Information Security
Assignment # 1

Team Members:

Areej Dar - 200901008

Farwa Rizvi - 200901098

Mahnoor Sadiq - 200901005

Contents

1.1: Introduction:

Ransomware attacks have become more common in recent years, posing a substantial threat to both enterprises and people. These harmful programs encrypt files and demand money for their decryption, frequently inflicting significant damage and financial loss. To reduce such risks, it is critical to develop and install strong systems capable of detecting and protecting against ransomware assaults.

1.2: Methodology:

1.2.1: File Monitoring and Guarding:

- Maintain a current list of files to monitor and guard, along with their modification dates.
- Implement a file-similarity function, such as entropy monitoring, to identify changes in file attributes that indicate ransomware activity.
- To discover possible risks, monitor file alterations continuously, and evaluate patterns.

1.2.2: Resource Monitoring:

- Use tools like WMI (Windows Management Instrumentation) to monitor CPU and hard disk consumption in real-time.
- Set CPU and disk utilization thresholds to identify any anomalous behavior that might indicate a ransomware assault.

1.2.3: Ransomware Detection and Safeguarding:

- To identify suspected ransomware activity, look for indications such as repeated file alterations in a short period and increasing file entropy.
- When compromised files are detected, add a.tmp extension and transfer them to a honeypot folder to avoid further encryption.
- Implement procedures to prevent ransomware attacks and safeguard vital data from encryption.

1.2.4: Alerting and Response:

- Create systems to warn users of possible ransomware activity, giving timely notifications to trigger response actions.
- Implement a restart plan to restore system integrity and limit the harm caused by ransomware.

1.3: Summary of System Features:

The system has full file monitoring and guarding capabilities, which ensure file integrity by actively monitoring alterations and preserving updated file hashes. It uses resource monitoring techniques to continually check CPU and disk consumption, alerting users to any aberrant patterns that indicate ransomware activity. When ransomware assaults are detected, the system takes preemptive actions such as adding .tmp extensions to affected files and transferring them to a protected honey pot folder, significantly minimizing their impact and protecting crucial data.

1.4: Potential Improvements:

- The use of sophisticated anomaly detection techniques to improve the accuracy of ransomware detection.
- Machine learning algorithms are being integrated to dynamically alter detection levels in response to historical data and emerging ransomware methods.
- Real-time behavioral analysis is used to detect ransomware-like behavior patterns, such as quick file encryption.
- Create a centralized dashboard for full monitoring and administration of different systems inside a business.
- Integration of automating incident response techniques to streamline the mitigation process and reduce reaction time.

1.5: Code:

```
import os
import time
import threading
import shutil
import logging
import hashlib
import psutil
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
from collections import Counter
from scipy.stats import entropy
```

```
# Setup logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s -
%(message)s')

file_hashes = {}  # Global dictionary to store file hashes and entropy values
recent_modifications = []  # Track timestamps of recent file modifications

def calculate_hash(filepath):
    """Calculate the SHA-256 hash of the file."""
    sha256_hash = hashlib.sha256()
    with open(filepath, "rb") as f:
        for byte_block in iter(lambda: f.read(4096), b""):
            sha256_hash.update(byte_block)
    return sha256_hash.hexdigest()

def calculate_entropy(file_path):
    """Calculate the entropy of the file using the byte frequency."""
    with open(file_path, 'rb') as f:
        data = f.read()
    _, counts = zip(*Counter(data).items())
    return entropy(counts)

def get_system_usage():
    """Retrieve and return current system CPU and disk usage."""
    cpu_usage = psutil.cpu_percent(interval=1)
    disk_usage = psutil.disk_usage('/').percent
    return cpu_usage, disk_usage

class MonitorHandler(FileSystemEventHandler):
    def __init__(self, backup_dir, monitor_dir):
        self.backup_dir = backup_dir
        self.monitor_dir = monitor_dir
        self.ensure_directory(self.backup_dir)
        self.ensure_directory(self.monitor_dir)
        self.update_file_hashes()

    def ensure_directory(self, path):
        """Ensure directory exists."""
        if not os.path.exists(path):
            os.makedirs(path, exist_ok=True)

    def update_file_hashes(self):
        """Update file hashes and entropy values for all files in the monitored
directory."""
```

```
        for dirpath, dirnames, filenames in os.walk(self.monitor_dir):
            for filename in filenames:
                filepath = os.path.join(dirpath, filename)
                if not filepath.startswith(self.backup_dir) and not
filepath.endswith(('~$')):
                    file_hashes[filepath] = {
                        'hash': calculate_hash(filepath),
                        'entropy': calculate_entropy(filepath)
                    }

    def on_modified(self, event):
        """Handle file modification events."""
        if not event.is_directory:
            self.handle_event(event)

    def handle_event(self, event):
        """Handle individual file modification by checking hash and entropy
changes."""
        path = event.src_path
        if os.path.isfile(path) and not path.startswith(self.backup_dir) and not
path.endswith(('~$')):
            recent_modifications.append(time.time())
            cpu_before, disk_before = get_system_usage()
            old_data = file_hashes.get(path, {})
            logging.info(f"Before modification - CPU: {cpu_before}%, Disk:
{disk_before}%, File entropy: {old_data.get('entropy', 'N/A')}")

            new_hash = calculate_hash(path)
            new_entropy = calculate_entropy(path)
            if new_hash != old_data.get('hash') or abs(new_entropy -
old_data.get('entropy', 0)) > 0.1:
                try:
                    os.chmod(path, 0o444)  # Lock the file
                    new_path = os.path.join(self.backup_dir,
os.path.basename(path) + ".tmp")
                    if not os.path.exists(new_path):
                        shutil.move(path, new_path)  # Move file to backup
                        logging.info(f"Moved to backup and marked as .tmp and
read-only due to suspected tampering: {new_path}")
                    else:
                        logging.info(f"File already backed up: {new_path}")
                except Exception as e:
                    logging.error(f"Failed to handle file {path} due to:
{str(e)}")
                cpu_after, disk_after = get_system_usage()
```

```
                logging.info(f"After modification - CPU: {cpu_after}%, Disk:
{disk_after}%, New File entropy: {new_entropy}")

def monitor_high_resource_usage():
    """Monitor high resource usage, which might indicate ransomware activity."""
    while True:
        cpu_percent, disk_percent = get_system_usage()
        current_time = time.time()
        recent_mod_count = sum(1 for mod_time in recent_modifications if
current_time - mod_time < 30)
        if len(recent_modifications) >= 2 and cpu_percent > 80 and disk_percent >
98:
            logging.warning(f"High CPU ({cpu_percent}%) and disk usage
({disk_percent}%) detected with {recent_mod_count} recent file modifications.
System will restart shortly.")
            os.system("shutdown /r /t 1")  # enable system restart
        time.sleep(10)

def main():
    base_path = r"monitor"
    backup_path = r"backup\Recycler\a\b\c\d\e\f\g\h"
    os.makedirs(base_path, exist_ok=True)
    os.makedirs(backup_path, exist_ok=True)

    event_handler = MonitorHandler(backup_path, base_path)
    observer = Observer()
    observer.schedule(event_handler, base_path, recursive=True)
    observer.start()

    monitor_thread = threading.Thread(target=monitor_high_resource_usage)
    monitor_thread.start()

    try:
        while True:
            logging.info("Monitoring active...")
            time.sleep(60)  # Logs every 60 seconds
    except KeyboardInterrupt:
        observer.stop()
        observer.join()
        monitor_thread.join()
        logging.info("Monitoring stopped by user.")

if __name__ == "__main__":
    main()
```

1.6: Code Documentation:

The Python script given builds a system for detecting and protecting against ransomware assaults by monitoring file changes, CPU and disk consumption, and looking for any abnormalities that indicate ransomware activity.

The 'calculate_hash' function computes a file's SHA-256 hash to keep track of its integrity. Similarly, the 'calculate_entropy' function calculates a file's entropy to identify changes caused by compression or encryption.

The 'MonitorHandler' class is an extension of the 'FileSystemEventHandler' class from the 'watchdog' library that handles file modification events. During setup, it updates the file hashes and entropy values for all files in the monitored directory, and when a file is modified, it compares the updated hash and entropy values to the previous ones.

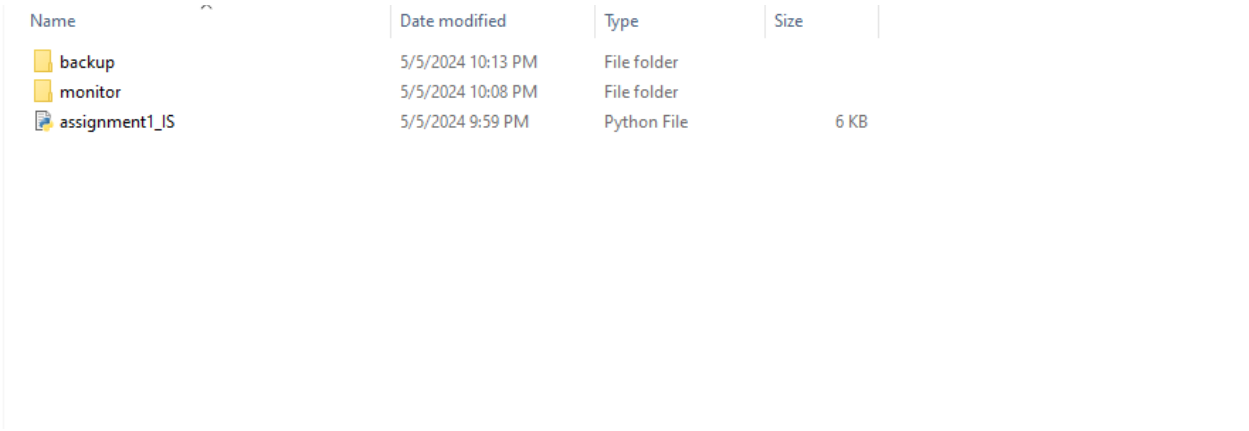
The script makes the file read-only and moves it to a secure backup directory to protect it in case it detects a major change in hash or entropy. For monitoring purposes, this process is logged together with the CPU and disk consumption before and following the adjustment.

Furthermore, real-time CPU and disk consumption monitoring is done continually via the monitor_high_resource_usage function. The system starts a system restart to lessen the threat if it detects excessive resource utilization along with several recent file alterations, which are signs of possible ransomware activity.

By establishing event handlers, initiating resource monitoring threads and observers, and building the required directories, the main method initializes the monitoring system. Additionally, it manages keyboard interruptions to end the monitoring session politely.

1.7: Output:

```
PS H:\IS> & C:/Users/Areej/AppData/Local/Microsoft/WindowsApps/python3.11.exe h:/IS/assignment1_IS.py
2024-05-05 22:13:30,482 - INFO - Monitoring active...
2024-05-05 22:13:53,823 - INFO - Before modification - CPU: 14.2%, Disk: 0.1%, File entropy: 4.158875836486422
2024-05-05 22:13:54,083 - INFO - Moved to backup and marked as .tmp and read-only due to suspected tampering: backup\Recyc
ler\a\b\c\d\e\f\g\h\Band theory.pdf.tmp
2024-05-05 22:13:55,084 - INFO - After modification - CPU: 45.8%, Disk: 0.1%, New File entropy: 4.158875839299533
2024-05-05 22:14:30,484 - INFO - Monitoring active...
2024-05-05 22:15:04,702 - INFO - Before modification - CPU: 51.1%, Disk: 0.1%, File entropy: 4.1574622390376295
2024-05-05 22:15:04,823 - INFO - Moved to backup and marked as .tmp and read-only due to suspected tampering: backup\Recyc
ler\a\b\c\d\e\f\g\h\bob.xlsx.tmp
2024-05-05 22:15:05,825 - INFO - After modification - CPU: 38.8%, Disk: 0.1%, New File entropy: 4.1577863447842365
2024-05-05 22:15:30,486 - INFO - Monitoring active...
```



1.8: Conclusion:

To summarize, the developed system provides an effective defensive mechanism against ransomware assaults by constantly monitoring file alterations and resource consumption, recognizing possible threats, and taking proactive actions to protect crucial data. The solution improves organizational and individual resilience to ransomware attacks by employing techniques such as file integrity checks, entropy analysis, and process monitoring.

However, ongoing research and improvement are required to remain abreast of changing ransomware techniques and future threats. By merging powerful detection algorithms, machine learning models, and real-time behavioral analysis, the system may improve its capabilities and provide strong protection against ransomware assaults in an ever-changing threat scenario.