Intelligent Task Manager

Introduction:

This project provides an integrated suite of tools designed to enhance the monitoring, security, and management of a computer system. It aims to tackle critical aspects of system health, network security, and hardware control by using real-time monitoring and advanced detection techniques. By leveraging automation and user-friendly interfaces, the solution empowers users to maintain robust security measures, identify and mitigate potential threats, and manage hardware efficiently.

Key Features:

1. Process and Network Monitoring

- **Real-Time Process Tracking:**
    - Monitors all active processes running on the system, providing real-time updates about their state, resource usage (CPU, memory), and origin.
    - Helps identify unauthorized or malicious processes to prevent security breaches.
- **Network Activity Monitoring:**
    - Tracks inbound and outbound network traffic, analyzing connections to detect abnormal activities such as unauthorized data transfers or communication with suspicious IPs.
    - Logs network activity for later review, offering transparency and control over system operations.

2. Process Chain Management

- **Parent-Child Process Tracking:**
    - Examines the relationship between processes, identifying parent and child processes to monitor process hierarchies.
    - Flags unauthorized child processes spawned by malicious or compromised parent processes.
- **Anomaly Detection in Process Chains:**
    - Automatically identifies irregularities in process behavior or execution order that may indicate malware or exploits.

3. Device Control

- **Microphone and Camera Access Management:**

    - A graphical user interface (GUI) allows users to enable or disable access to the microphone and camera with ease.
    - Prevents unauthorized applications or attackers from accessing sensitive hardware, enhancing user privacy.
- **Hardware Access Logs:**

    - Maintains a log of all access attempts to controlled devices, aiding in forensic analysis if needed.

4. File Scanning for Malware

- **Executable File Scanning:**
    - Integrates with the **MetaDefender API** to scan all running executables for known malware signatures and potential threats.
    - Ensures timely alerts and reports, enabling users to act before damage occurs.
- **Automated Quarantine:**
    - Malicious files detected during scans are automatically flagged or quarantined to prevent further harm to the system.

5. IP Threat Analysis

- **Packet Capture and Analysis:**
    - Uses **Scapy** to capture live network packets and analyze them for suspicious patterns or threats.
    - Examines packet headers and payloads to detect malicious activities, such as data exfiltration or denial-of-service attempts.
- **Malicious IP Detection:**
    - Compares captured IP addresses against known threat databases to identify and block potentially harmful connections.
    - Supports automated blocking of flagged IPs through firewall configurations.

6. Hardware Monitoring

- **USB and Audio Device Tracking:**
    - Continuously monitors connected USB devices (e.g., storage drives, input peripherals) and audio devices.
    - Flags unauthorized USB connections or devices with suspicious behavior.

Technology Stack:

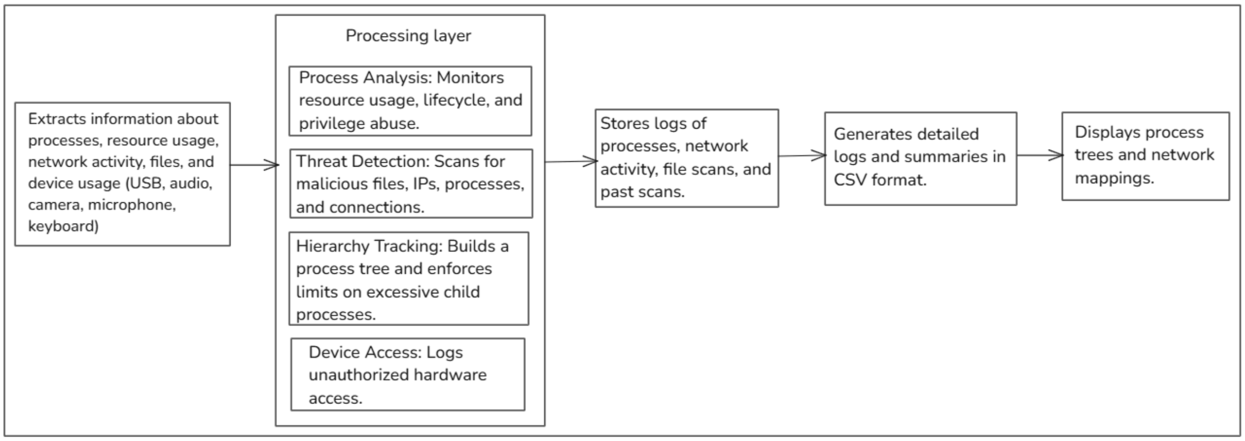
Primary Frameworks and Tools:

- **Python:**
  - The core programming language used to develop the scripts, chosen for its versatility and rich library ecosystem.
- **Scapy:**
  - A powerful library for capturing, decoding, and analyzing network packets to implement IP threat analysis and packet monitoring.
- **psutil:**
  - Enables system process tracking, resource usage analysis, and hardware monitoring.
- **tkinter:**
  - Provides a lightweight and user-friendly GUI to manage device controls such as microphone and camera access.
- **MetaDefender API:**
  - Offers robust malware detection capabilities by scanning files against a vast database of malware signatures.
- **WMI (Windows Management Instrumentation):**
  - Allows deep integration with Windows systems for querying and controlling hardware devices.

Auxiliary Technologies:

- **PowerShell Scripts:**
  - Used to control hardware features, such as enabling or disabling devices, adding an additional layer of device management flexibility.

Architecture and Workflow:


Extracts information about processes, resource usage, network activity, files, and device usage (USB, audio, camera, microphone, keyboard)
Processing layer
Process Analysis: Monitors resource usage, lifecycle, and privilege abuse.
Threat Detection: Scans for malicious files, IPs, processes, and connections.
Hierarchy Tracking: Builds a process tree and enforces limits on excessive child processes.
Device Access: Logs unauthorized hardware access.
Stores logs of processes, network activity, file scans, and past scans.
Generates detailed logs and summaries in CSV format.
Displays process trees and network mappings.

The system operates through a structured workflow involving data collection, processing, analysis, and reporting. Each layer is designed to work in real-time, ensuring continuous monitoring and prompt response to potential threats.

1. Information Extraction

This layer collects and aggregates data from various system components to provide a comprehensive overview.

- **Process Monitoring:**
    - Captures active processes running on the system, including their resource usage (CPU, memory) and operational status.
    - Monitors the origin of each process to detect unauthorized or suspicious executions.
- **Network Connection Tracking:**
    - Logs all active network connections, recording details such as IP addresses, ports, and protocols used.
    - Identifies unusual or unauthorized connections that may indicate data leaks or attacks.
- **Hardware Access Monitoring:**
    - Tracks hardware interactions, including USB device connections, microphone access, and camera usage.
    - Logs any unauthorized attempts to access these components.

2. Processing Layer Functions

The processing layer applies advanced analytical functions to identify anomalies, detect threats, and manage system behavior.

a. Process Analysis

- **Unusual Activity Detection:**
    - Identifies processes exhibiting unusual behavior, such as running unauthorized scripts or accessing restricted areas of the system.
- **Resource Consumption Monitoring:**
    - Flags processes consuming excessive CPU, memory, or disk resources, which could indicate malware or inefficient applications.
- **Privilege Escalation Detection:**
    - Tracks processes attempting to gain elevated privileges without proper authorization.

b. Threat Detection

- **File Scanning:**
    - Uses the MetaDefender API to scan files and running executables for known malware or suspicious patterns.
- **Network Traffic Analysis:**
    - Analyzes packet data to detect malicious payloads or traffic patterns indicative of an attack.

- **IP Reputation Check:**
    - Compares network IPs against threat intelligence databases to identify known malicious addresses.

c. Hierarchy Tracking

- **Process Tree Enforcement:**
    - Maintains a hierarchical structure of parent and child processes, ensuring process chains follow legitimate paths.
    - Limits the creation of excessive subprocesses, which may be indicative of process injection or botnet activity.

d. Device Access Management

- **Unauthorized Access Logging:**
    - Logs any attempts to access restricted devices such as USB storage, microphone, or camera without proper permissions.
- **Proactive Alerts:**
    - Sends alerts when unauthorized access attempts are detected, enabling timely user intervention.

3. Data Output

This layer ensures all collected and processed information is presented in a structured and actionable format.

- **Data Logging:**
    - Stores detailed logs of processes, network activity, and hardware interactions in easily accessible formats like CSV.
    - Generates clear and detailed mappings of system processes and network connections to facilitate better understanding and analysis.

Current Implementation Status:

The current implementation consists of **six separate scripts**, each performing a specific function to monitor and secure the system. Although these scripts are operational, they are not yet integrated into a unified system. Below is a detailed description of the functionality and status of each script:

1. Process and Network Monitoring

- **Functionality:**
    - This script collects detailed information about system processes and their associated network connections over a specified duration (default of 15 minutes).
    - It gathers details such as **Process ID (PID)**, **CPU usage**, **memory usage**, **I/O counters**, and **parent-child relationships** among processes.
    - It also monitors **network connections**, logging source and destination IPs, ports, and statuses for each process.
- **Current Status:**
    - The script is fully operational and collects comprehensive process and network information.
    - The data is exported to a **CSV file**, ensuring that the collected data is well-structured and can be used for future analysis.
    - The script operates in a separate thread for **non-blocking execution**, ensuring that it does not interfere with other tasks on the system.
    - **Challenges:** No real-time dashboard or centralized reporting exists yet.

2. Process Chain Monitoring and Control

- **Functionality:**
    - This script focuses on monitoring specific processes and managing their child processes.
    - It builds a **process tree**, mapping parent-child relationships, and monitors processes in real time.
    - The script can **terminate excessive child processes** if the number of child processes exceeds a set threshold.
- **Current Status:**
    - The process monitoring and child process management features are working as expected.
    - **Real-time monitoring** runs every 2 seconds, ensuring continuous updates on process status.
    - However, there is no integration with other modules yet, and data is logged separately.

3. Microphone and Camera Control

- **Functionality:**
    - This script provides a **graphical user interface (GUI)** for controlling the system's microphone and camera access.
    - Users can **enable, disable, or reset** the access permissions for the microphone and camera using the GUI.

    - The script uses **PowerShell commands** to change device settings and ensures that the application runs with **admin privileges**.
- **Current Status:**
    - The GUI is functional, allowing users to interact with the system and manage microphone and camera access.
    - The script displays **success or failure messages** based on user actions, ensuring that the user is informed of the changes made.
    - However, there is no error handling for **PowerShell command failures** due to permissions or incorrect paths, and no logging mechanism is in place yet.

4. File Scanning with MetaDefender

- **Functionality:**
    - This script scans all currently running executables for malware using the **MetaDefender Cloud API**.
    - It integrates **API keys** for scanning files and retrieves scan reports to check for malware.
    - The script also maintains a **scan history** to avoid redundant scans, storing previously scanned files in a **scan_history.json** file.
- **Current Status:**
    - The scanning functionality works correctly, successfully uploading files to MetaDefender and retrieving scan results.
    - The script uses a **round-robin approach** to balance the API key usage and avoid hitting rate limits.
    - The **scan history** is maintained, ensuring that previously scanned files are not resubmitted.
    - Some minor errors occur when files cannot be scanned or the scan report is unavailable, which needs further handling.
    - **Challenges:** There is a need to improve error handling and optimize performance, especially when dealing with failed API calls.

5. IP Threat Analysis with Chaining and Logging

- **Functionality:**
    - This script is designed to capture network traffic and analyze it for potential threats.
    - It automatically **downloads blocklists** of malicious IPs and uses a **Bloom filter** for fast lookups.
    - The script captures **IP packets** using **Scapy**, extracts source and destination IPs, and checks them against the blocklist.

    - It also integrates with the **Google Safe Browsing API** for reputation analysis and performs a basic reputation check.
- **Current Status:**
    - The script is operational and performs **real-time IP threat analysis**, checking for malicious IPs and logging the results.
    - The **Bloom filter** optimizes IP lookup speed, ensuring efficient detection.
    - The **logging functionality** is working, and results are stored in a CSV file on the desktop.
    - **Challenges:** The logging can be further optimized, and data aggregation for analysis purposes is missing. Additionally, periodic blocklist updates occur every 24 hours, but more frequent updates might be beneficial.

6. Monitoring USB and Audio Devices

- **Functionality:**
    - This script monitors connected **USB** and **audio devices** on the system.
    - It lists all **USB devices** connected to the system and also tracks **audio devices** using the **wmi** module.
    - The script can be extended to include other hardware devices in future iterations.
- **Current Status:**
    - The USB and audio monitoring features are fully functional.
    - It successfully lists all connected devices and provides real-time updates.
    - **Challenges:** Expanding the scope to include other hardware devices, such as network adapters and storage devices, is needed to make it more comprehensive.

Future Scope

The project has significant potential for enhancement, with planned features aimed at increasing functionality, scalability, and user convenience. Below are the details of the envisioned improvements:

1. ML-Based Anomaly Detection

- **Objective:**
    - Train an anomaly detection model using unsupervised machine learning techniques.
    - Detect deviations from normal behavior patterns in system processes, network traffic, and resource usage.
- **Implementation:**
    - **Data Preprocessing:**

        - Extract features such as CPU usage, memory usage, I/O statistics, and network activity logs.
        - Normalize and scale the data for consistent input.
    - **Model Training:**
        - Use unsupervised algorithms like:
            - **Isolation Forest:** Identifies anomalies based on their separability in feature space.
            - **Autoencoders:** Learns compressed representations of normal data and flags instances with high reconstruction error.
            - **K-Means Clustering:** Groups data into clusters and identifies points farthest from cluster centers as anomalies.
    - **Real-Time Detection:**
        - Continuously feed data to the model to flag unusual activity and provide alerts.
- **Impact:**
    - Automates threat detection without requiring labeled datasets.
    - Improves system adaptability to new or evolving threats by identifying patterns not explicitly modeled.

2. Cloud-Based Storage

- **Objective:**
    - Store monitoring data in cloud services (e.g., AWS S3, Google Cloud) to ensure scalability, remote accessibility, and data security.
- **Implementation:**
    - **Integration with Cloud Services:** Set up APIs to upload monitoring data to the cloud.
    - **Backup and Analysis:** Use cloud-based tools for data backup, advanced analysis, and visualization (e.g., AWS QuickSight, Google Data Studio).
- **Impact:**
    - Provides a centralized repository for all monitoring data, enabling secure access from multiple devices.
    - Facilitates long-term storage and detailed analysis for trend detection and compliance reporting.

3. Executable Integration

- **Objective:**
    - Integrate the unsupervised anomaly detection models into the executable for seamless deployment.
- **Implementation:**

    - Package the trained model alongside the scripts using tools like **ONNX** (for model conversion) and **PyInstaller** (for packaging).
    - Design a GUI to display real-time anomaly detection results with a user-friendly interface.

4. Alert System

- **Objective:**
    - Modify the alert mechanism to include anomalies detected by the unsupervised ML models.
- **Implementation:**
    - Define thresholds for anomaly scores generated by the models.
    - Notify users in real-time when scores exceed these thresholds, using email or desktop alerts.

Estimated Time Frame:

Feature	Task	Duration
ML-Based Anomaly Detection	Data preprocessing and feature engineering	~1 week
	Model training and evaluation	~1 week
	Integration with monitoring scripts	~1 week
	Total	~3 weeks
Cloud-Based Storage	Cloud service setup and integration	~1 week
	Data backup and visualization tools	~1 week
	Total	~2 weeks
Executable Integration	Packaging scripts and models into an EXE	~1 week
	GUI design and implementation	~1 week
	Testing and debugging	~1 week

	Total	~3 weeks
Alert System	Modifying alert mechanism for anomaly detection	~1 week
	Cloud-based alert delivery integration	~1 week
	Total	~2 weeks

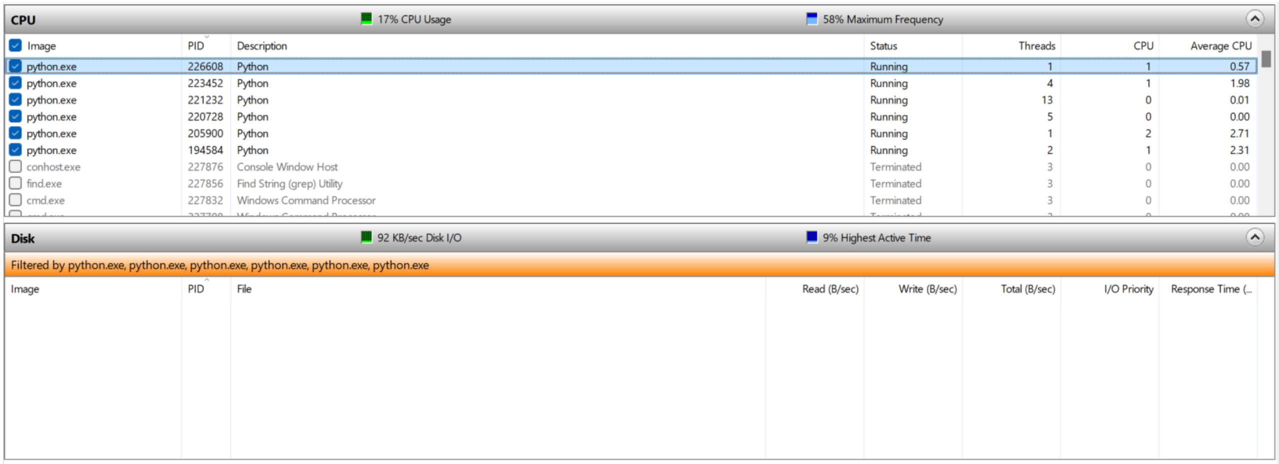
Overall Estimated Time Frame:

- **~10 weeks (2.5 months)** for full implementation, testing, and deployment.

Overall Resource Utilization:

The resource utilization for the system, both **before** and **after** the integration of machine learning (ML) for anomaly detection, provides a comprehensive view of how the system's performance will be affected by the new ML features.

1. Current Resource Utilization (Before ML Integration)



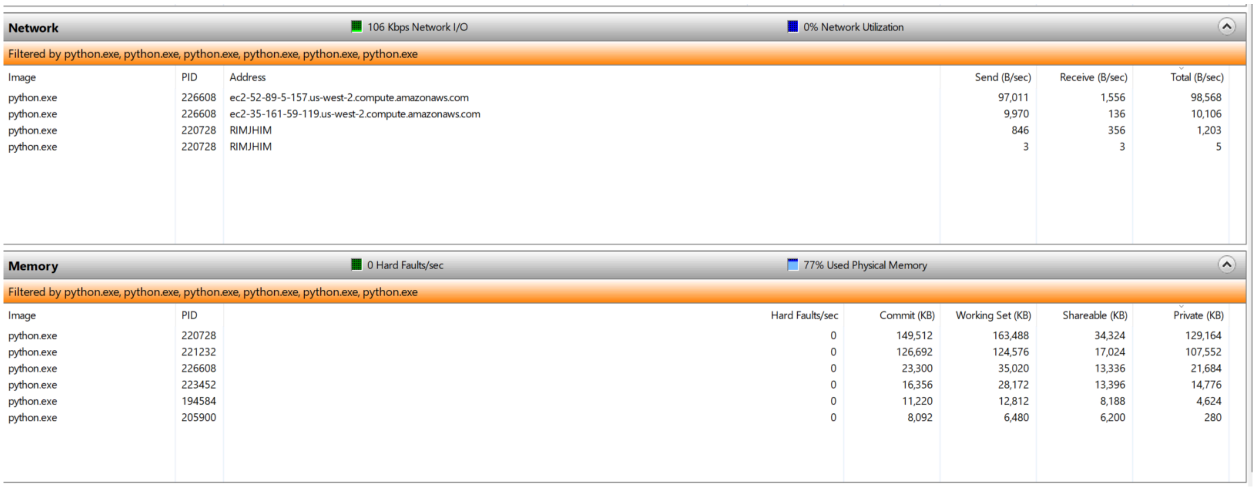
CPU — 🟩 17% CPU Usage — 🟦 58% Maximum Frequency

☑ Image	PID	Description	Status	Threads	CPU	Average CPU
☑ python.exe	226608	Python	Running	1	1	0.57
☑ python.exe	223452	Python	Running	4	1	1.98
☑ python.exe	221232	Python	Running	13	0	0.01
☑ python.exe	220728	Python	Running	5	0	0.00
☑ python.exe	205900	Python	Running	1	2	2.71
☑ python.exe	194584	Python	Running	2	1	2.31
☐ conhost.exe	227876	Console Window Host	Terminated	3	0	0.00
☐ find.exe	227856	Find String (grep) Utility	Terminated	3	0	0.00
☐ cmd.exe	227832	Windows Command Processor	Terminated	3	0	0.00

Disk — 🟩 92 KB/sec Disk I/O — 🟦 9% Highest Active Time

Filtered by python.exe, python.exe, python.exe, python.exe, python.exe, python.exe

Image	PID	File	Read (B/sec)	Write (B/sec)	Total (B/sec)	I/O Priority	Response Time (...



Network 🟩 106 Kbps Network I/O 🟦 0% Network Utilization

Filtered by python.exe, python.exe, python.exe, python.exe, python.exe, python.exe

Image	PID	Address	Send (B/sec)	Receive (B/sec)	Total (B/sec)
python.exe	226608	ec2-52-89-5-157.us-west-2.compute.amazonaws.com	97,011	1,556	98,568
python.exe	226608	ec2-35-161-59-119.us-west-2.compute.amazonaws.com	9,970	136	10,106
python.exe	220728	RIMJHIM	846	356	1,203
python.exe	220728	RIMJHIM	3	3	5

Memory 🟩 0 Hard Faults/sec 🟦 77% Used Physical Memory

Filtered by python.exe, python.exe, python.exe, python.exe, python.exe, python.exe

Image	PID	Hard Faults/sec	Commit (KB)	Working Set (KB)	Shareable (KB)	Private (KB)
python.exe	220728	0	149,512	163,488	34,324	129,164
python.exe	221232	0	126,692	124,576	17,024	107,552
python.exe	226608	0	23,300	35,020	13,336	21,684
python.exe	223452	0	16,356	28,172	13,396	14,776
python.exe	194584	0	11,220	12,812	8,188	4,624
python.exe	205900	0	8,092	6,480	6,200	280

- **CPU Stats**:
  - **No. of Threads:** 26
  - **Average CPU Consumption by Processes:** 7.58%
  - These stats reflect typical resource consumption by the system's processes without the computational load added by ML models.
- **Network Activity**:
  - **Sent:** 107,830 B/sec (~105.37 KB/sec)
  - **Received:** 2,051 B/sec (~2.00 KB/sec)
  - **Total Network Activity:** 109,882 B/sec (~107.19 KB/sec)
  - This represents the base network traffic due to the ongoing system operations and communication requirements.
- **Memory Usage**:
  - **Commit:** 335,172 KB (~327.5 MB)
  - **Working Set:** 370,548 KB (~361.8 MB)
  - **Shareable:** 92,468 KB (~90.3 MB)
  - **Private:** 278,080 KB (~271 MB)
  - These values reflect memory utilization by system processes, with committed memory being the total amount of memory allocated by the OS, while private memory is dedicated solely to individual processes.

2. Estimated Resource Utilization After ML Integration

The integration of ML models for real-time anomaly detection will introduce some changes to the overall resource utilization, including increases in CPU usage, network traffic, memory usage, and disk space.

- **CPU Stats**:
  - **No. of Threads:** 28 (an increase of 2 threads due to the ML model execution)

    - **Average CPU Consumption:** ~8.5% (a slight increase from 7.58% due to the additional computational requirements of ML, especially during inference)
        - **Reasoning:** The ML model adds extra computation overhead during real-time detection, especially when processing new data for anomaly identification.
- **Network Activity**:
    - **Sent:** ~110,000 B/sec (~107.5 KB/sec) (a mild increase due to additional cloud-based communication for real-time threat updates and data storage)
    - **Received:** ~2,100 B/sec (~2.05 KB/sec) (slightly increased due to the model's need to fetch updates or additional data from the cloud)
    - **Total Network Activity:** ~112,100 B/sec (~109.55 KB/sec)
        - **Reasoning:** The integration of cloud-based threat updates and alert systems slightly increases the network activity, but the increase is moderate compared to other resources.
- **Memory Usage**:
    - **Commit:** ~340,000 KB (~332 MB) (slightly increased due to the memory overhead from the ML model and handling additional network traffic)
    - **Working Set:** ~380,000 KB (~371 MB) (additional memory needed for storing and processing data from the ML model)
    - **Shareable:** ~95,000 KB (~93 MB) (slight increase as some operations in the ML model can share resources)
    - **Private:** ~290,000 KB (~283 MB) (increase in private memory due to dedicated resources for the ML model)
        - **Reasoning:** The introduction of the ML model necessitates more memory for model execution, data processing, and storage of temporary results.
- **Total Disk Usage**:
    - **Data Storage (logs and CSV files):** ~50-100 MB per day (depending on data collection frequency)
        - **Reasoning:** Disk space usage for logging and storing monitoring data will increase as logs and CSV files accumulate with continuous system monitoring.
    - **Scan History (MetaDefender API results):** ~20-50 MB per day
        - **Reasoning:** As the file scanning functionality remains active, additional space will be needed for storing the results from the MetaDefender API.
    - **Disk Space Used by ML Models:** 100-150 MB (for model weights and training data)
        - **Reasoning:** The ML models will require additional storage space for saving their weights and related training data, especially if the models are saved locally.

Conclusion:

The **Intelligent Task Manager** is a robust and adaptable tool designed to address the growing demands of system monitoring, network security, and device management. By incorporating advanced functionalities such as real-time monitoring of system processes, network activity, and device access, it provides a comprehensive solution for maintaining system integrity and security. The **current implementation** already covers essential features such as **process monitoring**, **file scanning**, **device control**, and **IP threat analysis**, ensuring a well-rounded approach to system management.

Looking to the future, the integration of **machine learning** for **malicious process detection**, coupled with the adoption of **cloud storage** for scalability and data accessibility, significantly enhances the system's capability to adapt to evolving security threats. The addition of **real-time alerts** and a **user-friendly GUI** further strengthens the system's usability, making it a proactive and intelligent security tool.

With a well-defined **development roadmap** and clear timelines, the project is set to evolve into a comprehensive, highly efficient, and future-proof solution suitable for both **individual users** and **enterprise-level organizations**. The continuous enhancement of its capabilities ensures it will remain at the forefront of system management, providing seamless security and monitoring in an increasingly complex digital environment.

In summary, the **Intelligent Task Manager** stands out as an exemplary project in the realm of **system management and security**, offering both immediate value and long-term scalability through its planned features and forward-thinking approach.