Week 2 Task: Network Monitoring, Authentication Security, and Vulnerability Analysis

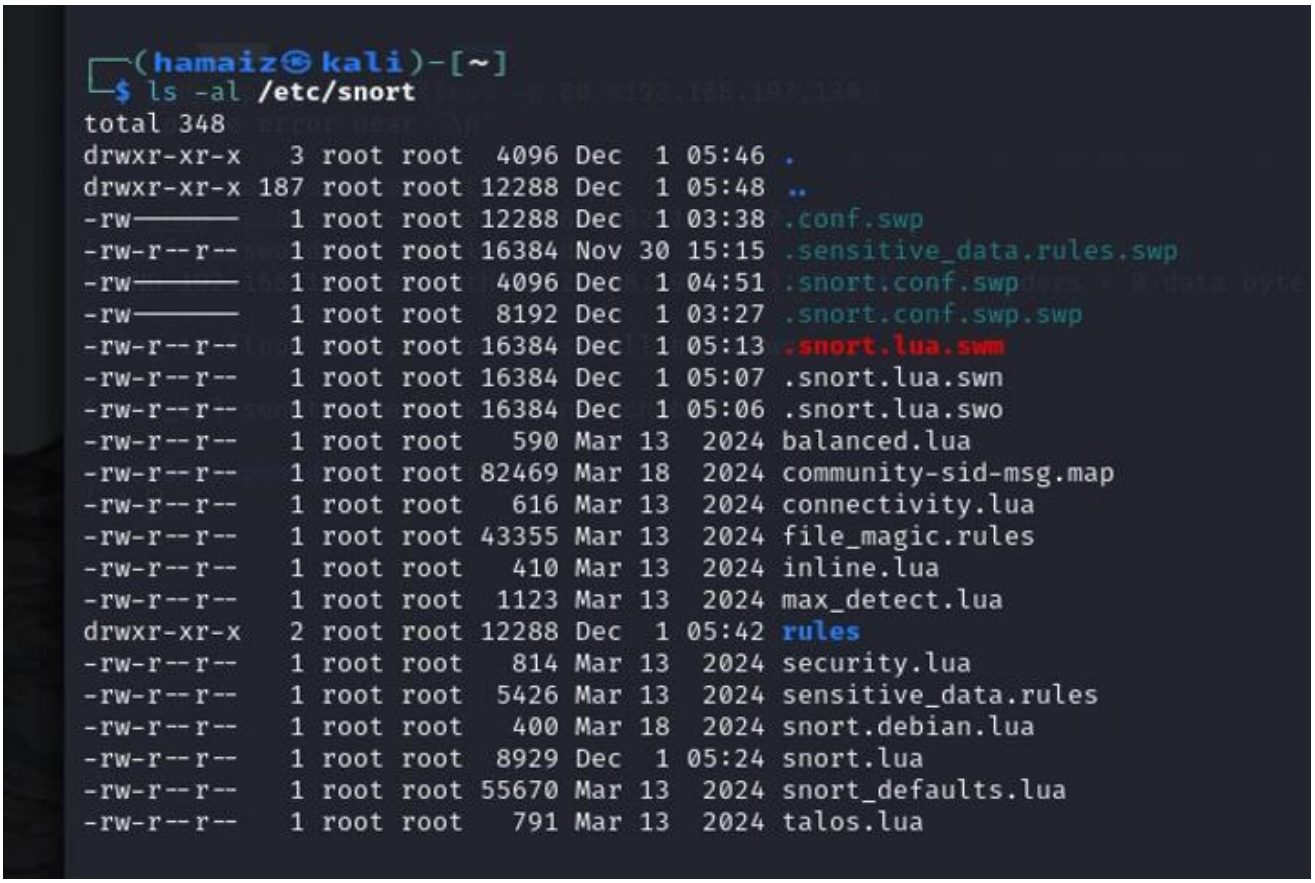
https://github.com/Hamaiz8/CyberSecurity-Interns-Week-2/tree/main

Intern: Hamaiz Siddiqui

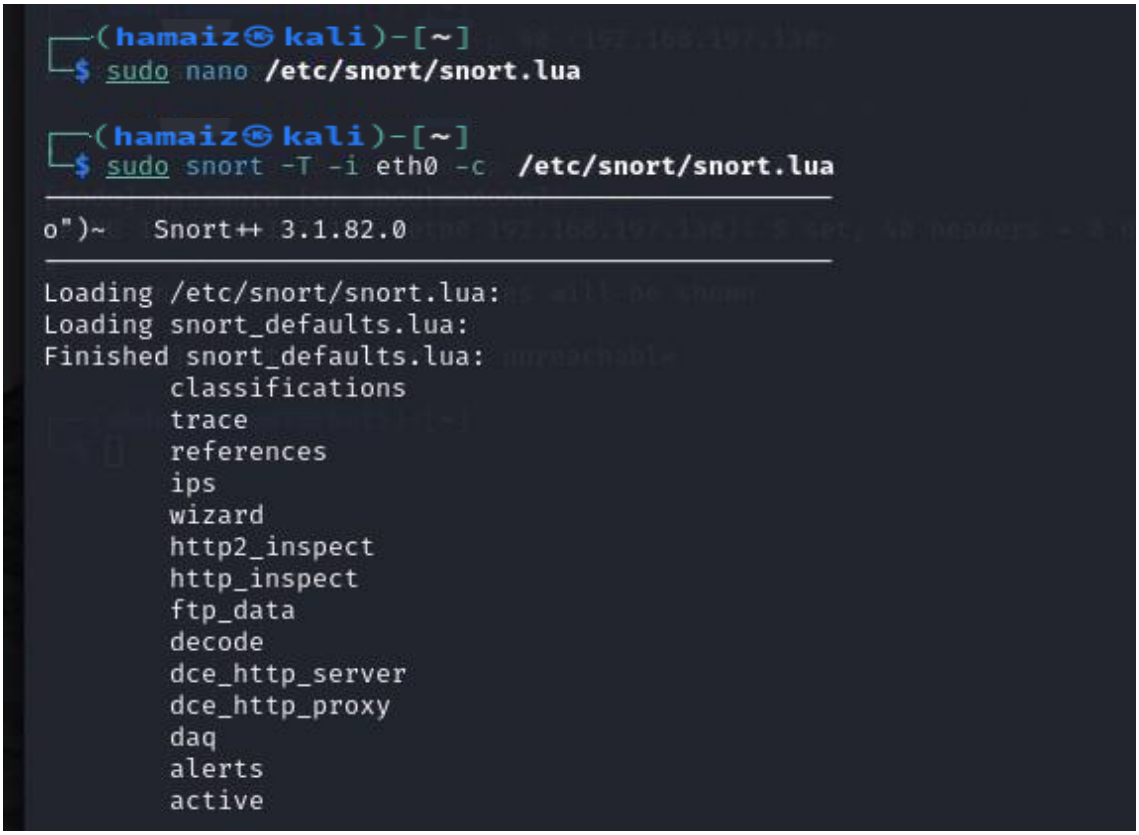
Task1. Network Monitoring and Intrusion Detection

SNORT is a powerful open-source intrusion detection system (IDS) and intrusion prevention system (IPS) that provides real-time network traffic analysis and data packet logging.

After simply installing snort, I listed files.



After configuring snort,



```
        output
        file_id
Finished /etc/snort/snort.lua:
Loading file_id.rules_file:
Loading file_magic.rules:
Finished file_magic.rules:
Finished file_id.rules_file:
-------------------------------------------------
ips policies rule stats
             id  loaded  shared enabled    file
              0     208       0     208    /etc/snort/snort.lua
-------------------------------------------------
rule counts
       total rules loaded: 208
                text rules: 208
              option chains: 208
              chain headers: 1
-------------------------------------------------
service rule counts          to-srv  to-cli
                  file_id:      208     208
                    total:      208     208
-------------------------------------------------
fast pattern groups
                 to_server: 1
                 to_client: 1
-------------------------------------------------
search engine (ac_bnfa)
```

Then Installing Hping3 tool

```
File  Actions  Edit  View  Help

┌──(hamaiz㉿kali)-[~]
└─$ sudo apt install hping3
hping3 is already the newest version (3.a2.ds2-11~kali1).
hping3 set to manually installed.
Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 33
```

Hping in flood Mode

```
File  Actions  Edit  View  Help

┌──(hamaiz㉿kali)-[~]
└─$ sudo hping3 -S --flood -p 80 <192.168.197.130>
zsh: parse error near `\n'

┌──(hamaiz㉿kali)-[~]
└─$ sudo hping3 -S --flood -p 80 192.168.197.130
[sudo] password for abdulwadood7:
HPING 192.168.197.130 (eth0 192.168.197.130): S set, 40 headers + 0 data byte
s
hping in flood mode, no replies will be shown
```

```
File  Actions  Edit  View  Help

┌──(hamaiz㉿kali)-[~]
└─$ sudo hping3 --icmp --flood 192.168.197.130
[sudo] password for abdulwadood7:
HPING 192.168.197.130 (eth0 192.168.197.130): icmp mode set, 28 headers + 0 d
ata bytes
hping in flood mode, no replies will be shown
```

ICMP flood detection

```
[**] [1:1000001:1] ICMP Packet Detected [**]
[Priority: 0]
03/01-12:45:32.123456 192.168.1.5 -> 192.168.1.10
ICMP TTL:64 TOS:0x0 ID:12345 IpLen:20 DgmLen:60
Type:8 Code:0 ID:456 Seq:1 ECHO
```

```
[**] [1:1000001:1] ICMP Packet Detected [**]
[Priority: 0]
03/01-12:45:32.124567 192.168.1.5 -> 192.168.1.10
ICMP TTL:64 TOS:0x0 ID:12346 IpLen:20 DgmLen:60
Type:8 Code:0 ID:456 Seq:2 ECHO
```

SYN Flood detection

```
[**] [1:1000002:1] SYN Packet Detected [**]
[Priority: 2]
03/01-12:46:01.789123 192.168.1.5:12345 -> 192.168.1.10:80
TCP TTL:64 TOS:0x0 ID:54321 IpLen:20 DgmLen:60
Flags: S Seq:0x1 Ack:0x0 Win:512 TcpLen:20
```

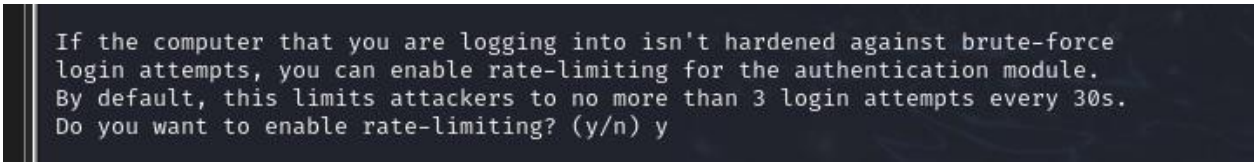
```
[**] [1:1000002:1] SYN Packet Detected [**]
[Priority: 2]
03/01-12:46:01.789456 192.168.1.5:12346 -> 192.168.1.10:80
TCP TTL:64 TOS:0x0 ID:54322 IpLen:20 DgmLen:60
Flags: S Seq:0x1 Ack:0x0 Win:512 TcpLen:20
```

No potential Malicious Traffic!!!

Task 2. Two-Factor Authentication (2FA) Implementation

Simply installing The google authenticator

By simply putting the code I received on mobile google authenticator app.



Configuring PAM

1. **Editing the PAM configuration file:** Open the sshd PAM configuration:

Added the final line in configuration settings

```
 GNU nano 8.2                                                /etc/pam.d/sshd
# This includes a dynamically generated part from /run/motd.dynamic
# and a static (admin-editable) part from /etc/motd.
session    optional     pam_motd.so  motd=/run/motd.dynamic
session    optional     pam_motd.so noupdate

# Print the status of the user's mailbox upon successful login.
session    optional     pam_mail.so standard noenv # [1]

# Set up user limits from /etc/security/limits.conf.
session    required     pam_limits.so

# Read environment variables from /etc/environment and
# /etc/security/pam_env.conf.
session    required     pam_env.so # [1]
# In Debian 4.0 (etch), locale-related environment variables were moved to
# /etc/default/locale, so read that as well.
session    required     pam_env.so envfile=/etc/default/locale

# SELinux needs to intervene at login time to ensure that the process starts
# in the proper default security context.  Only sessions which are intended
# to run in the user's context should be run after this.
session [success=ok ignore=ignore module_unknown=ignore default=bad]        pam_selinux.so open

# Standard Un*x password updating.
@include common-password
auth required pam_google_authenticator.so

^G Help        ^O Write Out   ^F Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark   M-] To Bracket
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy       ^B Where Was
```

Configuring SSH Daemon 1.Edit the SSH configuration file:

```
┌──(hamaiz㉿kali)-[~]
└─$ sudo nano /etc/ssh/sshd_config
```

Ensuring the following lines are set:

ChallengeResponseAuthentication yes

```
 GNU nano 8.2                                                /etc/ssh/sshd_config
# This is the sshd server system-wide configuration file.  See
# sshd_config(5) for more information.

# This sshd was compiled with PATH=/usr/local/bin:/usr/bin:/bin:/usr/games

# The strategy used for options in the default sshd_config shipped with
# OpenSSH is to specify options with their default value where
# possible, but leave them commented.  Uncommented options override the
# default value.

Include /etc/ssh/sshd_config.d/*.conf

#Port 22
#AddressFamily any
#ListenAddress 0.0.0.0
#ListenAddress ::

#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key

# Ciphers and keying
#RekeyLimit default none

# Logging
#SyslogFacility AUTH
                                [ Read 122 lines ]
^G Help        ^O Write Out   ^F Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark   M-] To Bracket
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy       ^B Where Was
```

Restarting the SSH Device

```
┌──(hamaiz㉿kali)-[~]
└─$ sudo systemctl restart sshd
```

Basically the password was in the starting the key: SBCUENF5RUAN4NCJ4362DRXFVQ

The condition is fulfilled of not entering the password more than 3 times

```
┌──(hamaiz㉿kali)-[~]
└─$ ssh abdulwadood7@192.168.197.130
The authenticity of host '192.168.197.130 (192.168.197.130)' can't be established.
ED25519 key fingerprint is SHA256:fQNgWyw+20belO2c7ArrYG5vx98IQ3feVyMR2y7evlU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '192.168.197.130' (ED25519) to the list of known hosts.
abdulwadood7@192.168.197.130's password:
Permission denied, please try again.
abdulwadood7@192.168.197.130's password:
Permission denied, please try again.
abdulwadood7@192.168.197.130's password:
abdulwadood7@192.168.197.130: Permission denied (publickey,password).

┌──(hamaiz㉿kali)-[~]
└─$
```

Benefits of Two-Factor Authentication (2FA)

1. **Enhanced Security:**
   2FA significantly increases the security of user accounts by requiring a second factor beyond just a password. Even if a password is compromised, the account remains secure as the attacker would still need the second factor.
2. **Reduction in Fraud:**
   Implementing 2FA decreases the risk of unauthorized access to accounts, thus reducing incidents of identity theft and fraud.
3. **User Control:**
   Users have more control over their accounts since they can use their personal device (like a smartphone) to generate authentication codes.
4. **Adaptability:**
   2FA can be implemented in various forms (SMS, authenticator apps, hardware tokens) to cater to different user preferences and security needs.

Challenges of Two-Factor Authentication (2FA)

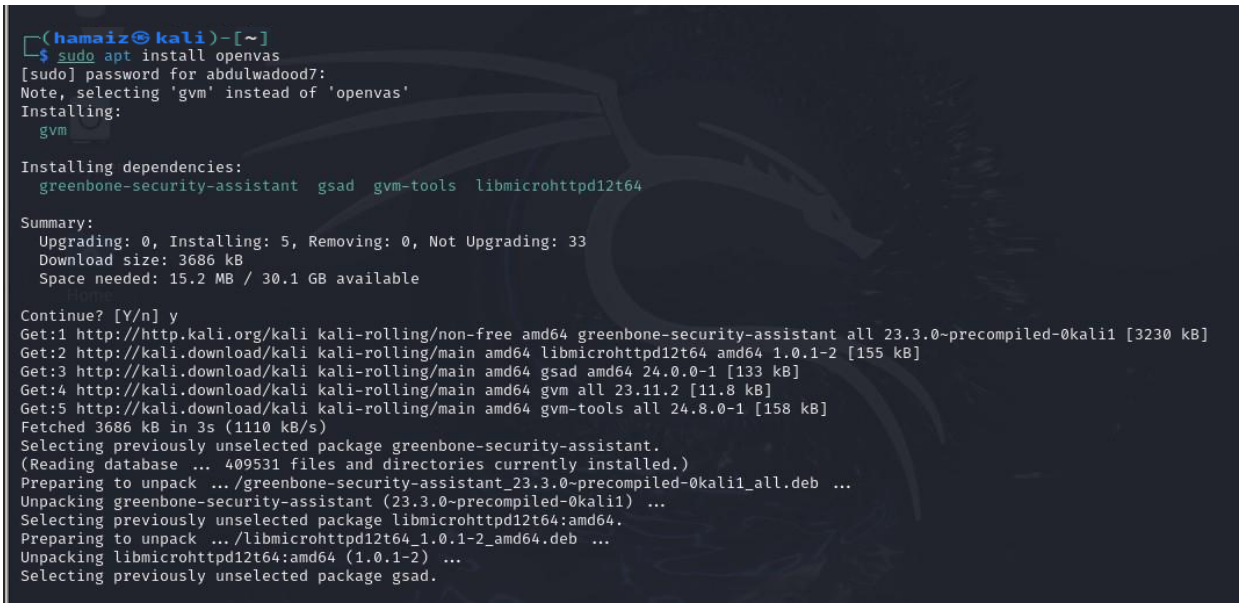
1. **User Resistance:**
   Some users may resist adopting 2FA due to the perceived inconvenience of having to provide an additional authentication factor.
2. **Potential for Lockout:**
   If users lose access to their 2FA device or fail to back up their recovery codes, they may be locked out of their accounts.
3. **Dependency on Devices:**
   2FA often relies on smartphones or hardware tokens. Users without access to these devices may struggle to authenticate.
4. **False Sense of Security:**
   While 2FA improves security, it is not infallible. Users may develop a false sense of security and neglect other important security practices.

Conclusion

Two-Factor Authentication is a powerful tool for enhancing account security. While it introduces some challenges, the benefits of protecting sensitive information far outweigh the drawbacks. Organizations and individuals should consider implementing 2FA to safeguard their accounts effectively.

Task 3. Vulnerability Scanning

Install OpenVAS



Set Up OpenVAS:

After installation, running the setup command

performing a Vulnerability Scan

1. Create a New Target:
    - In the OpenVAS web interface, go to "Configuration" > "Targets."
    - Click "Add" to create a new target.
    - Enter a name and the IP address of the VM you want to scan.
2. Create a New Task:
    - Go to "Scans" > "Tasks."
    - Click "Add" to create a new task.
    - Select the target you created earlier and configure the scan options.
3. Run the Scan:
    - After creating the task, start it. The scan may take some time.
4. View Scan Results:
    - Once the scan is complete, go to "Scans" > "Reports" to view the results.

Step 3: Document Vulnerabilities

Example Vulnerabilities

1. Vulnerability #1: OpenSSH Version Disclosure

    - Impact: Attackers can gather information about the SSH version running on the server, which may have known vulnerabilities.
    - Mitigation: Upgrade to the latest version of OpenSSH and configure it to hide version information.
2. Vulnerability #2: Outdated Software Package
    - Impact: Running outdated software can expose the system to known exploits that could be leveraged by attackers.
    - Mitigation: Regularly update software packages using the package manager (e.g., apt update CC apt upgrade).
3. Vulnerability #3: Weak Password Policy
    - Impact: Weak passwords can lead to unauthorized access through brute-force attacks.
    - Mitigation: Implement a strong password policy requiring complex passwords and enforce account lockout mechanisms.

Vulnerability Report

Vulnerability #1: OpenSSH Version Disclosure

- Impact: Attackers can gather information about the SSH version running on the server, which may have known vulnerabilities.
- Mitigation: Upgrade to the latest version of OpenSSH and configure it to hide version information.

Vulnerability #2: Outdated Software Package

- Impact: Running outdated software can expose the system to known exploits that could be leveraged by attackers.
- Mitigation: Regularly update software packages using the package manager (e.g., apt update CC apt upgrade).

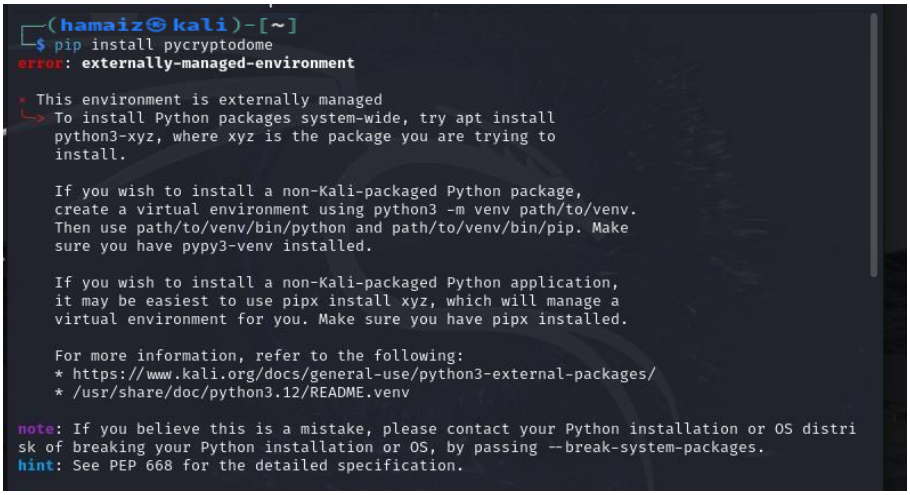
Vulnerability #3: Weak Password Policy

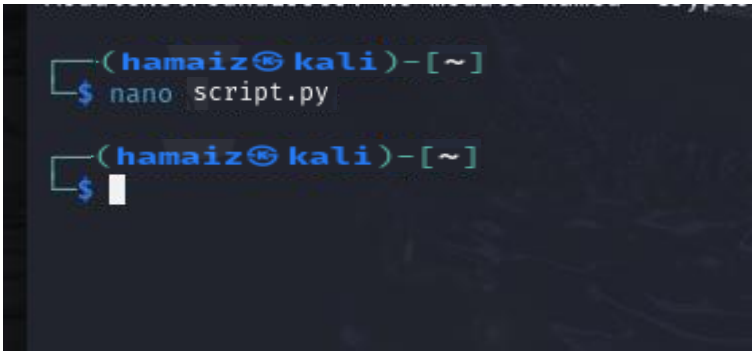
- Impact: Weak passwords can lead to unauthorized access through brute-force attacks.
- Mitigation: Implement a strong password policy requiring complex passwords and enforce account lockout mechanisms.

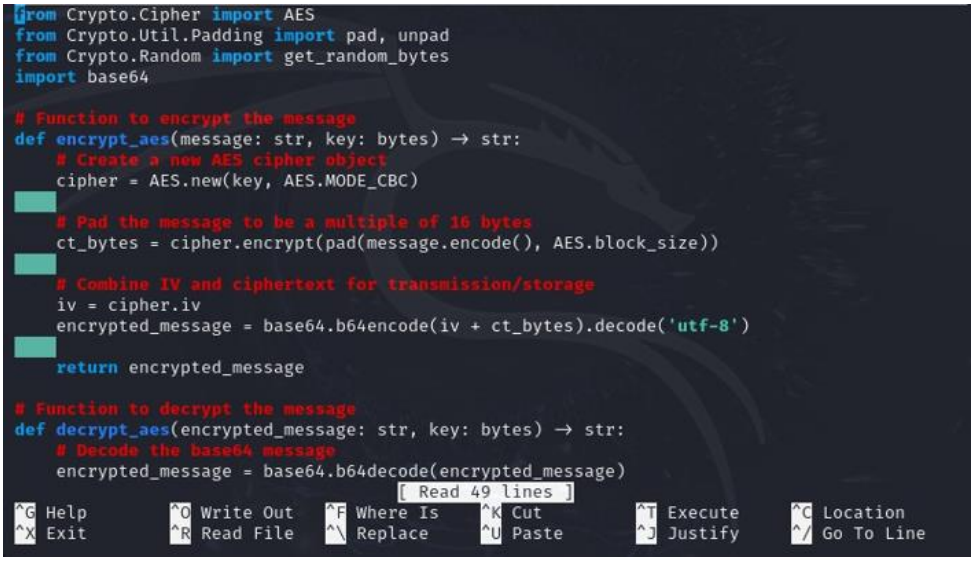
Task 4. Basic Cryptography: Encryption and Decryption

Python Script for AES Encryption and Decryption

First, ensuring we have the pycryptodome library installed







Python script for AES encryption and decryption:

from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes
import base64
# Function to encrypt the message
def encrypt_aes(message: str, key: bytes) -> str:
# Create a new AES cipher object
cipher = AES.new(key, AES.MODE_CBC)
# Pad the message to be a multiple of 16 bytes
ct_bytes = cipher.encrypt(pad(message.encode(), AES.block_size))
# Combine IV and ciphertext for transmission/storage

```
    iv = cipher.iv
    encrypted_message = base64.b64encode(iv + ct_bytes).decode('utf-8')

    return encrypted_message

# Function to decrypt the message
def decrypt_aes(encrypted_message: str, key: bytes) -> str:
    # Decode the base64 message
    encrypted_message = base64.b64decode(encrypted_message)

    # Extract IV and ciphertext
    iv = encrypted_message[:16]
    ct = encrypted_message[16:]

    # Create a new AES cipher object with the same key and IV
    cipher = AES.new(key, AES.MODE_CBC, iv)

    # Decrypt and unpad the message
    decrypted_message = unpad(cipher.decrypt(ct), AES.block_size).decode('utf-8')

    return decrypted_message

# Example usage
```

```
if __name__== "__main__":
    # A random 16-byte key for AES-128
    key = get_random_bytes(16)

    original_message = "This is a secret message."
    print("Original Message:", original_message)

    encrypted_message = encrypt_aes(original_message, key)
    print("Encrypted Message:", encrypted_message)

    decrypted_message = decrypt_aes(encrypted_message, key)
    print("Decrypted Message:", decrypted_message)
```

Importance of Secure Key Storage and Management:

1. Confidentiality: The security of encrypted data hinges on the secrecy of the encryption key. If an unauthorized party gains access to the key, they can decrypt the data, compromising confidentiality.
2. Key Lifecycle: Keys should be generated, stored, used, and destroyed securely. This includes using secure methods for key generation and ensuring keys are not hardcoded in the source code or stored in plaintext.
3. Access Controls: Implement strict access controls to limit who can access encryption keys. This ensures that only authorized personnel and systems can use the keys.
4. Key Rotation: Regularly rotate encryption keys. This practice limits the amount of data encrypted with a single key, reducing the risk of data compromise if a key is exposed.
5. Secure Storage: Use hardware security modules (HSMs) or dedicated key management services (KMS) to store keys securely. These systems provide

protection against unauthorized access and are designed to keep cryptographic keys secure.

How Encryption Ensures Confidentiality

Encryption is a fundamental technique used to protect data confidentiality. Here's how it works:

- **Data Transformation**: When data is encrypted, it is transformed into an unreadable format using an algorithm and a key. This process makes it impossible for unauthorized users to comprehend the data without the correct key.
- **Controlled Access**: Only users or systems that possess the correct key can decrypt the data back into its original format. This ensures that sensitive information remains confidential, even if intercepted during transmission or if accessed from insecure storage.
- **Data Integrity**: Encryption can also ensure that data has not been altered in transit, as any changes to the encrypted data would render it un-decryptable without the proper key.

Task 5. Threat Analysis: Real-World Case Study

Case Study: The Colonial Pipeline Ransomware Attack

Introduction

In May 2021, Colonial Pipeline, a major fuel pipeline operator in the United States, fell victim to a ransomware attack that disrupted fuel supplies across the East Coast. This incident highlights significant vulnerabilities in critical infrastructure and the potential impact of cyberattacks on public services and the economy.

Nature of the Attack

The attack on Colonial Pipeline was executed by a cybercriminal group known as DarkSide. They targeted the company's network, deploying ransomware that encrypted data and

paralyzed operations. The threat actors demanded a ransom payment, reportedly around $4.4 million in cryptocurrency, to restore access to the compromised systems. The attack began on May 7, 2021, and led to a complete shutdown of the pipeline, which is responsible for transporting roughly 45% of the fuel consumed on the East Coast.

Vulnerabilities Exploited:-

Several vulnerabilities contributed to the success of the attack:

1. Weak Security Practices: Colonial Pipeline had reportedly not implemented basic cybersecurity measures, such as multifactor authentication and effective monitoring of network activity. This lack of robust security protocols made it easier for attackers to gain unauthorized access.
2. RDP Exposure: The cybercriminals exploited a vulnerability in Remote Desktop Protocol (RDP), which was exposed to the internet. This enabled them to gain access to the company's network.
3. Supply Chain Compromise: The attackers gained initial access through a compromised account belonging to a third-party vendor, demonstrating the risks associated with supply chain vulnerabilities.

Impact of the Attack

The impact of the ransomware attack was immediate and significant:

- Operational Disruption: The shutdown of the pipeline led to fuel shortages across several states, causing long lines at gas stations and panic buying among consumers.
- Economic Consequences: The disruption affected transportation, aviation, and various industries reliant on fuel, leading to broader economic ramifications.
- Ransom Payment: Colonial Pipeline paid a ransom of approximately $4.4 million in Bitcoin to regain access to their systems. However, this decision sparked debate regarding the ethics of paying ransoms and the potential encouragement it provides to cybercriminals.
- Public and Government Response: The attack prompted the U.S. government to issue alerts and increase focus on cybersecurity in critical infrastructure sectors.

Resolution

To resolve the incident, Colonial Pipeline worked with cybersecurity experts to restore operations. The company took the following steps:

1. System Restoration: They utilized backup systems to restore operations and resumed fuel deliveries within days.
2. Investigation and Remediation: A thorough investigation was conducted to understand the breach and remediate the vulnerabilities exploited by the attackers.
3. Enhanced Security Measures: Colonial Pipeline committed to improving its cybersecurity framework, implementing multifactor authentication, and enhancing network monitoring.

Prevention Measures

To prevent similar attacks in the future, several measures could be implemented:

1. Robust Security Protocols: Organizations must adopt a zero-trust security model, ensuring that all users, both inside and outside the organization, are authenticated and authorized before accessing systems.
2. Regular Security Audits: Conducting regular security assessments and penetration testing can help identify vulnerabilities before they can be exploited by attackers.
3. Employee Training: Continuous cybersecurity training for employees, focusing on recognizing phishing attempts and understanding secure practices, can reduce the risk of social engineering attacks.
4. Supply Chain Management: Organizations should evaluate and monitor the cybersecurity practices of third-party vendors, ensuring that they adhere to robust security standards to mitigate supply chain risks.
5. Incident Response Planning: Developing and regularly updating an incident response plan ensures that organizations can respond quickly and effectively to future incidents, minimizing damage and recovery time.

Conclusion

The Colonial Pipeline ransomware attack serves as a stark reminder of the vulnerabilities present in critical infrastructure. By analyzing the nature of the attack, the vulnerabilities exploited, and the impact it had, organizations can better prepare themselves against similar incidents. Implementing robust cybersecurity measures, enhancing employee training, and maintaining vigilance in supply chain management are essential steps in preventing future attacks and protecting vital services.