server with SSL/TLS:**
    - Use a web server like Apache or Nginx.
    - Generate a self-signed SSL certificate and private key (for demonstration purposes).
    - Configure the web server to use this certificate and enable HTTPS on port 443.
3. **Client Connection (Browser):** Access the HTTPS-enabled web server from a web browser. Observe the browser's security indicators (e.g., padlock icon, certificate warnings for self-signed certs).
4. **Inspect Traffic (Wireshark):** Use Wireshark to capture traffic between the client and the HTTPS server. Observe that the application data is encrypted, unlike HTTP traffic.
5. **Demonstrate Certificate Trust:** Explain how trusted Certificate Authorities (CAs) validate certificates and how browsers verify this trust.
6. **Simulate Protocol Version Differences (Optional):** If possible with your server/client setup, demonstrate how to force specific SSL/TLS versions (e.g., TLSv1.2, TLSv1.3) and observe the handshake differences.

Source Code:

```
# Example for generating a self-signed certificate using OpenSSL
# This creates a private key (server.key) and a self-signed certificate
(server.crt)
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout server.key -out
server.crt

# Example Apache configuration snippet for enabling SSL/TLS (in httpd-ssl.conf
or similar)
# Ensure mod_ssl is enabled
# <VirtualHost _default_:443>
#     DocumentRoot "/var/www/html"
#     ServerName www.example.com:443
#     SSLEngine on
#     SSLCertificateFile "/etc/ssl/certs/server.crt"
#     SSLCertificateKeyFile "/etc/ssl/private/server.key"
# </VirtualHost>

# Nginx configuration snippet for enabling SSL/TLS
# server {
#     listen 443 ssl;
#     server_name example.com;
#     ssl_certificate /etc/nginx/ssl/server.crt;
#     ssl_certificate_key /etc/nginx/ssl/server.key;
#     location / {
#         root /var/www/html;
#         index index.html;
```

```
#       }
# }
```

Input:

- A web server (e.g., Apache, Nginx).
- OpenSSL for certificate generation.
- A web browser.
- Wireshark (optional, for traffic inspection).

Expected Output:

- A functional HTTPS web server accessible via a web browser.
- Browser indicating a secure connection (though with a warning for self-signed certificates).
- Wireshark capture showing encrypted application data when communicating over HTTPS, contrasting with unencrypted HTTP traffic.
- Understanding of the SSL/TLS handshake process and its security benefits.

Lab 13: Setup a honey pot on network .

Title: Setting Up a Honeypot on a Network - Part 1

Aim: To understand the concept of a honeypot and practically set up a basic low-interaction honeypot to attract and detect malicious activity on a network.

Procedure:

1. **Understand Honeypots:** Learn about different types of honeypots (low-interaction, high-interaction) and their purpose.
2. **Choose Honeypot Software:** Select a suitable low-interaction honeypot (e.g., `Dionaea`, `Kippo` (for SSH), `Cowrie` (modern SSH/Telnet), `HoneyPy`, `T-Pot`). For simplicity, a basic `netcat` listener or a simple Python script can also act as a rudimentary honeypot.
3. **Prepare a Dedicated Machine:** Set up a dedicated virtual machine or isolated physical machine for the honeypot. It should ideally have no critical services or data.
4. **Install and Configure Honeypot:** Install the chosen honeypot software and configure it to listen on common ports (e.g., 21 (FTP), 22 (SSH), 23 (Telnet), 80 (HTTP), 445 (SMB), 3389 (RDP)).
    - Ensure logging is enabled and configured to store attack data.
5. **Network Placement:** Place the honeypot in a network segment where it can be easily discovered by attackers (e.g., in a DMZ or a less protected segment, but still isolated from critical assets).
6. **Start Honeypot:** Launch the honeypot service.
7. **Monitor Initial Activity:** Observe the honeypot's logs for any incoming connection attempts or interactions.

Source Code:

```
# Example of a very basic Netcat-based honeypot (low-interaction, for
demonstration)
# This will listen on port 23 (Telnet) and log any incoming data to a file.
# Note: This is extremely basic and not a full-fledged honeypot.
nc -lvp 23 >> honeypot_log.txt

# For a more realistic setup, you would install a dedicated honeypot like
Cowrie:
# (These are general steps, refer to Cowrie's official documentation for exact
commands)
# 1. Install dependencies (Python, virtualenv, etc.)
# 2. Clone Cowrie repository: git clone https://github.com/cowrie/cowrie.git
# 3. Create virtual environment and install requirements:
#    cd cowrie
#    python3 -m venv cowrie-env
#    source cowrie-env/bin/activate
#    pip install -r requirements.txt
# 4. Configure cowrie.cfg (e.g., listening ports, logging)
# 5. Start Cowrie: bin/cowrie start
```

Input:

- A dedicated machine (VM or physical).
- Honeypot software configuration.
- Network traffic from potential attackers.

Expected Output:

- The honeypot service running and listening on configured ports.
- Log files generated by the honeypot, detailing:
    - Source IP addresses of connection attempts.
    - Timestamps of interactions.
    - Attempted commands or data sent by attackers.
    - Credentials tried (if applicable to the honeypot type).

Lab 14: Monitor the honey pot on network

Title: Monitoring and Analyzing Honeypot Data

Aim: To continuously monitor the activity on a deployed honeypot, analyze the collected data, and derive insights into attack patterns, attacker methodologies, and common vulnerabilities being exploited.

Procedure:

1. **Ensure Honeypot is Running:** Verify that the honeypot configured in Lab 13 is still active and logging.
2. **Access Logs:** Locate and access the log files generated by the honeypot software.
3. **Real-time Monitoring:** Use `tail -f` or a log viewer to monitor logs in real-time for new activity.
4. **Analyze Log Data:**
    - **Source IPs:** Identify the source IP addresses of attackers. Perform reverse lookups or use threat intelligence feeds to identify their origin.
    - **Attack Types:** Categorize the types of attacks (e.g., brute-force attempts, vulnerability scans, exploitation attempts).
    - **Payloads/Commands:** Examine any commands executed or payloads delivered by attackers.
    - **Timelines:** Track the timing and frequency of attacks.
5. **Visualize Data (Optional):** If the honeypot supports it or if you use a SIEM (Security Information and Event Management) system, visualize the attack data (e.g., geographical origin of attacks, most targeted ports).
6. **Report Findings:** Document the observed attack trends, common vulnerabilities targeted, and any unique attacker behaviors. Use this information to improve defensive strategies.

Source Code:

```
# Example: Viewing Cowrie logs
# Cowrie logs are typically in JSON format, often under cowrie/log/cowrie.json
# You can use tools like `jq` to parse them or just view them directly.
tail -f cowrie/log/cowrie.json

# Example: Filtering logs for specific attack types (conceptual, depends on log
format)
# grep "login attempt failed" cowrie/log/cowrie.json
# grep "command: wget" cowrie/log/cowrie.json
```

Input:

- Log files generated by the honeypot.
- Threat intelligence data (optional, for enriching IP information).

Expected Output:

- A clear understanding of the types of attacks targeting your honeypot.
- Identification of common attack sources and patterns.
- Insights into attacker tools and techniques.
- A summary report detailing observed malicious activity.

Lab 15: Demonstrate intrusion detection system (ids) using any tool (snort or any other s/w)

Title: Demonstrating Intrusion Detection System (IDS) using Snort

Aim: To understand the principles of Intrusion Detection Systems (IDS) and practically demonstrate their functionality using Snort, a widely used open-source network intrusion detection system.

Procedure:

1. **Understand IDS Concepts:** Learn about signature-based vs. anomaly-based IDS, common IDS architectures, and the role of rules.
2. **Install Snort:** Install Snort on a dedicated machine or VM that can monitor network traffic (e.g., in promiscuous mode on a network interface).
3. **Configure Snort Rules:**
    - Download and configure Snort rules (community rules or custom rules).
    - Understand the structure of Snort rules (action, protocol, source/destination IP/port, direction, options).
    - Create a simple custom rule (e.g., to detect a specific string in HTTP traffic or a specific ICMP packet).
4. **Run Snort in Sniffer Mode:** Run Snort to simply capture and display packets (similar to Wireshark, but text-based).
    - `snort -vde -i <interface>`
5. **Run Snort in IDS Mode (Alerts):** Run Snort in NIDS mode to detect and alert on suspicious activity based on configured rules.
    - `snort -A console -c /etc/snort/snort.conf -i <interface>`
6. **Generate Test Traffic:** Generate traffic that should trigger your custom or standard Snort rules (e.g., ping a specific IP, attempt a known exploit signature, visit a specific URL containing a keyword in your rule).
7. **Analyze Alerts:** Observe Snort's console output or log files for generated alerts.
8. **Interpret Alerts:** Understand why an alert was triggered and what information it provides (source/destination, rule ID, message).

Source Code:

```
# Example Snort Rule (save in a .rules file, e.g., local.rules)
# Alert when "test_string" is found in HTTP traffic
alert tcp any any -> any 80 (msg:"HTTP test string detected";
content:"test_string"; classtype:misc-activity; sid:1000001; rev:1;)

# Run Snort in packet dumping mode
snort -vde -i eth0

# Run Snort in NIDS (Network Intrusion Detection System) mode with console
alerts
# Assuming snort.conf is in /etc/snort/
snort -A console -c /etc/snort/snort.conf -i eth0

# To generate test traffic that triggers the rule above:
# On another machine, try to access a web server on port 80 and send
"test_string"
# e.g., using Netcat:
# nc <web_server_IP> 80
# GET /index.html HTTP/1.0
```

```
# User-Agent: test_string
# (Press Enter twice)
```

Input:

- A network interface to monitor (e.g., `eth0`).
- Snort configuration file (`snort.conf`) and rule files.
- Test network traffic designed to trigger IDS alerts.

Expected Output:

- Snort running successfully in IDS mode.
- Alert messages generated by Snort when suspicious traffic (matching defined rules) is detected.
- Log files containing detailed information about triggered alerts (timestamps, source/destination IPs, rule IDs, alert messages).
- Understanding of how Snort identifies and reports network intrusions.