

## Step 4: Monitoring

### Libraries

Our script uses the python watchdog library to actively monitor our “critical” folder. Along with watchdog, we added the “auditd” command to our Linux machine to get more information about the processes used during this time.

### Auditd

“Auditd” is a good command to track fast processes like the “curl” command we are using in our ransomware script.

The following code runs the command “systemctl is-active --quiet auditd” to start the auditing process on Linux.

```
# Run auditd to catch the curl process
def run_auditd():
    try:
        subprocess.run(["systemctl", "is-active", "--quiet", "auditd"],
check=True)
    except subprocess.CalledProcessError:
        print("starting auditd")
        try:
            subprocess.run(["sudo", "systemctl", "start", "auditd"],
check=True)
            print(f"auditd started successfully")
        except subprocess.CalledProcessError:
            print(f"failed to start auditd")
```

The following code runs the command “auditctl -w “./critical” -p rwx -k “critical watch” to add the audit rule to the user’s Linux machine if they have not already. The audit is set to watch the “critical” folder with a key of “critical watch” to make filtering easier.

```
def add_audit(path_to_watch):
    try:
        subprocess.run(["auditctl", "-w", path_to_watch, "-p", "rwx",
"-k", AUDIT_KEY], check=True)
        print(f"rule added for {path_to_watch}")
    except subprocess.CalledProcessError:
```

```
print(f"audit rule already exists")
```

The following code runs the command “sudo ausearch -k “critical watch” -ts recent to parse the audit to find the process id that altered the files in the “./critical” directory.

```
def pid_from_audit():
    try:
        output = subprocess.check_output(["ausearch", "-k", AUDIT_KEY,
        "-ts", "recent"]).decode()
        for line in reversed(output.splitlines()):
            if "pid=" in line:
                parts = line.strip().split()
                for part in parts:
                    if part.startswith("pid="):
                        return int(part.split("=")[1])
    except subprocess.CalledProcessError:
        pass
    return None
```

## Logging

Our script logs the data to a .json file, so it can be easier to transfer the data to a database if needed. The timestamp, what was done to the files (add, moved, deleted, modified), file path, process that modified the file, and what action was taken are all logged to the .json file for further inspection. For now, the action will be logging until we reach the mitigation stage.

The following is the code for writing to the .json file. For now, it also writes to the command prompt for debugging.

```
# Write to a json log file
def log_event(event_type, file_path, pid):
    log_entry = {
        "timestamp": datetime.now().isoformat(),
        "event_type": event_type,
        "file": file_path,
        "pid": pid,
        "action_taken": "logged"
    }
    with open(LOG_FILE, "a") as f:
        f.write(json.dumps(log_entry) + "\n")
    print(f"[{log_entry['timestamp']}] {event_type.upper()} | File:
    {file_path} | pid: {pid}")
```

## Our Script

This is our python code for monitoring the “./critical” folder:

```
# Group 1
# Monitoring Assignment
# CSCE 5550

from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
import time
import psutil
import json
import os
import subprocess
from datetime import datetime

# setup variables, watchdog will be watching the "critical" folder and
write to monitoring_log.json
LOG_FILE = "monitoring_log.json"
WATCH_PATH = os.path.abspath("./critical")
AUDIT_KEY = "critical_watch"

# Run auditd to catch the curl process
def run_auditd():
    try:
        subprocess.run(["systemctl", "is-active", "--quiet", "auditd"],
check=True)
    except subprocess.CalledProcessError:
        print("starting auditd")
    try:
        subprocess.run(["sudo", "systemctl", "start", "auditd"],
check=True)
        print(f"auditd started successfully")
    except subprocess.CalledProcessError:
        print(f"failed to start auditd")

# Write to a json log file
def log_event(event_type, file_path, pid):
```

```
log_entry = {
    "timestamp": datetime.now().isoformat(),
    "event_type": event_type,
    "file": file_path,
    "pid": pid,
    "action_taken": "logged"
}

with open(LOG_FILE, "a") as f:
    f.write(json.dumps(log_entry) + "\n")
    print(f"[{log_entry['timestamp']}] {event_type.upper()} | File: {file_path} | pid: {pid}")

# Add the auditd rule to linux to find fast processes
def add_audit(path_to_watch):
    try:
        subprocess.run(["auditctl", "-w", path_to_watch, "-p", "rwx",
                        "-k", AUDIT_KEY], check=True)
        print(f"[AUDIT] Rule added for {path_to_watch}")
    except subprocess.CalledProcessError:
        print(f"Audit rule already exists")

# Find pid of the ransomware from the audit
def pid_from_audit():
    try:
        output = subprocess.check_output(["ausearch", "-k", AUDIT_KEY,
                                           "-ts", "recent"]).decode()
        for line in reversed(output.splitlines()):
            if "pid=" in line:
                parts = line.strip().split()
                for part in parts:
                    if part.startswith("pid="):
                        return int(part.split("=")[1])
    except subprocess.CalledProcessError:
        pass
    return None

# Separate the actions
class MonitorHandler(FileSystemEventHandler):
    def handle_event(self, event_type, event):
```

```
        pid = pid_from_audit()
        if pid:
            time.sleep(0.2)
        else:
            pid = "Unknown"

    log_event(event_type, event.src_path, pid)

def on_created(self, event):
    self.handle_event("created", event)

def on_modified(self, event):
    self.handle_event("modified", event)

def on_deleted(self, event):
    self.handle_event("deleted", event)

def on_moved(self, event):
    self.handle_event("moved", event)

# Continuously monitor the folder
if __name__ == "__main__":
    if not os.path.exists(WATCH_PATH):
        os.makedirs(WATCH_PATH)

    run_auditd()

    add_audit(WATCH_PATH)

    handler = MonitorHandler()
    observer = Observer()

    path_to_watch = "./critical"
    observer.schedule(handler, path=WATCH_PATH, recursive=True)
    observer.start()

    print(f"Monitoring started on: {WATCH_PATH}")
    print(f"Logging to: {os.path.abspath(LOG_FILE)}")
```

```
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    print(f"Monitoring stopped")
    observer.stop()

observer.join()
subprocess.run(["auditctl", "-W", WATCH_PATH, "-k", AUDIT_KEY])
```

The following is the bash script to compile and run the python code:

```
#!/bin/bash

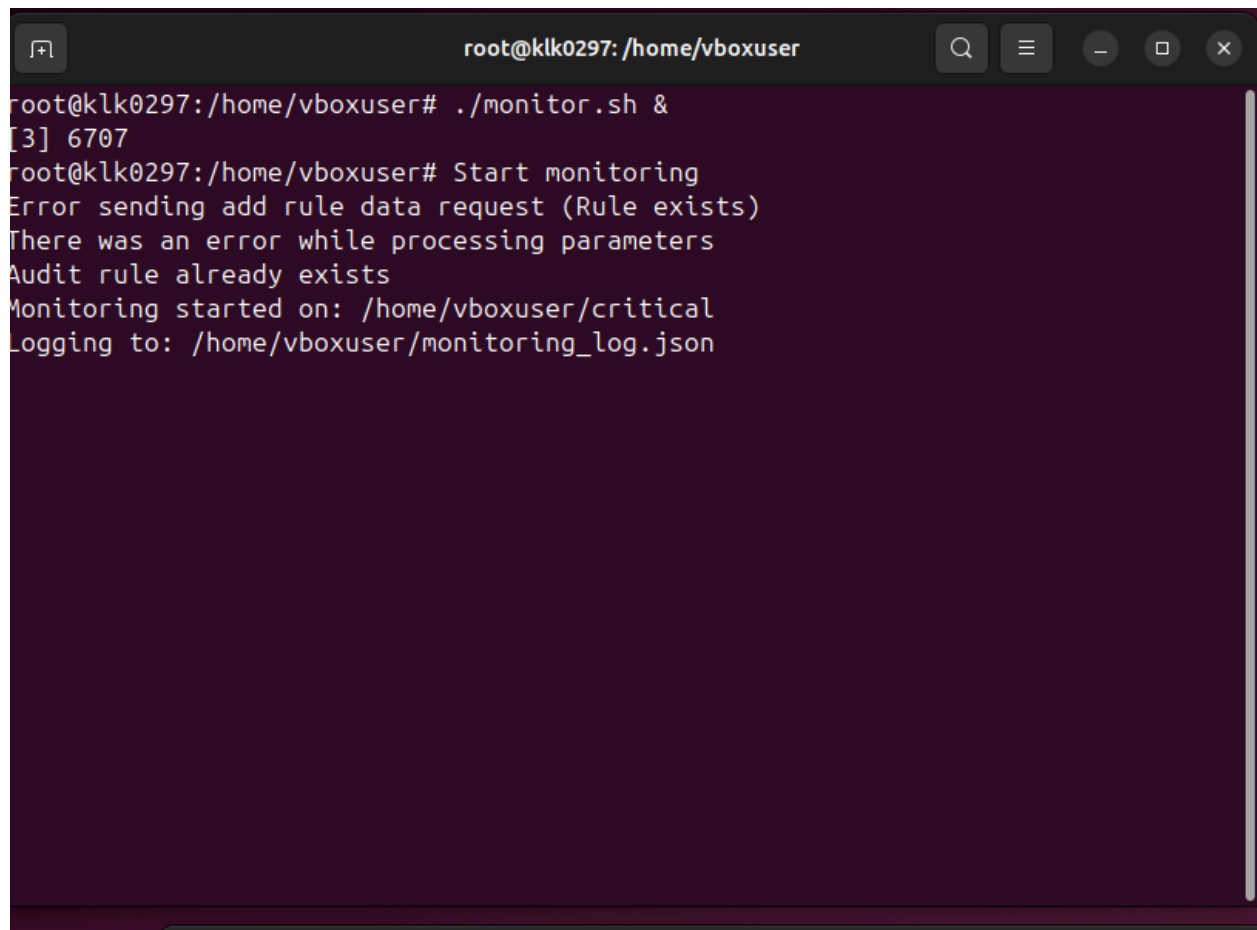
echo "Start monitoring"

python3 ./monitor.py
```

## Results

### Compiling

“./monitor.sh &” continuously runs the monitoring script without disrupting the current terminal from use. This should be used before running the encryption script. The following screenshot shows the result of running the command:

A terminal window with a dark purple background and white text. The window title bar shows 'root@klk0297: /home/vboxuser' and standard window controls. The terminal content shows a sequence of commands and their outputs. The first command is './monitor.sh &', which returns '[3] 6707'. The second command is 'Start monitoring', which produces three lines of error messages: 'Error sending add rule data request (Rule exists)', 'There was an error while processing parameters', and 'Audit rule already exists'. The final two lines of output are 'Monitoring started on: /home/vboxuser/critical' and 'Logging to: /home/vboxuser/monitoring\_log.json'.

```
root@klk0297: /home/vboxuser
root@klk0297:/home/vboxuser# ./monitor.sh &
[3] 6707
root@klk0297:/home/vboxuser# Start monitoring
Error sending add rule data request (Rule exists)
There was an error while processing parameters
Audit rule already exists
Monitoring started on: /home/vboxuser/critical
Logging to: /home/vboxuser/monitoring_log.json
```

## Results on Terminal

The screenshot shows a terminal window with three panes. The top pane displays a list of file operations: encryption of 't.Sniffing.pdf', modification of 'et.Sniffing.pdf', deletion of 't.Sniffing.pdf', creation of 'lls\_IDS.pdf', modification of 'lls\_IDS.pdf', deletion of 'lls\_IDS.pdf', and modification of 'lls\_IDS.pdf'. The bottom-left pane shows the execution of a script that encrypts files using a deprecated key derivation method. The bottom-right pane shows a C2 server running on port 8000, receiving a POST request from a victim machine.

```
root@klk0297:/home/vboxuser# t.Sniffing.pdf.encrypted | pid: 6660
[2025-04-07T20:08:48.466230] MODIFIED | File: /home/vboxuser/critical | pid: 666
0
[2025-04-07T20:08:48.679024] MODIFIED | File: /home/vboxuser/critical/Lab1b_Pack
et.Sniffing.pdf.encrypted | pid: 6660
[2025-04-07T20:08:48.901078] MODIFIED | File: /home/vboxuser/critical | pid: 666
0
[2025-04-07T20:08:49.117390] DELETED | File: /home/vboxuser/critical/Lab1b_Packe
t.Sniffing.pdf | pid: 6660
[2025-04-07T20:08:49.339173] MODIFIED | File: /home/vboxuser/critical | pid: 666
0
[2025-04-07T20:08:49.553284] CREATED | File: /home/vboxuser/critical/Lab2_Firewa
lls_IDS.pdf.encrypted | pid: 6660
[2025-04-07T20:08:49.777525] MODIFIED | File: /home/vboxuser/critical | pid: 666
0
[2025-04-07T20:08:50.008653] MODIFIED | File: /home/vboxuser/critical/Lab2_Firew
all_IDS.pdf.encrypted | pid: 6660
[2025-04-07T20:08:50.223284] MODIFIED | File: /home/vboxuser/critical | pid: 666
0
[2025-04-07T20:08:50.433840] DELETED | File: /home/vboxuser/critical/Lab2_Firewa
lls_IDS.pdf | pid: 6660
[2025-04-07T20:08:50.649993] MODIFIED | File: /home/vboxuser/critical | pid: 666
0

root@klk0297:/home/vboxuser# ./encrypt.sh
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
Encrypted: ./critical/Lab1a_Network_Scanning.pdf
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
Encrypted: ./critical/Lab1b_Packet_Sniffing.pdf
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
Encrypted: ./critical/Lab2_Firewalls_IDS.pdf
tee: /c/README_DECRYPT.txt: No such file or directory
Encryption complete. Password: 5b8ae14c0e72002641ebf2e4d854fa57edd6e27638876ec45
787834d080dd6de
root@klk0297:/home/vboxuser#

vboxuser@klk0297:~$ sudo su
[sudo] password for vboxuser:
root@klk0297:/home/vboxuser# sudo python3 c2_server.py
C2 server running on port 8000
^CTraceback (most recent call last):
  File "/home/vboxuser/c2_server.py", line 14, in <module>
    httpd.server_forever()
  File "/usr/lib/python3.12/socketserver.py", line 235, in serve_forever
    ready = selector.select(poll_interval)
             ~~~~~^~~~~~
  File "/usr/lib/python3.12/selectors.py", line 415, in select
    fd_event_list = self._selector.poll(timeout)
                    ~~~~~^~~~~~
KeyboardInterrupt

root@klk0297:/home/vboxuser# sudo python3 c2_server.py
C2 server running on port 8000
[+] Received POST to /log: victim=10.0.2.15&key=5b8ae14c0e72002641ebf2e4d854fa57
edd6e27638876ec45787834d080dd6de
10.0.2.15 - - [07/Apr/2025 20:08:47] "POST /log HTTP/1.1" 200 -
```

## .json log

The following screenshot shows the data saved in the .json file:

The screenshot shows a terminal window displaying the contents of a JSON log file named 'monitoring\_log.json'. The log contains 18 entries, each with a timestamp, event type, file path, and PID. The events include file creation, modification, deletion, and encryption.

```
GNU nano 7.2
monitoring_log.json

[{"timestamp": "2025-04-07T20:08:46.915292", "event_type": "created", "file": "/home/vboxuser/critical/Lab1a_Network_Scanning.pdf.encrypted", "pid": 6654, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:47.165542", "event_type": "modified", "file": "/home/vboxuser/critical", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:47.383674", "event_type": "modified", "file": "/home/vboxuser/critical/Lab1a_Network_Scanning.pdf.encrypted", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:47.599520", "event_type": "modified", "file": "/home/vboxuser/critical", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:47.830023", "event_type": "deleted", "file": "/home/vboxuser/critical/Lab1a_Network_Scanning.pdf", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:48.035745", "event_type": "modified", "file": "/home/vboxuser/critical", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:48.255560", "event_type": "created", "file": "/home/vboxuser/critical/Lab1b_Packet_Sniffing.pdf.encrypted", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:48.466230", "event_type": "modified", "file": "/home/vboxuser/critical", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:48.679024", "event_type": "modified", "file": "/home/vboxuser/critical/Lab1b_Packet_Sniffing.pdf.encrypted", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:48.901078", "event_type": "modified", "file": "/home/vboxuser/critical", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:49.117390", "event_type": "deleted", "file": "/home/vboxuser/critical/Lab1b_Packet_Sniffing.pdf", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:49.339173", "event_type": "modified", "file": "/home/vboxuser/critical", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:49.553284", "event_type": "created", "file": "/home/vboxuser/critical/Lab2_Firewalls_IDS.pdf.encrypted", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:49.777525", "event_type": "modified", "file": "/home/vboxuser/critical", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:50.008653", "event_type": "modified", "file": "/home/vboxuser/critical/Lab2_Firewalls_IDS.pdf.encrypted", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:50.223284", "event_type": "modified", "file": "/home/vboxuser/critical", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:50.433840", "event_type": "deleted", "file": "/home/vboxuser/critical/Lab2_Firewalls_IDS.pdf", "pid": 6660, "action_taken": "logged"},
{"timestamp": "2025-04-07T20:08:50.649993", "event_type": "modified", "file": "/home/vboxuser/critical", "pid": 6660, "action_taken": "logged"}]
```



## Next Steps

In the next assignment, we will set permission rules to detect the malicious ransomware code with the information we gathered from monitoring the system.