Kaylyn King, Tasmaiya Tamboli, Jean-Charles Hekamanu, Ujjwal Rana Magar
CSCE 5550
Spring 2025

Step 5: Detection

Methods

Honeypot

Our first method of detecting ransomware was to develop a honeypot. This file is created as a read only file and only used to see if it was accessed. In theory, everyone authorized to write to this “critical” folder will know not to touch the honeypot file, therefore anyone who does touch it is unauthorized.
The following code creates the honeypot in the “critical” directory with read only permissions:

```
# Create honeypot file to catch ransomware
def create_honeypot_file(path="critical/zzzzz_HONEYPOT.txt"):
    if not os.path.exists(path):
        with open(path, "w") as f:
            f.write("Gotcha")
        os.chmod(path, 0o444)
```

File extensions

Before the original files are deleted, a “.encrypted” file is formed within the “critical” directory. Therefore, we updated our previous monitoring code to add a note to the log if watchdog detects a “.encrypted” file has been created.
The following code is the updated log and handle_event functions:

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

    # Only prints to terminal if honeypot has been touched or suspicious
extension has been detected
    if "honeypot" in note.lower() or "suspicious" in note.lower():
```

```
        print(f"[{log_entry['timestamp']}] {event_type.upper()} | File:
{file_path} | pid: {pid} | note: {note}")
```

```
def handle_event(self, event_type, event):
        pid = pid_from_audit()
        time.sleep(0.2)

        # Added this part to discover suspicious extensions and to see if
the honeypot was accessed
        sus_extensions = [".encrypted", ".enc", ".lock"]
        _, ext = os.path.splitext(event.src_path)

        if "HONEYPOT" in os.path.basename(event.src_path):
            note = "Honeypot was accessed. Ransomware maybe detected"
        elif ext in sus_extensions:
            note = f"suspicious file: {ext}"
        else:
            note = f"Safe"
```

Our Script

```
# 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"
```

```
# Create honeypot file to catch ransomware
def create_honeypot_file(path="critical/zzzzz_HONEYPOT.txt"):
    if not os.path.exists(path):
        with open(path, "w") as f:
            f.write("Gotcha")
        os.chmod(path, 0o444)

# 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, note):
    log_entry = {
        "timestamp": datetime.now().isoformat(),
        "event_type": event_type,
        "file": file_path,
        "pid": pid,
        "note": note,
        "action_taken": "logged"
    }
    with open(LOG_FILE, "a") as f:
        f.write(json.dumps(log_entry) + "\n")

    # Only prints to terminal if honeypot has been touched or suspicious
extension has been detected
    if "honeypot" in note.lower() or "suspicious" in note.lower():
        print(f"[{log_entry['timestamp']}] {event_type.upper()} | File:
{file_path} | pid: {pid} | note: {note}")
```

```
# 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", "rwxa",
"-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 ransomeware 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()
        time.sleep(0.2)

        # Added this part to discover suspicious extensions and to see if
the honeypot was accessed
        sus_extensions = [".encrypted", ".enc", ".lock"]
        _, ext = os.path.splitext(event.src_path)

        if "HONEYPOT" in os.path.basename(event.src_path):
            note = "Honeypot was accessed. Ransomware maybe detected"
        elif ext in sus_extensions:
            note = f"suspicious file: {ext}"
        else:
```

```
            note = f"Safe"

        log_event(event_type, event.src_path, pid, note)

    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)

    create_honeypot_file()

    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])
```

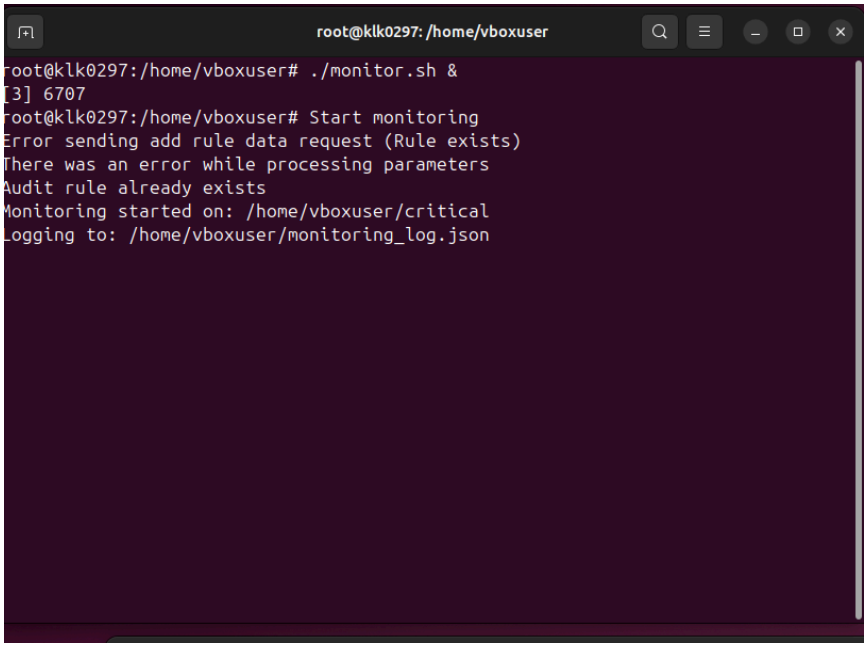
The following is the bash script to compile and run the python code:

```
#!/bin/bash

echo "Start monitoring"

python3 ./monitor.py
```

Results

Compiling

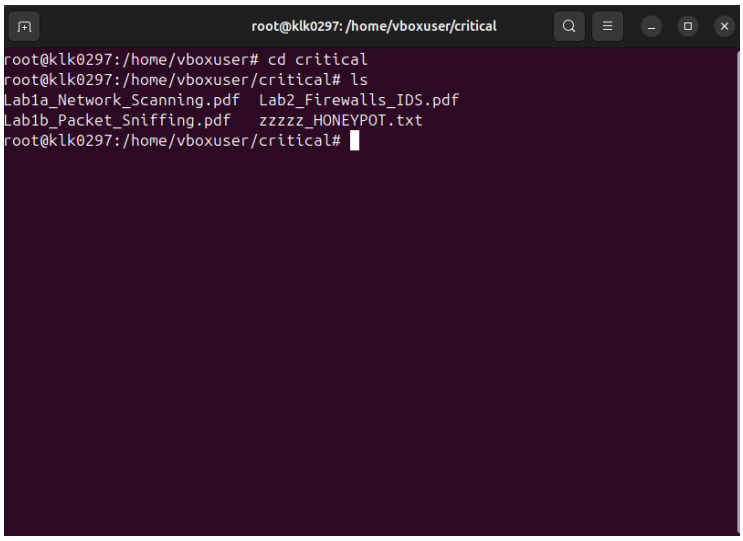
"./monitor.sh &" continuously runs the monitoring script without disrupting the current terminal from use. This should be used before running the encryption script. The following screenshot shows the result of running the command:



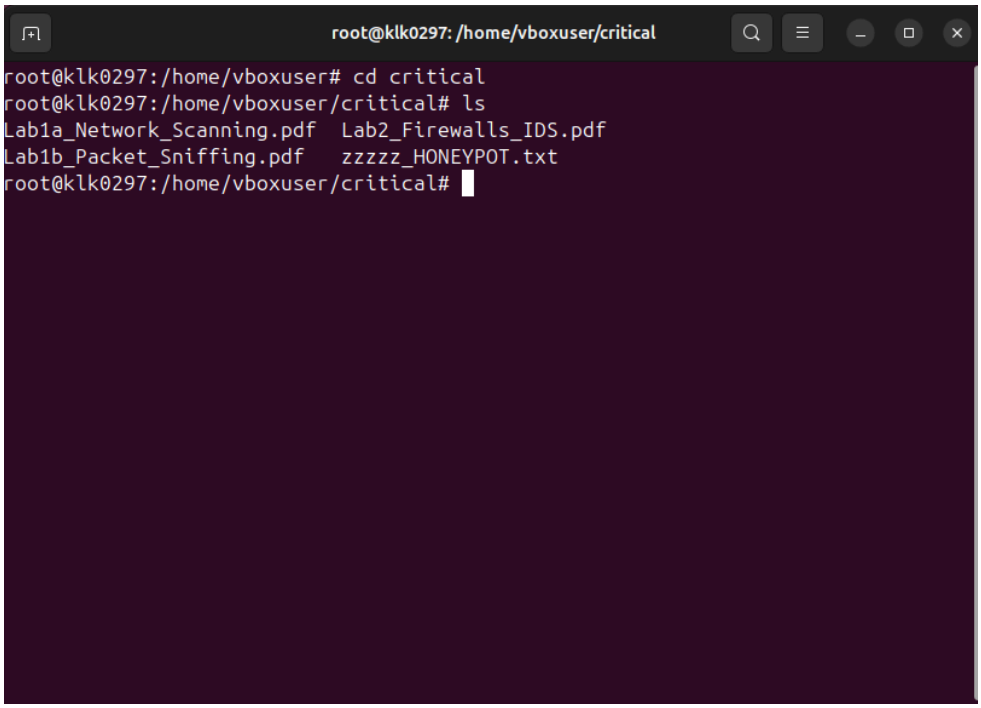
Kaylyn King, Tasmaiya Tamboli, Jean-Charles Hekamanu, Ujjwal Rana Magar
CSCE 5550
Spring 2025

Results After Encryption

For demonstration for this step, we have it setup to output the log information to the terminal when the policies are violated.

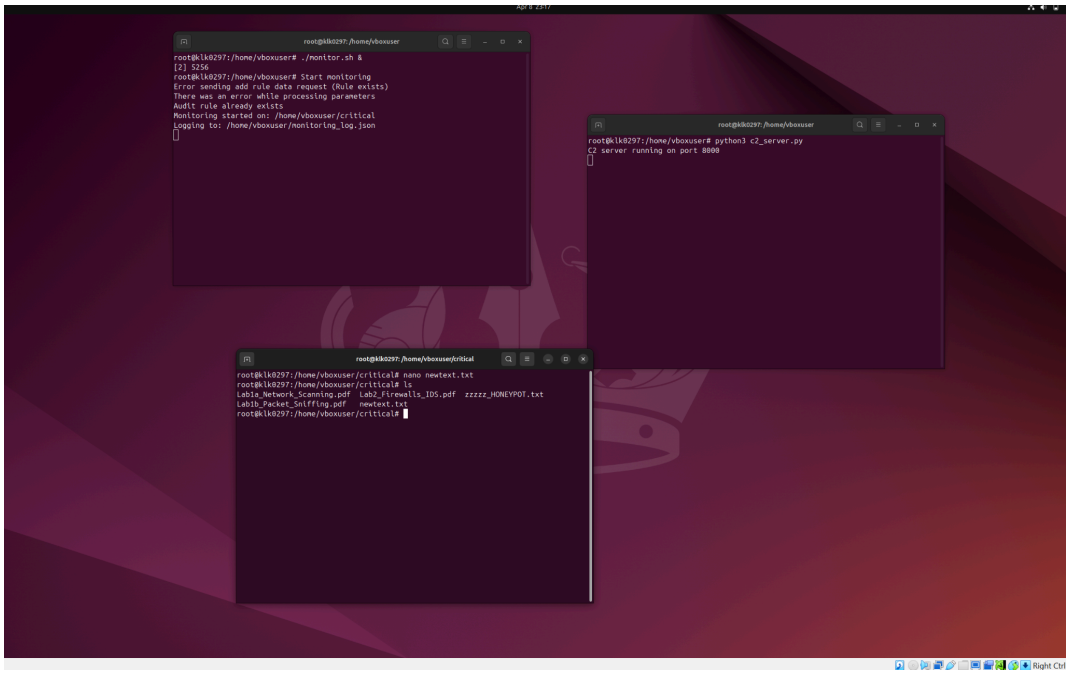


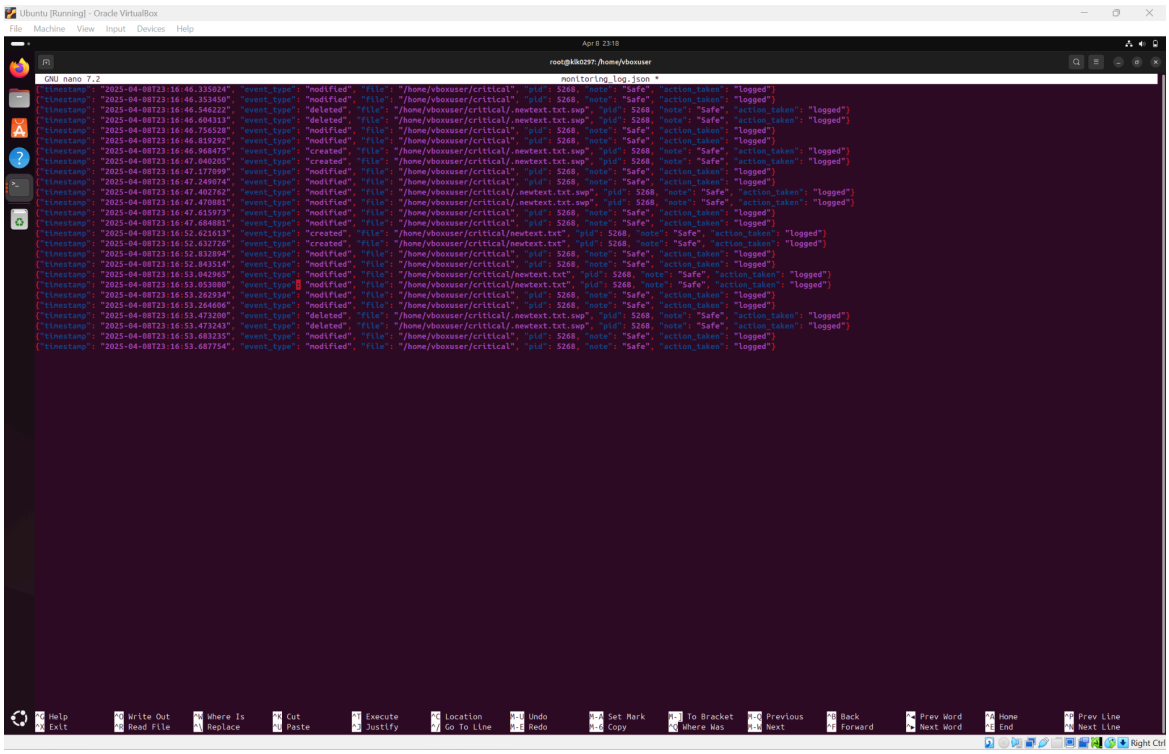
The following screenshot is of the honeypot being created within the directory after running our script:



The following screenshots of the result normal operations on the critical folder, and no notifications were outputting to the terminal while still logged in the .json file:

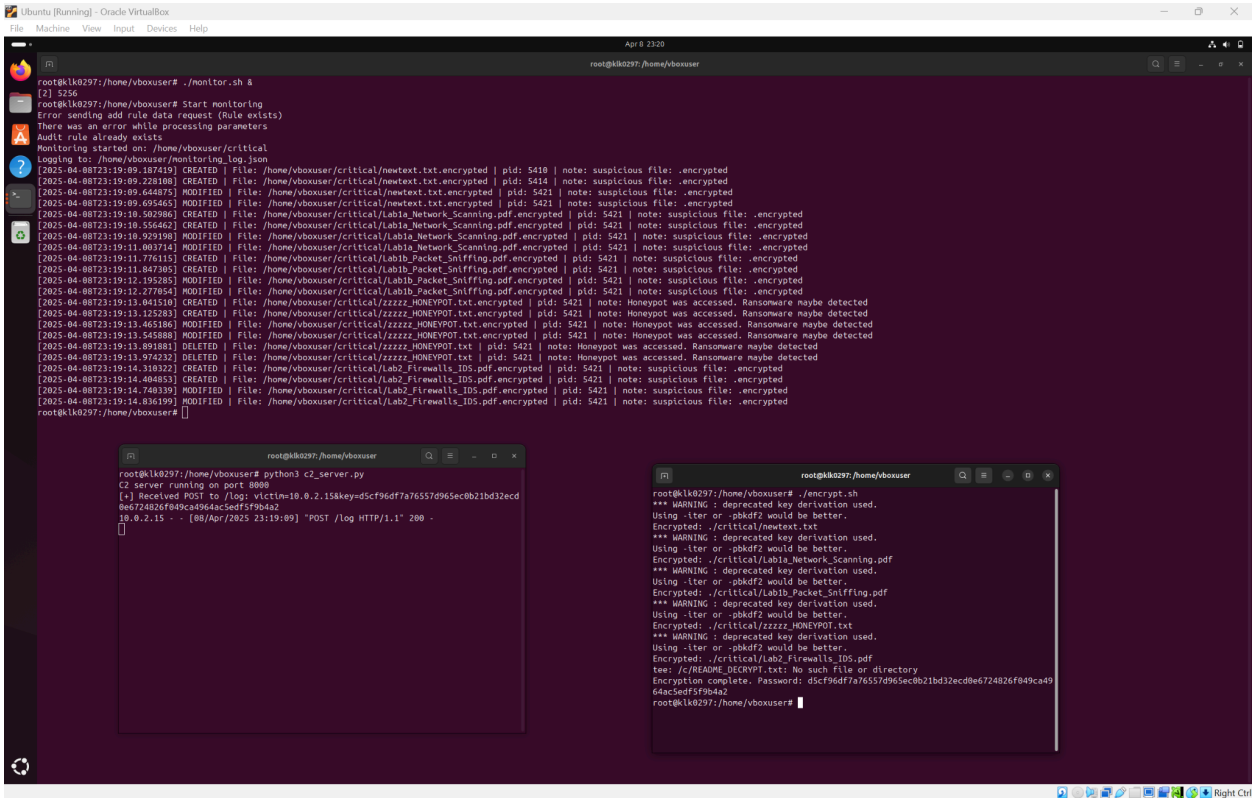
Kaylyn King, Tasmaiya Tamboli, Jean-Charles Hekamanu, Ujjwal Rana Magar
CSCE 5550
Spring 2025

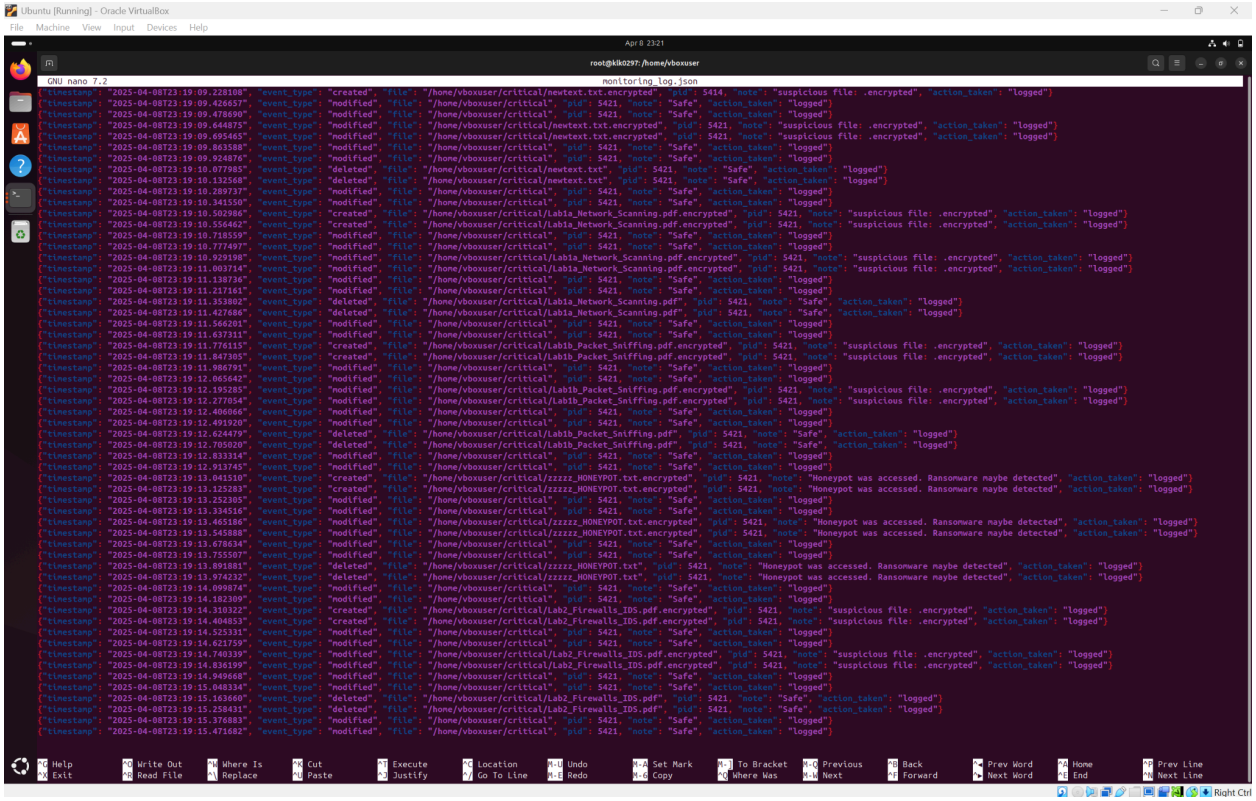




The following screenshots are of when malicious actions were spotted where the notifications are shown when the .encrypted files were created with the honeypot being accessed:

Kaylyn King, Tasmaiya Tamboli, Jean-Charles Hekamanu, Ujjwal Rana Magar
CSCE 5550
Spring 2025





Kaylyn King, Tasmaiya Tamboli, Jean-Charles Hekamanu, Ujjwal Rana Magar
CSCE 5550
Spring 2025

Next Steps

We will use these policies to notify the user that an attack is happening along with killing the process before it infects the rest of the folder.