Windows Service & Process Monitoring Agent

is a project submitted in partial fulfillment of the requirements for the

Internship Role: Cyber Security Intern

Organization: Unified Mentor

Prepared By: Tanisha Sagar

Project Type: Defensive Cyber Security / Blue Team Project

Duration of Internship: 3 months

INDEX

INTRODUCTION

Windows operating systems rely heavily on processes and services to run applications and system components. Every background task, user application, or system function operates through a process or service. Because of this, attackers and malware often abuse Windows processes and services to execute malicious code, maintain persistence, or operate without user awareness.

Malware commonly disguises itself as legitimate processes, exploits abnormal parent–child process relationships, or registers itself as a startup service to ensure automatic execution after system reboot. Detecting such behavior requires continuous visibility into process activity and service configurations, which is not practical through manual monitoring.

This project focuses on building a **Windows Service & Process Monitoring Agent** that automatically monitors running processes and Windows services to identify suspicious or unauthorized activity. The system uses rule-based detection, severity classification, and logging to highlight behaviors that may indicate security risks. The project follows a defensive cybersecurity approach and is intended for learning, monitoring, and analysis purposes only.

PROJECT OVERVIEW

This project involves the development of a **Windows Service & Process Monitoring Agent** designed to monitor system activity and identify suspicious behavior related to running processes and Windows services. The system continuously observes process execution, parent–child relationships, service configurations, and execution paths to detect patterns commonly associated with malicious activity.

The monitoring agent is implemented using Python and follows a **rule-based detection approach**. It collects detailed information such as process IDs, parent process IDs, executable paths, and service status, and then evaluates this data against predefined detection rules. Activities running from user-writable directories, abnormal execution chains, or misconfigured services are flagged and classified based on severity.

To improve accuracy and reduce false positives, the system includes a **whitelist mechanism** that allows verified and trusted applications to be treated differently during analysis. All findings are logged with timestamps, and a structured report is generated to summarize detected anomalies and overall system behavior.

The project is designed as a **defensive security tool** that simulates the basic functionality of SOC monitoring systems. It does not attempt to remove malware or modify system settings, but instead focuses on providing visibility, detection, and clear reporting of potential security risks.

PROBLEM STATEMENT

Modern malware often avoids traditional detection methods by abusing legitimate Windows processes and services. Instead of dropping obvious malicious files, attackers use trusted system components, abnormal parent–child process relationships, and startup services to execute malicious code and maintain persistence on a system.

Manual identification of such activities is difficult because Windows systems run hundreds of processes and services simultaneously. Suspicious behavior can easily go unnoticed, especially when malware disguises itself using legitimate names or locations.

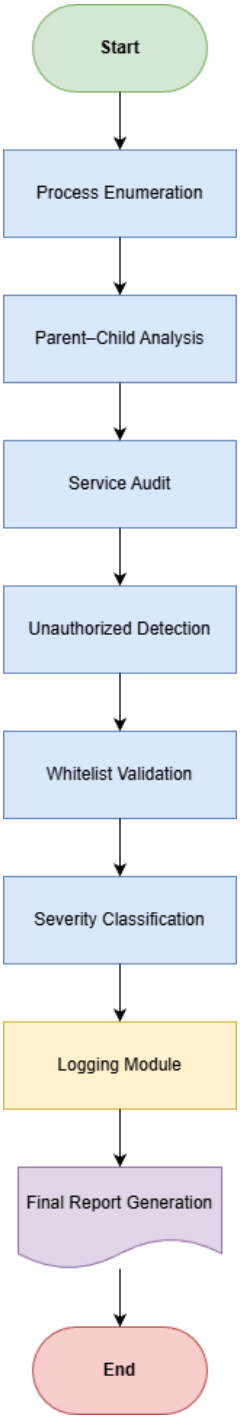
There is a need for a lightweight and explainable monitoring solution that can continuously observe process and service activity, identify abnormal patterns, and provide clear visibility into potential security risks. This project addresses that need by implementing a process and service monitoring system that highlights suspicious behavior using rule-based detection and structured reporting.

PROJECT OBJECTIVES

The main objectives of this project are:

- To monitor active Windows processes and collect relevant process information such as process name, PID, parent PID, and executable path
- To analyze parent–child process relationships and identify abnormal execution patterns
- To audit Windows startup services and detect suspicious or misconfigured service entries
- To detect unauthorized or suspicious processes and services based on execution location and behavior
- To reduce false positives using a whitelist-based validation approach
- To generate timestamped logs and a structured report summarizing all detected findings

SYSTEM ARCHITECTURE



*Flowchart 1: System Architecture of the Windows Service & Process Monitoring Agen*t

The diagram illustrates the overall workflow of the monitoring agent, starting from process enumeration and moving through parent–child process analysis, Windows service auditing, unauthorized activity detection, and whitelist validation. The system applies severity

classification to detected events and generates structured logs and a final report to provide visibility into suspicious process and service behavior.

SCOPE OF THE PROJECT

The scope of this project is limited to monitoring and analyzing Windows processes and services for suspicious or unauthorized behavior. The system focuses on detecting abnormal process execution patterns, unauthorized execution locations, and misconfigured or suspicious Windows services that may indicate potential security risks.

The project includes:

- Monitoring active Windows processes and their execution paths
- Analyzing parent–child process relationships
- Auditing Windows startup services and service configurations
- Detecting unauthorized or suspicious processes and services
- Logging detected events and generating a structured report

The project does not include:

- Active malware removal or system remediation
- Kernel-level or driver-based monitoring
- Real-time blocking or prevention of malicious activity

This system is intended for **learning, monitoring, and defensive analysis purposes only**.

TOOLS AND ENVIRONMENT USED

The following tools and environment were used for the development and testing of this project:

- **Python 3** – Used as the primary programming language to implement process monitoring, service auditing, detection logic, and reporting
- **psutil Library** – Used for enumerating running processes and collecting process-related information such as PID, parent PID, and executable paths
- **PowerShell (WMI)** – Used to query Windows service information including service name, status, and executable path
- **Windows 10 / Windows 11 (64-bit)** – Used as the testing environment for executing and validating the monitoring agent
- **Visual Studio Code** – Used as the main development environment for writing, testing, and debugging Python scripts
- **draw.io (diagrams.net)** – Used for designing the system architecture diagram

- **Command Prompt / PowerShell** – Used for running scripts and observing monitoring output

METHODOLOGY

The project follows a structured and sequential methodology to monitor Windows processes and services and identify suspicious behavior. Each stage builds on the output of the previous step to ensure clear detection and reporting.

The methodology followed is as outlined below:

1. **Process Enumeration**
   All active Windows processes are enumerated using Python. For each process, key details such as process name, PID, parent PID, and executable path are collected.
2. **Parent–Child Process Analysis**
   The relationship between parent and child processes is analyzed to identify abnormal execution patterns that may indicate malicious behavior.
3. **Windows Service Audit**
   Windows services are enumerated using PowerShell (WMI). Service name, status, and executable path are collected to detect suspicious or misconfigured services.
4. **Unauthorized Detection**
   Processes and services running from user-writable directories such as AppData and Temp are identified, as these locations are commonly abused by malware.
5. **Whitelist Validation**
   A whitelist mechanism is applied to reduce false positives. Verified applications are classified with lower severity after manual validation.
6. **Severity Classification**
   Detected events are categorized into LOW, MEDIUM, and HIGH severity levels based on their risk and behavior.
7. **Logging and Reporting**
   All findings are logged with timestamps, and a structured report is generated summarizing detected activity, observations, and conclusions.

DETECTION TECHNIQUES IMPLEMENTED

The monitoring agent uses **rule-based detection techniques** to identify suspicious or unauthorized activity related to Windows processes and services. These techniques are designed to be transparent, explainable, and easy to analyze.

The following detection techniques are implemented:

- **Process Enumeration and Metadata Collection**
  Active processes are continuously monitored by collecting process name, PID, parent PID, and executable path to understand runtime behavior.
- **Parent–Child Relationship Analysis**
  Process execution chains are analyzed to detect abnormal parent–child relationships that may indicate suspicious execution patterns.
- **Execution Path-Based Detection**
  Processes and services running from user-writable directories such as AppData, Temp, or Downloads are flagged, as these locations are commonly abused by malware.
- **Service Configuration Auditing**
  Windows services are audited to identify missing executable paths or unusual service configurations that may indicate persistence mechanisms.
- **Whitelist-Based Validation**
  Verified and trusted applications are added to a whitelist to reduce false positives and improve detection accuracy.
- **Severity-Based Classification**
  Detected events are classified into LOW, MEDIUM, or HIGH severity levels based on their potential security impact.

ANALYSIS EVIDENCE AND OBSERVATIONS

This section presents the practical execution of the project along with observations recorded during testing. Screenshots are included to demonstrate system behavior, detection results, and logging output

11.1 Process Enumeration Analysis

In the initial phase, the monitoring agent enumerated all active Windows processes running on the system. Information such as process name, Process ID (PID), Parent Process ID (PPID), and executable path was collected for each process.

Observation:

- The system successfully listed all active processes without errors.
- Process metadata was captured accurately and served as the foundation for further analysis.

```
D:\Cybersecurity_Projects\windows-process-monitor>python agent/process_enum.py
=== Running Process Enumeration ===

Process Name : System Idle Process
PID          : 0
Parent PID   : 0
Path         : None
----------------------------------------
Process Name : System
PID          : 4
Parent PID   : 0
Path         :
----------------------------------------
Process Name : Registry
PID          : 92
Parent PID   : 4
Path         : Registry
----------------------------------------
Process Name : svchost.exe
PID          : 452
Parent PID   : 468
Path         : C:\Windows\System32\svchost.exe
----------------------------------------
Process Name : services.exe
PID          : 468
Parent PID   : 916
Path         : C:\Windows\System32\services.exe
----------------------------------------
Process Name : smss.exe
PID          : 480
Parent PID   : 4
Path         : C:\Windows\System32\smss.exe
```

Fig 2: Process enumeration showing PID, PPID, and executable paths

11.2 Parent–Child Process Analysis

The parent–child relationship between processes was analyzed to identify abnormal execution patterns. This analysis helps detect suspicious behavior where trusted applications may spawn command-line tools or scripting engines.

Observation:

- Parent–child process relationships were correctly mapped.
- No forced or false alerts were generated during normal system operation, indicating accurate rule-based analysis.

```
D:\Cybersecurity_Projects\windows-process-monitor>python agent/parent_child_analysis.py
=== Parent-Child Process Relationship Analysis ===


D:\Cybersecurity_Projects\windows-process-monitor>
```

Fig 3: Parent–child process relationship analysis

11.3 Windows Service Audit Analysis

Windows services were audited using PowerShell (WMI) to collect service name, status, and executable path information. Services with missing or unusual paths were flagged for review.

Observation:

- The system successfully enumerated all Windows services.
- Services with missing executable paths were detected and logged for further analysis.

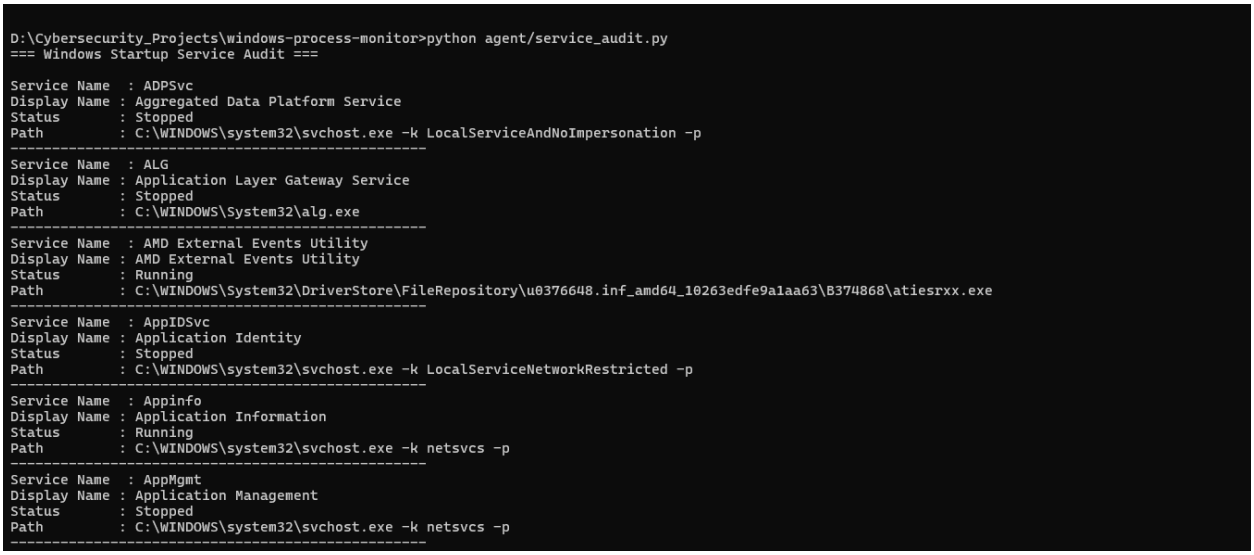


Fig 4: Windows service audit showing service status and executable paths

11.4 Unauthorized Process and Service Detection

The monitoring agent detected processes and services running from user-writable directories such as AppData and Temp, which are commonly abused by malware for execution and persistence.

Observation:

- Processes running from suspicious locations were correctly flagged.
- Development tools such as `Code.exe` (Visual Studio Code) were initially detected due to execution from user directories.
- After verification, trusted applications were added to the whitelist, and subsequent detections were classified as LOW severity, reducing false positives.

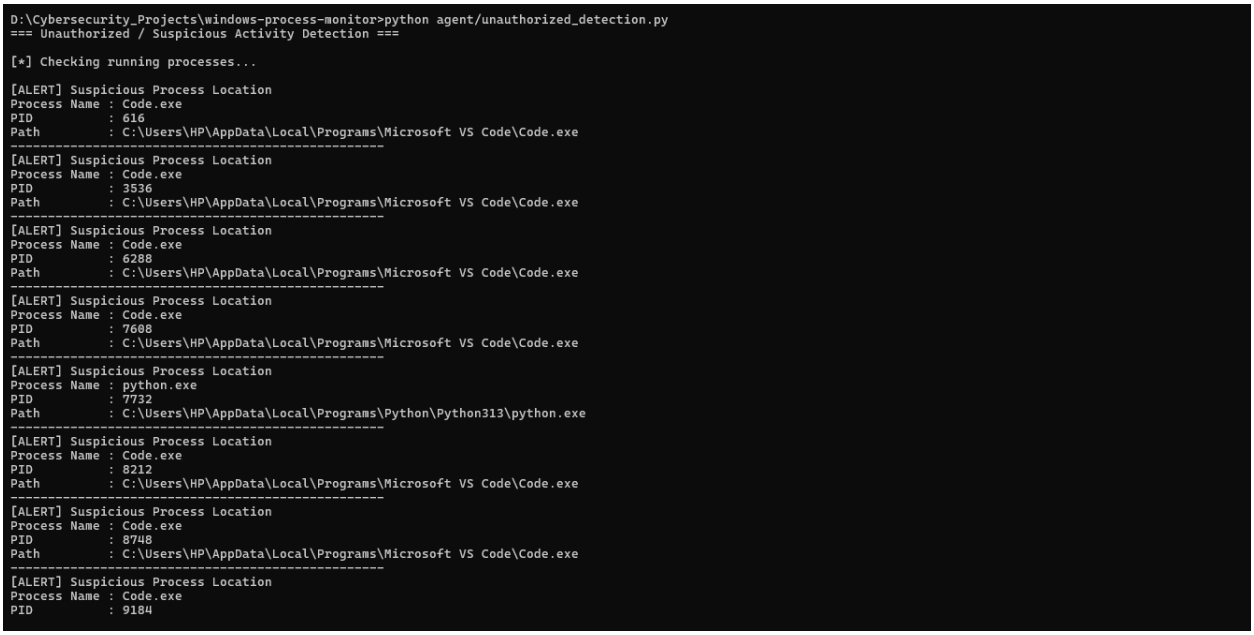


Fig 5: Detection of unauthorized processes based on execution path

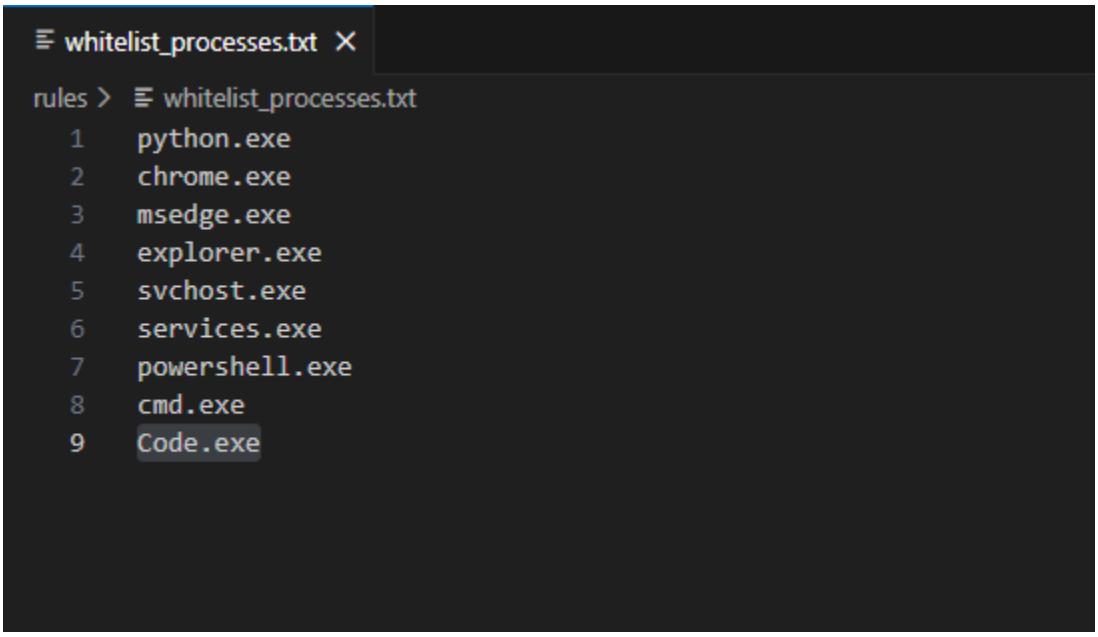


Fig 6: Whitelist configuration used to reduce false positives

11.5 Continuous Monitoring Behavior

The monitoring agent was executed multiple times to simulate continuous system monitoring. Each execution generated a new detection run with timestamps.

Observation:

- Multiple detection runs were recorded successfully.
- Logs accumulated over time, providing a clear timeline of system activity.
- Periodic execution effectively simulated continuous monitoring behavior.

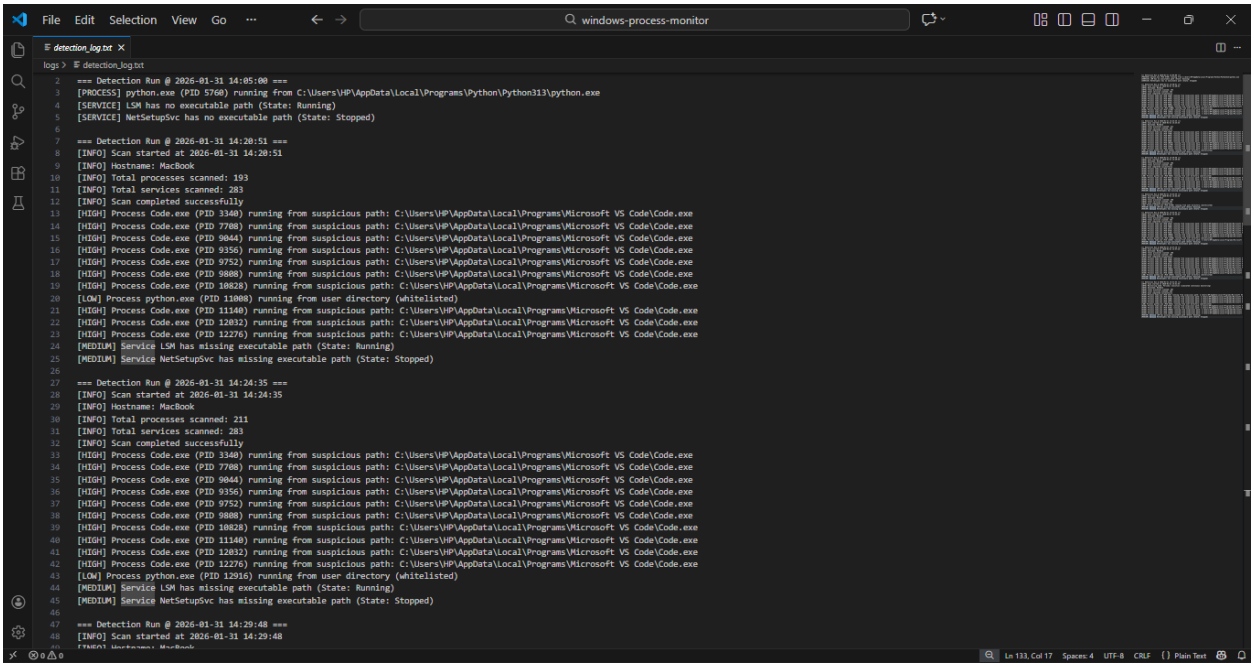


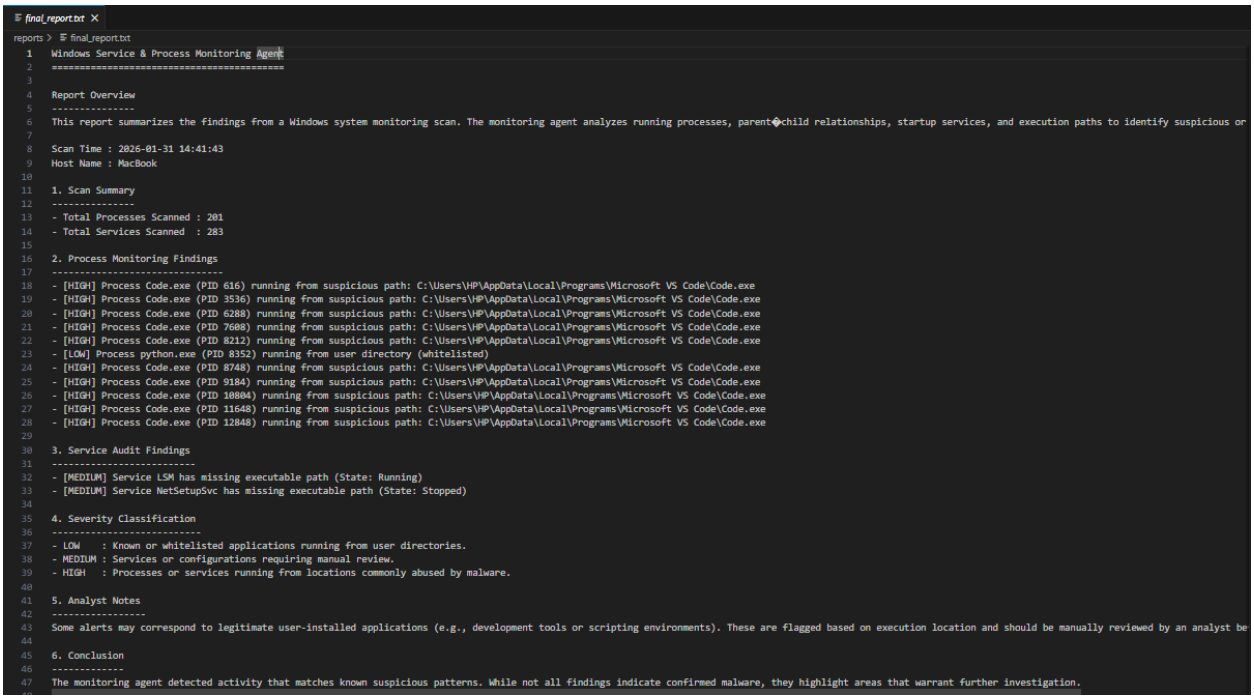
Fig 7: Multiple detection runs demonstrating periodic monitoring

11.6 Log File Analysis

All detected events were recorded in a log file with timestamps, severity levels, and descriptive messages. The log file provides an audit trail for analysis and investigation.

Observation:

- Log entries were consistent with console output.
- Severity levels (LOW, MEDIUM, HIGH) helped prioritize findings.
- The log file clearly demonstrated system behavior across multiple monitoring runs.



```
final_report.txt
reports > final_report.txt
1   Windows Service & Process Monitoring Agent
2   ==========================================
3
4   Report Overview
5   ---------------
6   This report summarizes the findings from a Windows system monitoring scan. The monitoring agent analyzes running processes, parent-child relationships, startup services, and execution paths to identify suspicious or
7
8   Scan Time : 2026-01-31 14:41:43
9   Host Name : MacBook
10
11  1. Scan Summary
12  ---------------
13  - Total Processes Scanned : 201
14  - Total Services Scanned  : 283
15
16  2. Process Monitoring Findings
17  ------------------------------
18  - [HIGH] Process Code.exe (PID 616) running from suspicious path: C:\Users\HP\AppData\Local\Programs\Microsoft VS Code\Code.exe
19  - [HIGH] Process Code.exe (PID 3536) running from suspicious path: C:\Users\HP\AppData\Local\Programs\Microsoft VS Code\Code.exe
20  - [HIGH] Process Code.exe (PID 6288) running from suspicious path: C:\Users\HP\AppData\Local\Programs\Microsoft VS Code\Code.exe
21  - [HIGH] Process Code.exe (PID 7608) running from suspicious path: C:\Users\HP\AppData\Local\Programs\Microsoft VS Code\Code.exe
22  - [HIGH] Process Code.exe (PID 8212) running from suspicious path: C:\Users\HP\AppData\Local\Programs\Microsoft VS Code\Code.exe
23  - [LOW] Process python.exe (PID 8352) running from user directory (whitelisted)
24  - [HIGH] Process Code.exe (PID 8748) running from suspicious path: C:\Users\HP\AppData\Local\Programs\Microsoft VS Code\Code.exe
25  - [HIGH] Process Code.exe (PID 9184) running from suspicious path: C:\Users\HP\AppData\Local\Programs\Microsoft VS Code\Code.exe
26  - [HIGH] Process Code.exe (PID 10804) running from suspicious path: C:\Users\HP\AppData\Local\Programs\Microsoft VS Code\Code.exe
27  - [HIGH] Process Code.exe (PID 11648) running from suspicious path: C:\Users\HP\AppData\Local\Programs\Microsoft VS Code\Code.exe
28  - [HIGH] Process Code.exe (PID 12848) running from suspicious path: C:\Users\HP\AppData\Local\Programs\Microsoft VS Code\Code.exe
29
30  3. Service Audit Findings
31  -------------------------
32  - [MEDIUM] Service LSM has missing executable path (State: Running)
33  - [MEDIUM] Service NetSetupSvc has missing executable path (State: Stopped)
34
35  4. Severity Classification
36  --------------------------
37  - LOW    : Known or whitelisted applications running from user directories.
38  - MEDIUM : Services or configurations requiring manual review.
39  - HIGH   : Processes or services running from locations commonly abused by malware.
40
41  5. Analyst Notes
42  ----------------
43  Some alerts may correspond to legitimate user-installed applications (e.g., development tools or scripting environments). These are flagged based on execution location and should be manually reviewed by an analyst be
44
45  6. Conclusion
46  -------------
47  The monitoring agent detected activity that matches known suspicious patterns. While not all findings indicate confirmed malware, they highlight areas that warrant further investigation.
```

Fig 8: Detection log showing severity-based alerts and timestamps

```
4. Severity Classification
---------------------------
- LOW    : Known or whitelisted applications running from user directories.
- MEDIUM : Services or configurations requiring manual review.
- HIGH   : Processes or services running from locations commonly abused by malware.
```

Fig 9: Severity Classification Used for Process and Service Detection

This figure illustrates the severity classification logic used by the monitoring agent to categorize detected events into LOW, MEDIUM, and HIGH risk levels. Severity levels help prioritize alerts and assist analysts in distinguishing between informational findings, suspicious activity, and high-risk behavior requiring further investigation.

Risk Assessment

Abuse of Windows processes and services poses a significant security risk because malicious activity can execute under the guise of legitimate system components. Malicious processes may run silently in the background, while unauthorized services can persist across system reboots without user awareness.

Processes or services running from user-writable directories such as AppData or Temp increase the risk of compromise, as these locations are commonly targeted by malware for execution and persistence. Abnormal parent–child process relationships further increase the likelihood of malicious behavior.

Early detection of such activity is critical for minimizing long-term system compromise. The monitoring agent helps reduce this risk by providing visibility into suspicious process and service behavior, enabling timely investigation and response.

Challenges Faced

During the development of this project, several challenges were encountered:

- Understanding normal versus suspicious process behavior, as modern Windows systems run a large number of legitimate background processes
- Handling false positives caused by trusted user-installed applications executing from user-writable directories
- Managing permission-related access issues when querying certain system processes and services
- Designing detection rules that are effective without generating excessive or misleading alerts
- Organizing logs, reports, and screenshots in a clear and structured manner for evaluation

These challenges helped improve practical understanding of Windows internals, defensive detection logic, and SOC-style analysis workflows.

Ethical and Safety Considerations

During the development of this project, several challenges were encountered:

- Understanding normal versus suspicious process behavior, as modern Windows systems run a large number of legitimate background processes
- Handling false positives caused by trusted user-installed applications executing from user-writable directories
- Managing permission-related access issues when querying certain system processes and services
- Designing detection rules that are effective without generating excessive or misleading alerts
- Organizing logs, reports, and screenshots in a clear and structured manner for evaluation

These challenges helped improve practical understanding of Windows internals, defensive detection logic, and SOC-style analysis workflows.

LEARNING OUTCOMES

Through the implementation of this project, the following learning outcomes were achieved:

- Gained practical understanding of Windows process and service architecture
- Learned how malware can abuse processes and services for execution and persistence
- Developed hands-on experience with Python-based system monitoring
- Understood the importance of parent–child process relationships in threat detection
- Learned how to reduce false positives using whitelist-based validation
- Gained experience in SOC-style logging, severity classification, and reporting

This project helped strengthen foundational knowledge of defensive cybersecurity and system monitoring techniques.

Future Scope

The current implementation of the monitoring agent provides a strong foundation for detecting suspicious process and service behavior. In the future, the system can be enhanced in several ways to improve effectiveness and usability.

Possible future enhancements include:

- Implementing real-time event-based monitoring instead of periodic execution
- Adding alert notifications such as email or dashboard-based alerts
- Expanding detection rules using additional behavioral indicators
- Integrating basic threat intelligence for improved context
- Developing a simple graphical interface for easier monitoring and analysis

These enhancements can further improve the system's capability as a lightweight endpoint monitoring tool.

CONCLUSION

This project successfully demonstrates the design and implementation of a **Windows Service & Process Monitoring Agent** for defensive cybersecurity purposes. By monitoring running processes, analyzing parent–child relationships, auditing Windows services, and detecting unauthorized execution patterns, the system provides clear visibility into potentially suspicious system activity.

The use of rule-based detection, severity classification, and whitelist validation ensures that alerts are meaningful and explainable, while reducing false positives. Timestamped logs and structured reports further support investigation and analysis by presenting system behavior in a clear and organized manner.

Overall, the project highlights the importance of process and service monitoring in identifying security risks and provides practical insight into how basic SOC monitoring tools operate. It serves as a solid foundation for understanding endpoint monitoring and defensive security techniques.

BIBLIOGRAPHY AND REFERENCES

1. Microsoft Documentation – Windows Processes and Services
2. Microsoft Documentation – Windows Service Management
3. Python `psutil` Library Documentation
4. PowerShell and Windows Management Instrumentation (WMI) Documentation
5. MITRE ATT&CK Framework – Execution and Persistence Techniques
6. draw.io (diagrams.net) – System Architecture Diagram Tool