PROJECT REPORT

Title: Sentinel Endpoint Monitor: A Behavior-Based Host Intrusion Detection System (HIDS)

Submitted by: Gunta Subham **Degree:** B.Tech in Computer Science **Domain:** Cybersecurity

1. Abstract

The Sentinel Endpoint Monitor is a host-based security tool designed to detect malicious activity on Windows endpoints in real-time. Unlike traditional antivirus software that relies on static file signatures, this system employs **behavioral analysis** to identify suspicious patterns, such as unauthorized persistence mechanisms, process injection attempts, and execution from volatile directories. The application features a Python-based backend agent for continuous system monitoring and a Tkinter-based Graphical User Interface (GUI) for alert visualization and risk assessment.

2. Introduction

2.1 Problem Statement

Modern malware often evades traditional detection by living off the land (LotL)—using legitimate system tools like PowerShell or CMD to execute attacks. Standard monitoring tools often lack visibility into these specific behavioral anomalies or are too complex for rapid deployment.

2.2 Objective

The primary objective of this project is to develop a lightweight, standalone agent that:

- Monitors system processes, services, and startup registries.
- Scores system events based on risk (0-100).
- Provides a visual dashboard for security analysts to review alerts.
- Maintains a local log of potential security incidents.

3. System Analysis & Design

3.1 Technical Architecture

The system is built using a modular architecture separating the monitoring logic from the user interface.

- **Backend (The Agent):** A multi-threaded engine using the psutil and winreg libraries to query the Windows Kernel. It runs in a daemon thread, polling the system every 3 seconds.

- **Frontend (The Dashboard):** A tkinter GUI that parses the JSON logs generated by the agent and displays them in a color-coded tree view.
- **Data Storage:** A local JSON file (monitor_logs.json) acts as the persistent storage for alert history.

3.2 Key Features

1. **Process Tree Analysis:** Detects suspicious parent-child relationships (e.g., MS Word spawning PowerShell).
2. **Persistence Detection:** Monitors Registry Run keys, Windows Services, and Scheduled Tasks for unauthorized startup entries.
3. **Heuristic Risk Scoring:**
    - **0-40 (Safe):** Standard background noise.
    - **40-60 (Review):** Unknown binaries or unusual parents.
    - **60-80 (Suspicious):** Execution from AppData, Temp, or Downloads.
    - **80+ (Critical):** Known evasion patterns or persistence creation.
4. **Self-Preservation:** The tool attempts to install itself as a Scheduled Task (EndpointMonitorAgent) to ensure it runs on system boot.

4. Implementation Details

4.1 Technology Stack

- **Language:** Python 3.x
- **Core Libraries:** psutil (System iterators), winreg (Registry access), subprocess (Command execution), ctypes (Windows API integration).
- **GUI Framework:** tkinter & ttk (Themed widgets).

4.2 Core Modules

- **check_process(proc)**: Iterates through running processes. It flags binaries running from user-writable directories (like %TEMP%) which is a common tactic for droppers.
- **check_startup_registry()**: Scans HKCU and HKLM "Run" and "RunOnce" keys to detect malware trying to survive a reboot.
- **ensure_startup()**: A utility function that uses the Windows schtasks command to create a high-privilege task for the agent itself.

4.3 Code Logic Example (Risk Scoring)

The system assigns risk points dynamically. For example:

- +40 points if explorer.exe spawns powershell.exe.
- +30 points if the file path contains \temp\.
- +15 points if the binary has not been seen before in the current session.

5. Results and Output

5.1 Dashboard Interface

The application launches a 1300x700 window with a "Dark Mode" theme (GitHub Dark style).

- **Left Panel:** Displays a table of alerts with columns for Timestamp, Detection Type, and Risk Score. High-risk items are highlighted in Red.
- **Right Panel:** Shows a live list of all active processes on the system.
- **Status Bar:** Indicates agent health and total alert count.

5.2 Event Logging

All detections are serialized to monitor_logs.json. An example log entry:

JSON


```
{
  "timestamp": "2026-01-28 10:45:12",
  "type": "Process Execution Detected",
  "process": "unknown.exe",
  "risk_score": 65,
  "reasons": ["Executed from user-writable directory"]
}
```


6. Limitations & Future Scope

- **Current Limitation:** The tool currently logs locally. If the machine is compromised, the attacker could delete the JSON log.
- **Future Enhancement:** Implement centralized logging (sending logs to a remote SIEM or Syslog server).
- **Future Enhancement:** Add "Active Response" capabilities to automatically kill processes with a risk score > 90.

7. Conclusion

The Sentinel Endpoint Monitor successfully demonstrates how low-level system APIs can be leveraged to build a functional security monitoring tool. By focusing on behavior rather than signatures, it offers a proactive layer of defense against modern threats like fileless malware and persistence backdoors.

8. References

1. Python Software Foundation. (n.d.). *psutil documentation*.
2. Microsoft Corporation. (n.d.). *Windows Registry API*.
3. MITRE ATT&CK Framework. (n.d.). *Technique T1547: Boot or Logon Autostart Execution*.

9. Test Cases & Validation

The system was tested using a variety of scenarios to validate both functional reliability and security detection logic. The following test cases cover normal operations (Positive Testing) and simulated attack scenarios (Negative/Security Testing).

9.1 Functional Test Cases

Test Case ID	Test Scenario	Steps to Reproduce	Expected Output	Actual Result	Status
TC-F-01	**Agent Startup**	Run final.exe as Administrator.	GUI should launch; Status bar should show "Agent Status: Active".	GUI launched successfully.	**Pass**
TC-F-02	**Live Process Listing**	Open the dashboard and check the "Live Processes" panel on the right.	List should populate with currently running system processes (e.g., chrome.exe, svchost.exe).	List populated and sorted alphabetically.	**Pass**
TC-F-03	**Log Filtering**	Type "svchost" into the "Filter" search bar at the top.	The alert tree view should only show rows containing "svchost".	Table filtered correctly in real-time.	**Pass**

Test Case ID	Test Scenario	Steps to Reproduce	Expected Output	Actual Result	Status
TC-F-04	**Persistence Installation**	Run the agent and restart the computer.	The agent should automatically start upon user login via Windows Task Scheduler.	Task EndpointMonitorAgent was created and ran on boot.	**Pass**

9.2 Security Logic Test Cases (Behavioral Detection)

Test Case ID	Test Scenario	Steps to Reproduce	Risk Score Calculation	Expected Alert Level	Status
TC-S-01	**Suspicious Directory Execution**	Copy a safe executable (e.g., notepad.exe) to the %TEMP% folder and run it.	+30 (Writable Directory) +15 (First Time Seen) = **45**	**Review (Yellow)**	**Pass**
TC-S-02	**Suspicious Parent Process**	Launch cmd.exe directly from explorer.exe (simulating a user opening a terminal).	+40 (Suspicious Parent) + 20 (Unknown Trust) = **60**	**Suspicious (Orange)**	**Pass**
TC-S-03	**Registry Persistence Attempt**	Manually add a test key to HKCU\Software\Microsoft\Windows\CurrentVersion\Run pointing to a script in Downloads.	+60 (Startup Persistence) = **60**	**Suspicious (Orange)**	**Pass**
TC-S-04	**Combined Threat (Simulate**	Create a Python script, compile it to .exe, place it in AppData, and run it.	+30 (Writable Path) +15	**Critical (Red)**	**Pass**

Test Case ID	Test Scenario	Steps to Reproduce	Risk Score Calculation	Expected Alert Level	Status
	d Malware)		(Non-Standard Dir) +15 (First Time) +20 (Unknown) = **80**		

9.3 Stress & Reliability Testing

- **Objective:** Ensure the agent does not crash under load or blocking errors.
- **Scenario:** The agent was left running for 4 hours while opening multiple heavy applications (VS Code, Chrome with 20 tabs, Games).
- **Observation:** The standard RAM usage remained below 50MB, and the UI remained responsive. The agent_loop thread successfully handled Access Denied errors for system-protected processes without crashing the main application.