Day 9: Incident Response, Simulating and Investigating Suspicious Network Activity on Linux

This section documents how suspicious outbound network activity was simulated, detected, investigated, and mitigated on a Linux system.

Objective of the Simulation

The goal of this exercise was to replicate a real-world scenario where malware on a Linux system makes **periodic outbound connections** to a remote server, often used for:

- Command-and-Control (C2) communication
- Data exfiltration
- Beaconing to an attacker
- Botnet callbacks

This creates a realistic environment to practice **incident detection**, **analysis**, and **response**.

1. Simulating the Suspicious Activity

The first action was to intentionally generate abnormal outbound traffic to an external IP address (45.13.220.98). This simulated a typical malicious beaconing behavior, where malware repeatedly contacts a command-and-control (C2) server. The following command was used:

```
ubuntu@ubuntu:~$ nohup bash -c 'while true; do curl http://45.13.220.98/ping >/d
ev/null 2>&1; sleep 30; done' &
[2] 4417
ubuntu@ubuntu:~$ nohup: ignoring input and appending output to 'nohup.out'
```

This creates a background process that silently sends an HTTP request every 30 seconds. The use of **nohup** ensures the process keeps running even after the user logs out, simulating real-world stealth persistence. Redirecting output to `/dev/null` hides visible traces from the console. This behavior is suspicious because normal system processes rarely contact external IPs repeatedly in an infinite loop.

2. Detecting the Suspicious Process

Once the outbound traffic generator was launched, the next step was detection. The `ps` command was used to enumerate running processes:

```
ubuntu@ubuntu:~$ ps aux | grep curl
ubuntu      4371  0.0  0.0   9940  3508 pts/0    S    21:49   0:00 bash -c while
 true; do curl http://45.13.220.98/ping >/dev/null 2>&1; sleep 30; done
ubuntu      4417  0.0  0.0   9940  3504 pts/0    S    21:50   0:00 bash -c while
 true; do curl http://45.13.220.98/ping >/dev/null 2>&1; sleep 30; done
ubuntu      4424  0.0  0.0   9144  2248 pts/1    S+   21:51   0:00 grep --color=
auto curl
```

This revealed two Bash processes executing the looped `curl` command. The presence of long-running curl processes; especially those not tied to user activity, often indicates unauthorized data exfiltration or malware beaconing. Grabbing the process IDs (PIDs) is essential for deeper inspection and termination.

3. Inspecting the Process with lsof

To understand exactly what the suspicious process was doing, the `lsof` tool was invoked:

```
ubuntu@ubuntu:~$ sudo lsof -p 4371
lsof: WARNING: can't stat() fuse.portal file system /run/user/1000/doc
      Output information may be incomplete.
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
      Output information may be incomplete.
COMMAND  PID   USER   FD   TYPE DEVICE SIZE/OFF    NODE NAME
bash    4371 ubuntu  cwd    DIR    8,2     4096 2621442 /home/ubuntu
bash    4371 ubuntu  rtd    DIR    8,2     4096       2 /
bash    4371 ubuntu  txt    REG    8,2  1446024 4981306 /usr/bin/bash
bash    4371 ubuntu  mem    REG    8,2  5719296 5027659 /usr/lib/locale/locale-a
rchive
bash    4371 ubuntu  mem    REG    8,2  2125328 4983290 /usr/lib/x86_64-linux-gn
u/libc.so.6
bash    4371 ubuntu  mem    REG    8,2   208328 4996850 /usr/lib/x86_64-linux-gn
u/libtinfo.so.6.4
bash    4371 ubuntu  mem    REG    8,2    27028 4983248 /usr/lib/x86_64-linux-gn
u/gconv/gconv-modules.cache
bash    4371 ubuntu  mem    REG    8,2   236616 4983256 /usr/lib/x86_64-linux-gn
u/ld-linux-x86-64.so.2
bash    4371 ubuntu    0w   CHR    1,3      0t0       5 /dev/null
bash    4371 ubuntu    1w   REG    8,2        0 2622153 /home/ubuntu/nohup.out
bash    4371 ubuntu    2w   REG    8,2        0 2622153 /home/ubuntu/nohup.out
```

`lsof` shows all open files and network connections associated with the process. In this case, it confirmed:

- The script was executing from the user's home directory.
- Output was being redirected into `nohup.out`.
- Network activity was occurring, though suppressed by `/dev/null`.

This step mirrors real incident response, where security analysts inspect processes to confirm malicious behavior before taking action.

4. Terminating the Malicious Connection

After verifying that the curl loop was unauthorized, the next step was to stop the activity by killing the processes:

```
ubuntu@ubuntu:~$ kill 4371 4417
```

Killing the running processes immediately stops the system from contacting the suspicious IP. Process termination is a standard containment step when dealing with malware or unknown scripts.

```
ubuntu@ubuntu:~$ ps aux | grep curl
ubuntu      4474  0.1  0.1  26348 10716 pts/0    S    21:53   0:00 curl http://4
5.13.220.98/ping
ubuntu      4482 33.3  0.0   9144  2232 pts/1    S+   21:53   0:00 grep --color=
auto curl
```

5. Blocking the External IP with a Firewall Rule

Stopping the process alone is not enough; malware may relaunch itself. To prevent future outbound contact with the malicious IP, firewall rules were created using **UFW (Uncomplicated Firewall)**:

```
ubuntu@ubuntu:~$ sudo ufw deny out to 45.13.220.98
sudo ufw deny from 45.13.220.98
Rule added
Rule added
```

This adds two protections:

1. **Outbound block** – prevents any application from reaching the malicious server.
2. **Inbound block** – stops the attacker from initiating connections back into the system.

Checking the firewall configuration confirmed that both rules were active:

```
ubuntu@ubuntu:~$ sudo ufw status numbered
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] 22/tcp                     ALLOW IN    Anywhere
[ 2] 45.13.220.98               DENY OUT    Anywhere                   (out)
[ 3] Anywhere                   DENY IN     45.13.220.98
[ 4] 22/tcp (v6)                ALLOW IN    Anywhere (v6)
```

Blocking at the firewall level is an important hardening measure and ensures persistent protection even if a malicious process respawns.

6. Validating the Block

To confirm the firewall rule was working, a manual curl request was attempted:

```
curl http://45.13.220.98/ping
```

The request hung and eventually had to be canceled with **Ctrl+C**, indicating the system could no longer reach the server—proof that the firewall block was effective.

7. Cleaning Up Artifacts

Finally, leftover files from the simulation were removed to clean the system:

```
ubuntu@ubuntu:~$ rm -f ~/nohup.out
ubuntu@ubuntu:~$
```

This ensures the lab environment is left in a clean state and prevents confusion in future investigations.