PROJECT REPORT: Real-Time Active Defense system Against Ransomware

Student Name: Aakash Jha

Project Domain: Cybersecurity / Malware Analysis

Technology Stack: Python, Oracle VirtualBox, Watchdog, Psutil

1. Abstract

Ransomware is one of the fastest-growing cyber threats, capable of encrypting thousands of files in minutes. Traditional antivirus solutions often rely on "Signatures" (databases of known viruses), which fail to detect new, unknown threats (Zero-Day attacks). This project implements a **Behavioral** Active **Defense System.** Instead of looking for specific virus names, this tool monitors the file system for "abnormal behavior"—specifically, rapid file modifications. Upon detecting an attack, the system identifies the malicious process and automatically terminates it in real-time.

2. Problem Statement

- Latency: Human reaction time is too slow to stop ransomware once it starts.
- Signature Failure: Newransomware variants bypass traditional static detection.
- Need: A system that operates autonomously to detect high-frequency file changes and neutralize the threat immediately.

3. l"1ethodofogy

3.1. Infrastructure Setup (The Sandbox)

To safely develop and test the malware simulation, a "Sandbox" environment was created:

- **Virtualization:** Oracle VirtualBox running Windows 10.
- **Isolation:** Network adapters disabled; Shared Folders disabled to prevent host infection.
- Dependencies: Python 3.x installed with watchdog (monitoring) and psutil (process management) libraries.

3.2. Logic Flow

The system operates on a "Detect -> Hunt -> Kill" cycle:

1. **I"lonitor:** Watch the file system for Modified events

2. Analyze: If modification frequency exceeds the threshold (3 files in 100 seconds), trigger an ALERT.
3. Hunt: Scan running processes to identify which Process ID (PID) is currently holding the target file open.
4. Remediate: Check the PID against a "Whitelist." If it is not a safe app, terminate the process immediately.

4. Implementation Details (Code)

4.1. The Guard Script (ReafTimeGuard.py)

This script acts as the "Antivirus." It runs in the background and enforces the security policy.

Python--

import time

import sys

import os

import psutil

from watchdog.observers import Observer

from watchdog.events import FileSystemEventHandler

——— CONFIGURATION

——— DIRECTORY_TO_WATCH =

"" THRESHOLD = 3

TIf1E WINDOW = 100

——— WHITELIST (Safe Programs) ———

WHITELIST = ["explorer.exe", "notepad.exe", "svchost.exe"]

class ActiveDefenseHandler(FileSystemEventHandler): def

finite(self):

```
    self.modification_count = 0
    self.start time = time.time()

def find_and_kill_process(self, file_path):
    '''Scans running processes to find who is holding the file open." print(f" @
    Hunting for process touching: (file path}")

    for proc in psutil.process_iter(['pid', name', 'open_files']):
        try:
            if proc.info['open files']:
                for open_Tile in proc.info['open_files']:
                    if open_file.path == os.path.abspath(file_path):

                        proc_name = proc.info['name']
                        proc_pid - proc.inTo['pid']

                        if proc_name in WHITELIST:
                            print(f" 9  Safe process detected ({proc name}). Skipping.")
                            return

                        print(f"                                                    ")
                        print(f" >< MALWARE DETECTED: (proc_name) (PID: (proc_pid))") print(f"
                        @ TER IINATING PROCESS IMIfIEDIATELY. ")
                        proc.kill()
                        print(f" @ Threat Neutralized.")
                        print(f"                                                    ")
                        return
```

```
            except (psutil.NoSuchProcess, psutil.AccessDenied): pass

        def on_modified(self, event):
         if event.is_directory: return

       current_time = time.time()
       if current_time - self.start_time > TIME_WINDOW:
          self.modification_count = 0
          self.start time = current time

       self.modification_count += 1
       print(f"[INFO] File modified: (event.src_path}")

       if self.modification count >= THRESHOLD:
          print("!!! ALERT: High frequency modifications detected !!!")
          self.find_and_kifl_process(event.src_path)

if     name      == "  main  "
    print(f" 4  ACTIVE DEFENSE SYSTEM ONLINE")
    event_handler  = ActiveDefenseHandler()
    observer = Observer()
    observer.schedule(event handler, path=DIRECTORY TO WATCH, recursive=True) observer.start()

   try:
      while True: time.sleep(1)
```

except KeyboardInterrupt: observer.stop()

observer.join()

4.2. The Attacker Script (attacker.py)

This script simulates a ransomware attack. It writes malicious data to lifes and forces a system save to trigger the Guard.

Python--

```
import as

import time


TARGET FOLDER = "TestFolder"


def encrypt_fifes():

  print(f" @ MALWARE STARTED. Targeting: {TARGET_FOLDER}")


  if not os.path.exists(TARGET_FOLDER):

    print("Target folder not found!")

    return


  files = os.listdir(TARGET_FOLDER)


  for file_name in files:

    file_path = os.path.join(TARGET_FOLDER, file_name)


    if os.path.isfile(file_path):

      print(f"Encrypting(file_name)...")


      with open(file_path, "a") as f:

        f.write("\n ENCRYPTED_CHUNK_1 ")
```

```
            # Force Windows to update the file timestamp immediately
            f.flush()
            os.fsync(f.fileno())

            # Wait to allow the Guard to detect the open file handle
            print(" (WAITING 10s - CATCH ME NOW!)")
            time.sleep(10.0)

        time.sleep(0.1 )

    print(" @ ATTACK FINISHED.")

if    name    == "  main  "
    choice = input("Type 'yes’ to start the attack:
    ") if choice.lower() == "yes’".
        encrypt_files()
```

---

5. Testing and Results

5.1. Test Strategy

To simulate a real-world scenario without endangering the host machine, the test was conducted inside the isolated VirtualBox environment. Two separate Python IDLE instances were used: one to run the Guard and one to run the Attacker.

5.2. Test Scenario (TC-001)

- **Condition:** RealTimeGuard.py is active. attacker.py attempts to encrypt files in TestFolder.
- **Behavior:** The Attacker modified file1.txt, file2.txt, and file3.txt.
- **Detection:** On the 3rd modification, the Guard triggered the High Frequency Alert.

- **Response:** The Guard scanned for the process ID (PID) holding file3.txt, identified the Python process running the attack, and issued a SIGKILL (Force Kill) command.

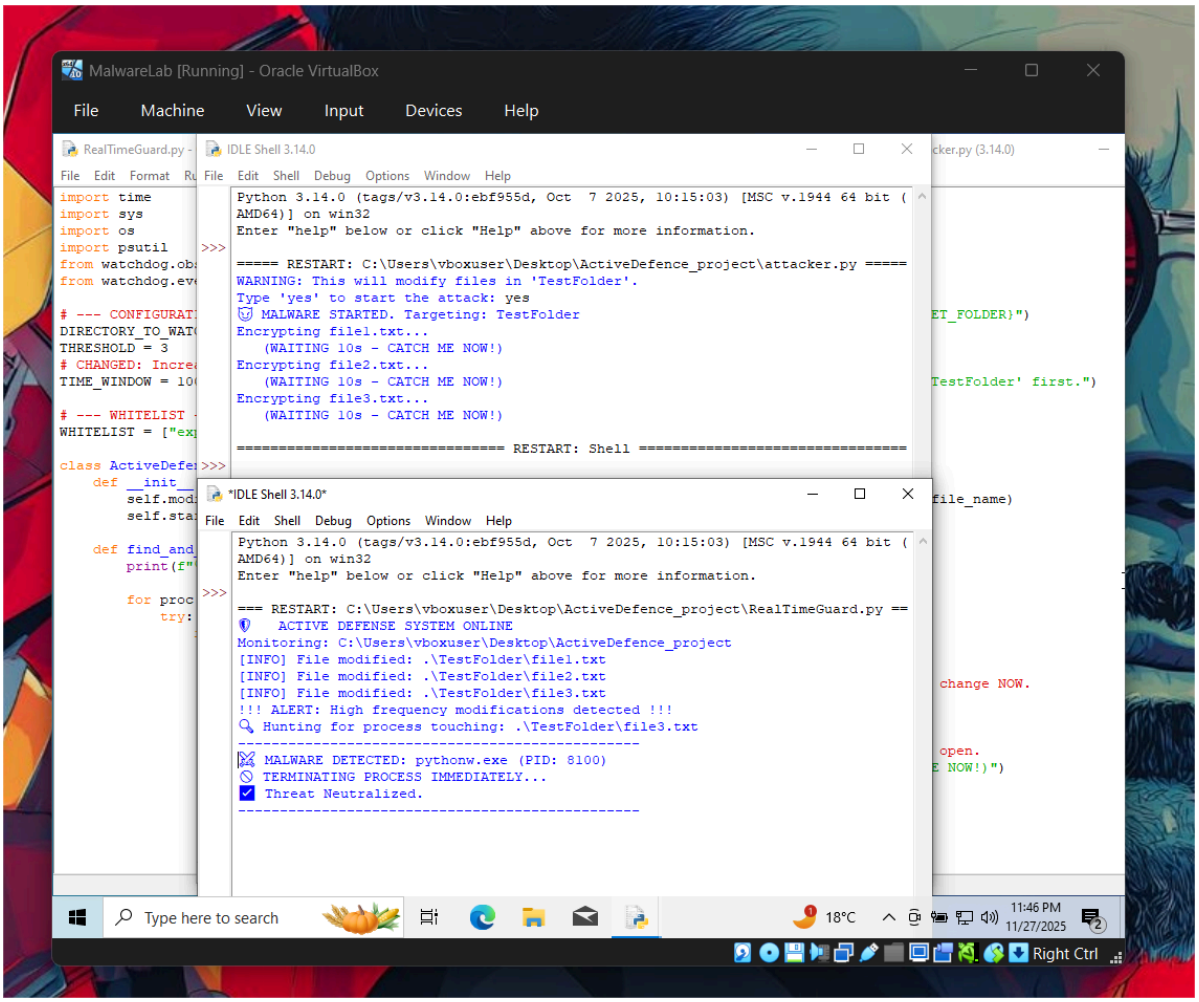
5.3. Results

The Attacker process was successfully terminated before it could complete its operation.
The system logs displayed:

MALWARE DETECTED: pythons.exe (PID: 4664) @

TERMINATING PROCESS IMMEDIATELY...

@ Threat Neutralized.



6. Conclusion

This project successfully demonstrated the effectiveness of **Heuristic/Behavioral Analysis** in cybersecurity. By monitoring the *rate of* file changes rather than the file *contents,* the system was able to detect and neutralize a simulated ransomware attack purely based on its aggressive behavior. This approach serves as a powerful layer of defense against Zero-Day threats that traditional antivirus scanners might miss.

Why can't we run both in one window?

There are two technical reasons why you must use separate windows. You can include

this in your "Challenges Faced™ section of the report.

1. **The "Sheff Restart" Feature:**
    a Python IDLE is designed for testing. Every time you press FS (Run), IDLE performs a "Shell Restart."
    c This wipes the memory clean to prevent old code from messing up new code.

    Result: If you run the Guard, then run the Attacker in the same window, IDLE automatically kills theGuard before starting the Attacker.

2. **The "Suicide" Risk (Process Identity):**

    Your Guard script works by finding the **Process ID (PID)** oT the attacker and killing it.

    o If you somehow managed to run both scripts in the same window, they would be part of the **same program** (sharing the same PID).

- Result: When the Guard tries to kill the Attacker, it would see that the Attacker is actually itself. It would issue the kill command and instantly crash the entire system. Keeping them in separate windows gives them separate PIDs (e.g., PID 4056 and PID 5022), allowing the Guard to kill one without dying itself.
