Wendigo Wraith Technical Report  
Introduction

|  |  |
| --- | --- |
| **Team Member** | **Role** |
| Gunel Salamova | Team-Lead |
| Elnur Huseynov | Front-end Developer |
| Fagan Abdullayev | Back-end Developer |
| Huseyn Hasanli | Security Configuration Engineer |
| Arif Gafarov | Security Architect |
| Royal Asgarov | Report Coordinator |

Table of Contents

[Summary 3](#_Toc202444631)

[Infrastructure 3](#_Toc202444632)

[Core Functions 4](#_Toc202444633)

[Dashboard Features 4](#_Toc202444634)

[Machines table 5](#_Toc202444635)

[Tools Table 5](#_Toc202444636)

[Wazuh Installation 5](#_Toc202444637)

[Sysmon Configuration 6](#_Toc202444638)

[Suricata Configuration 7](#_Toc202444639)

[Installation Process 7](#_Toc202444640)

[Network Interface Configuration 7](#_Toc202444641)

[Rule Update and Testing 7](#_Toc202444642)

[Suricata Logs 8](#_Toc202444643)

[Wazuh Integration with Suricata 8](#_Toc202444644)

[Custom-made Dashboard 9](#_Toc202444645)

[Embedded AI Incident Analysis 11](#_Toc202444646)

[AI integration workflow 12](#_Toc202444647)

[Live Terminal 12](#_Toc202444648)

[Security Architecture 12](#_Toc202444649)

[Key-based authentication 12](#_Toc202444650)

[Firewall enforcement 13](#_Toc202444651)

[Deployment considerations 13](#_Toc202444652)

[Backend implementation 13](#_Toc202444653)

[Frontend implementation 13](#_Toc202444654)

[Conclusion 14](#_Toc202444655)

[Endpoint Isolation 15](#_Toc202444656)

[File Fetching / Virus Total Analysis 16](#_Toc202444657)

[Workflow 16](#_Toc202444658)

# Summary

This capstone project was initiated on **June 23, 2025, at 9:00 AM**, led by **Arif Gafarov**, and carried out by team members **Elnur Huseynov**, **Huseyn Hasanli**, **Faqan Abdullayev**, **Gunel Salamova**, and **Royal Asgarov**. The project involved the creation of a fully custom **Security Operations Center (SOC) dashboard**, built from the ground up and tightly integrated with **Wazuh**.

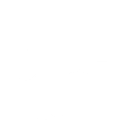
Unlike generic log collectors, the dashboard is engineered to ingest **only those logs that are triggered by custom-defined rules** within Wazuh. This selective log intake ensures focus on high-fidelity alerts and critical incidents.

An embedded AI module analyzes each triggered event in real time, generating comprehensive incident summaries along with intelligent **mitigation recommendations**. The system utilizes data from **Sysmon**, **Suricata**, and endpoint **Event Viewer logs**, effectively functioning as a lightweight **SIEM** tailored for rapid threat detection and decision support.

# Infrastructure

A diagram of a computer process

AI-generated content may be incorrect.

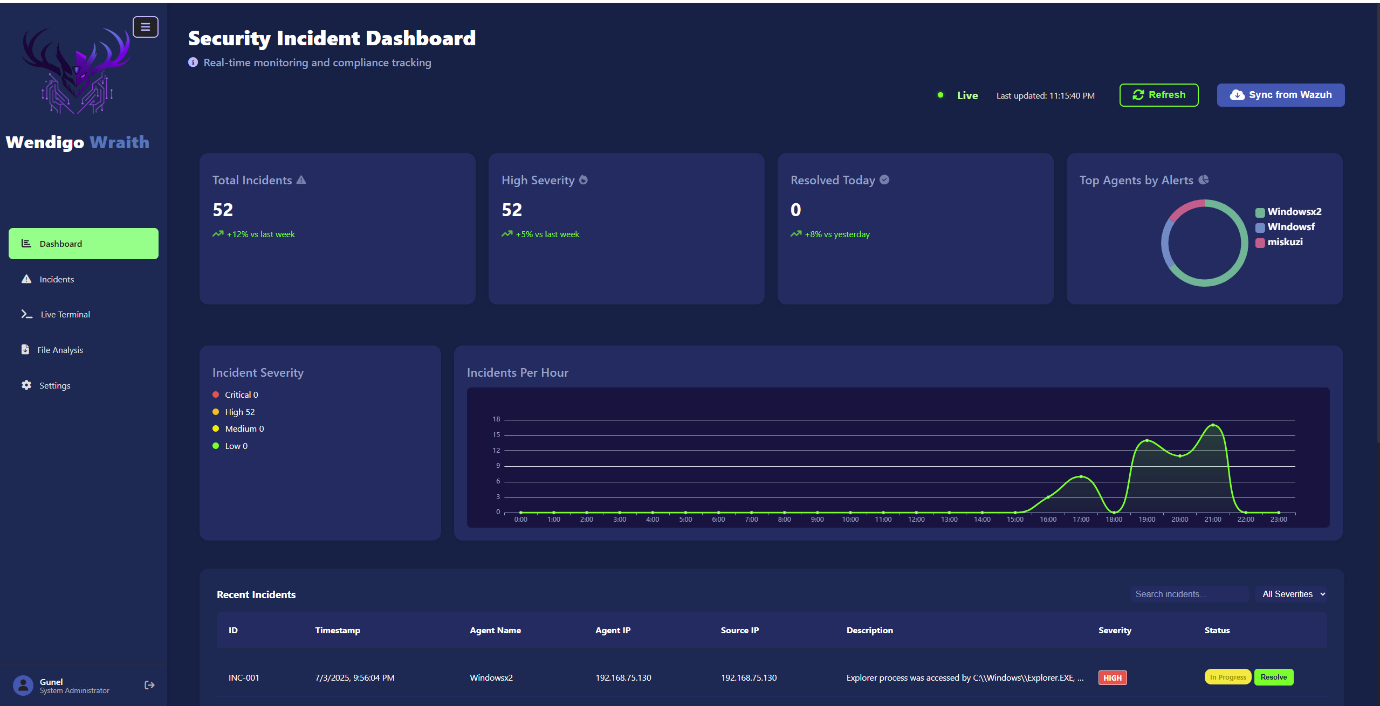


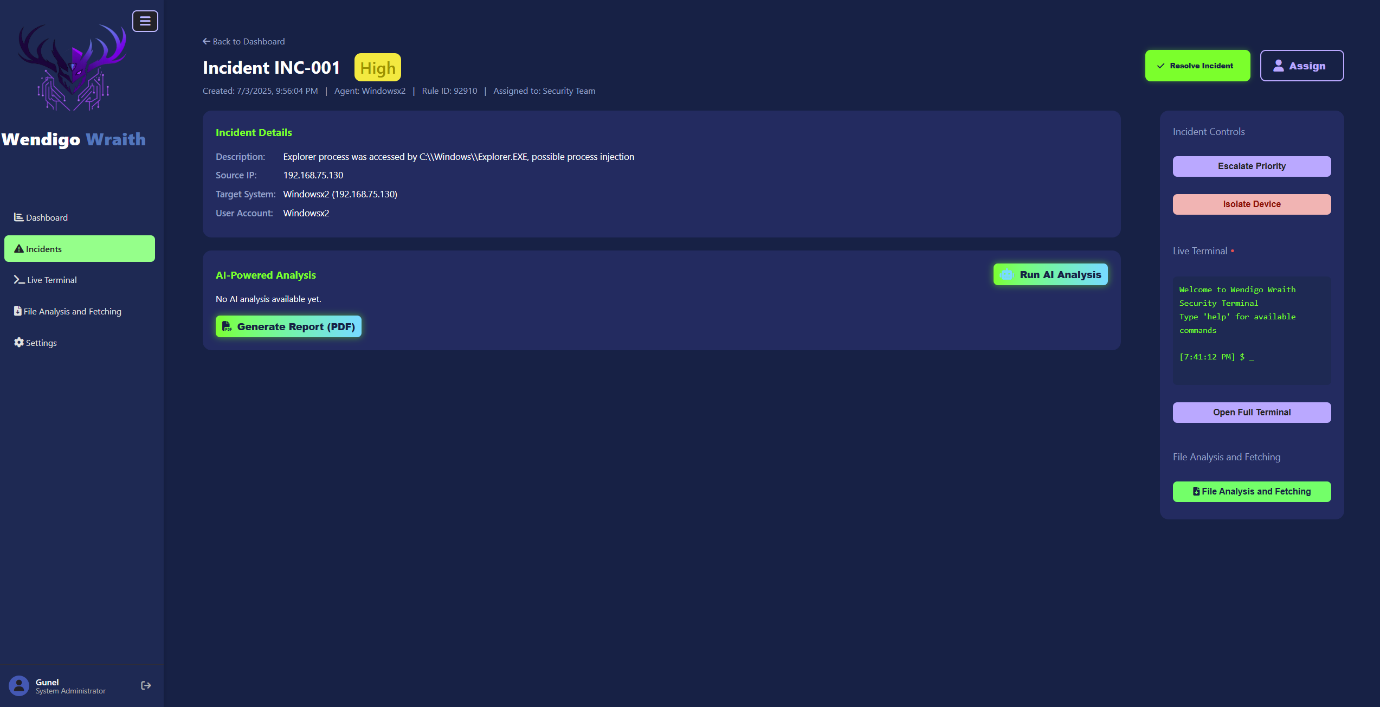
# Core Functions

* **Live Terminal (SSH)**: Analysts can open secure, real-time shell sessions to remote agents directly from the dashboard.
* **Endpoint Isolation**: Admins can remotely isolate compromised endpoints to prevent lateral movement.
* **Automated Report Generation**: Incidents are summarized and exported as structured reports (PDF/HTML).
* **Report Dispatch via Gmail**: Reports are emailed to designated recipients using Gmail’s secure API.
* **Discord Alert Notifications**: Any new incident triggers an instant alert to the team’s Discord server.
* **Embedded AI Analysis**: Each incident is passed through an integrated AI module (e.g., GPT-4), which:
  + Summarizes the incident
  + Highlights potential impact
  + Suggests mitigation/remediation steps

# Dashboard Features

* **Real-time statistics**: Total incidents, high severity count, resolved incidents, compliance rate
* **Incident breakdown by severity**: Critical, High, Medium, Low
* **Graphical insight**: Incidents per hour visualization
* **Recent incidents table**: Including ID, timestamp, agent name/IP, source IP, description, and severity badge
* **Sidebar navigation**: Dashboard, Incidents, Live Terminal, Settings

**Dashboard**

**Incidents**

# Machines table

|  |  |  |
| --- | --- | --- |
| **Component** | **OS version** | **Role / IP** |
| Ubuntu Server | 22.04.5 LTS | Wazuh Manager 192.168.75.131 |
| Windows | 10.0.19045 | End-point 192.168.75.129 |
| Windows | 10.0.19045 | End-point 192.168.75.130 |
| Ubuntu Desktop | 24.04.2 LTS | Suricata Integration 192.168.75.132 |

# Tools Table

|  |  |  |
| --- | --- | --- |
| **Tool** | **Version** | **Purpose** |
| Suricata | 7.0.10 | Network-based Intrusion Detection (NIDS) |
| Sysmon | 4.12.0 | System activity logging for endpoint visibility |
| Wazuh Agent | 4.12.0 | Endpoint security and log forwarding |
| Python | 3.12.3 | Backend scripting, AI integration, file automation |
| JavaScript | ES2024 | Dynamic client-side logic and interactivity |
| HTML | HTML5 | Structural layout for the dashboard frontend |
| CSS | CSS 3.4.1 | Styling with a modern, responsive design system |
| VSC | 1.91.0 | Code editor for front-end /back-end |

# Wazuh Installation

First install Ubuntu server 22.04.5 LTS. Run the following commands:   
curl -sO https://packages.wazuh.com/4.12/wazuh-install.sh

sudo bash wazuh-install.sh -a  
After installation, the services were started and enabled to run on boot. Basic firewall rules were configured to allow traffic on ports 1514/udp, 1515/tcp, and 55000/tcp for agent communication and the RESTful API.   
Install Wazuh agents of the same version to the endpoints to get logs .

# Sysmon Configuration

The following instructions must be done to install and send logs of Sysmon to Wazuh . The links to visit ( for installation medias ) :  
<https://learn.microsoft.com/en-us/sysinternals/downloads/sysmon>  
<https://github.com/OTRF/Blacksmith/blob/master/resources/configs/sysmon/sysmon.xml>  
Download both of medias and make a new folder . Unzip the Sysmon zip and add the xml file to the same folder . Run the command prompt as an administrator and run : Sysmon64.exe -accepteula -I Sysmon.xml

It will install Sysmon as a service on the system . Run sc query Sysmon64 to check if it is working . Do the same process for every agent in order to get Sysmon logs . The rest of the job will be done at Ubuntu server . Instead of configuring every machine to send Sysmon logs to Wazuh we will make a new configuration file( for a group ) and will add machines to that group . In this project the configuration is made on “default” group but on demand the configurations can be made on new folders . Following commands are demo :  
touch /var/ossec/etc/shared/default/agent.conf  
chown wazuh:wazuh /var/ossec/etc/shared/default/agent.conf  
chmod 660 /var/ossec/etc/shared/default/agent.conf  
Now for Wazuh to see Sysmon logs run “nano agent.conf”  
The following lines should be added to agent.conf in between:

<localfile>

<location>Microsoft-Windows-Sysmon/Operational</location>

<log\_format>eventchannel</log\_format>

</localfile>

<https://documentation.wazuh.com/current/user-manual/reference/centralized-configuration.html>  
Above link is the source of all information used for configuring Sysmon . Restart Wazuh manger and Wazuh agent and you will get all the Sysmon logs .

# Suricata Configuration

## Installation Process

Suricata was installed on an Ubuntu 24.04.2 LTS machine using the official OISF binary packages, ensuring stability and access to the latest features. Before installation, the following PPA repository was added to enable access to Suricata's maintained releases:

sudo add-apt-repository ppa:oisf/suricata-stable

This step was essential to ensure compatibility with Ubuntu's APT system and to receive updates directly from OISF. After updating the package index and installing Suricata, the service was verified and stopped temporarily to allow for safe configuration without service interruptions.

## Network Interface Configuration

Post-installation, the Suricata configuration file (/etc/suricata/suricata.yaml) was edited using a text editor. The network interface was manually defined to match the device’s primary monitoring segment—ensuring Suricata captures traffic from the correct interface. This step is crucial when Suricata operates on a span port or inline gateway.

## Rule Update and Testing

After Suricata was installed and its configuration completed, the next crucial step was to ensure that the system had the latest and most effective detection rules. The built-in Suricata update utility was used to retrieve the latest rules from trusted sources. This process ensures that the system is equipped to detect newly emerging threats and vulnerabilities.

Once the base rules were updated, a list of additional available rule sources was reviewed. These sources contain optional rule sets contributed by the community or maintained by third-party security organizations. Based on the needs of the environment, specific sources were selectively enabled to expand the rule base beyond the default sets.

Following the rule updates, Suricata’s configuration was rigorously tested using its validation functionality. This test confirmed that the syntax of the configuration file and all rule sets was correct, and that there were no structural issues that could prevent Suricata from starting or processing network traffic effectively.

This step was critical to ensuring stability and security. Validating configurations before deployment reduces downtime and ensures accurate detection once Suricata is actively monitoring network traffic.

## Suricata Logs

Once the service was running, logs were verified at:

/var/log/suricata/fast.log

/var/log/suricata/eve.json

The eve.json file was particularly important for integration with Wazuh, as it is JSON-formatted and suitable for structured ingestion.

## Wazuh Integration with Suricata

The Wazuh agent was installed on the same machine where Suricata is running. This allows Suricata alerts to be picked up and forwarded to the Wazuh Manager for centralized analysis and response.

To configure the Wazuh agent to monitor Suricata logs, the following block was added to the /var/ossec/etc/ossec.conf file:

<localfile>

<log\_format>json</log\_format>

<location>/var/log/suricata/eve.json</location>

</localfile>

This configuration ensures that Suricata's alerts (in JSON format) are continuously monitored and forwarded by the Wazuh agent and it also implemented to manager configuration of wazuh for receiving alerts from Suricata.

# Custom-made Dashboard

Imported libraries:

* requests: For HTTP(S) communication with Elasticsearch.
* mysql.connector: For connecting and interacting with MySQL.
* datetime: For parsing and converting timestamps.
* urllib3: To suppress SSL warnings during insecure HTTPS connections.

Constants and Settings:

* ELASTIC\_URL, ELASTIC\_USER, ELASTIC\_PASS: Elasticsearch API endpoint and credentials.
* DB\_CONFIG: MySQL database connection parameters.
* AZERBAIJAN\_TZ: Defines the target timezone (UTC+4) for timestamp conversion.

Wazuh connectivity test:  
Function: test\_wazuh\_connection()

* **Purpose:** Ensures that the Wazuh Elasticsearch API is reachable before querying it.
* **Behavior:** Makes a simple GET request to the Elasticsearch root URL.
* **Output:** Logs the status or error, and returns a boolean result (True or False)

Alert fetching from Elasticsearch:  
Function: fetch\_alerts()

* **Pre-check:** Calls test\_wazuh\_connection() to verify API reachability.
* **Query:** Sends a POST request with a JSON payload to filter alerts by rule.id = 111111.
* **Output:** Returns a list of alert records (Elasticsearch \_source documents).
* **Error Handling:** Catches authentication errors, timeouts, or JSON parsing issues.

Ff

Inserting alerts to MySQL   
Function: insert\_to\_mysql(alerts)

* **Database Connection:** Opens a MySQL connection using mysql.connector.
* **Table Creation:** Ensures that a table named alerts exists using CREATE TABLE IF NOT EXISTS.
* Extracts fields like timestamp, rule.id, rule.description, agent.name, etc.
* Parses and converts the alert timestamp from UTC to Azerbaijan tim e.
* Inserts the alert using INSERT IGNORE to prevent duplication based on alert ID.
* **Commit:** Changes are saved using db.commit(), and connections are closed gracefully.

Main function

* Calls fetch\_alerts() to get relevant alerts.
* If alerts are returned, it passes them to insert\_to\_mysql(alerts).
* If no alerts are found or connection fails, it exits with an appropriate message.

Summary of the dataflow

A diagram of a computer program

AI-generated content may be incorrect.

Multi-Factor Authentication

This feature enhances identity verification by requiring users to provide two distinct forms of authentication: something they *know* (username and password), and something they *have* (a time-sensitive MFA code).This layered security reduces the risk of unauthorized access due to stolen credentials and aligns the platform with best practices in modern cybersecurity.

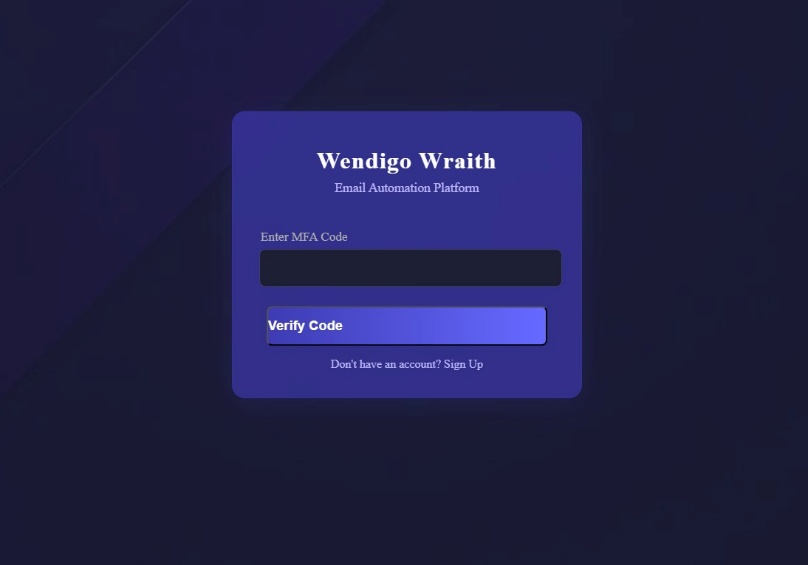
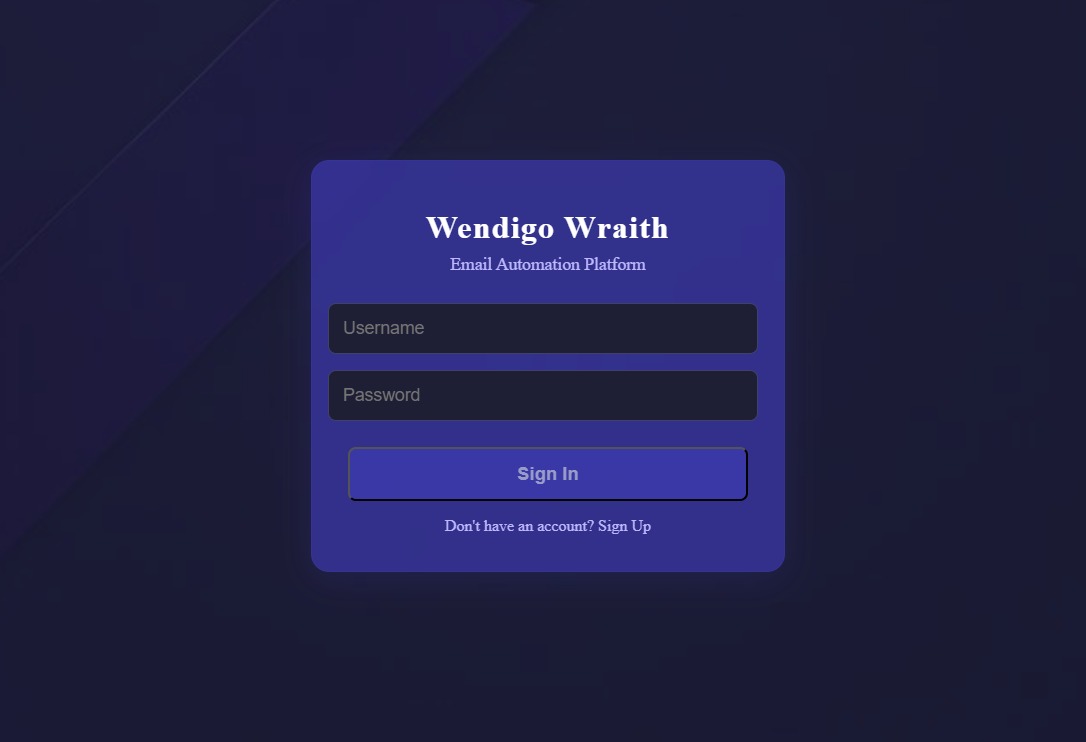
Initial login

* ser Input: Username and password.
* Interface: A login page with a clean, modern UI and clearly marked input fields.
* Validation: Backend verifies credentials against the user database.

MFA

* User Input: MFA code (typically generated by a mobile authenticator app or sent via email/SMS).
* Interface: A second-stage page requesting the code.
* Code Validity**:**

Uses TOTP with 30 seconds validity



# Embedded AI Incident Analysis

* **Predefined Prompt Engineering**:  
  A carefully crafted prompt is designed in advance. This prompt instructs the AI on how to interpret and format incident logs.
* **Live Log Injection**:  
  As each alert is received into the dashboard, it is **automatically sent to the ChatGPT API** using the defined prompt.
* **Standardized AI Output**:  
  The AI returns a **template-based response** which includes:

**IOC (Indicators of Compromise)**

**Mitigation Recommendations**

**Remediation Steps**

**Risk Assessment**

**Summary of the Incident**

* **Dashboard Integration**:  
  The AI-generated insight is immediately attached to the alert record and displayed in the dashboard for the SOC analyst to review or act upon.

Within all those functions the result will be:

* Converting raw logs into actionable intelligence in real-time
* Accelerating incident triage and response
* Ensuring consistency in assessment and remediation documentation
* Enhancing decision-making for junior analysts through expert-level summaries

# 

# AI integration workflow

1. **When a new alert is triggered in Wazuh**, it is immediately forwarded to the custom-built SOC dashboard.
2. The dashboard extracts relevant metadata from the alert such as the agent name, source IP address, timestamp, rule ID, and event description.
3. This structured log data is **combined with a predefined prompt** and sent to the **ChatGPT API** using a secure HTTP request.
4. ChatGPT processes the input using its natural language understanding and returns a **standardized incident summary**, formatted according to the structure defined in the prompt.
5. The AI-generated summary is then **attached to the corresponding incident card** in the dashboard interface, making it instantly visible to SOC analysts for review and action.

# Live Terminal

This system was built **from the ground up** using the **SSH protocol** to ensure secure, authenticated, and encrypted command execution.

## Security Architecture

* Each agent endpoint has a **dedicated admin user account**.
* All admin users are configured with **strong, unique passwords**—high-entropy and designed to resist brute-force attempts.
* **Password-based SSH login is disabled** on all agents.
* All connections are authenticated solely via **public key cryptography**.



## Key-based authentication

* A .ssh directory exists on each agent under the admin user's home folder.
* The same public key is stored in ~/.ssh/authorized\_keys on all agents.
* The dashboard machine holds the matching private key, securely stored and used by our backend to initiate connections.

## Firewall enforcement

* Each agent has strict firewall rules configured (e.g., ufw/iptables) to only allow incoming SSH connections from the dashboard’s IP address.
* Any unauthorized SSH attempt is blocked and logged for SIEM correlation.

# Deployment considerations

* Our proof-of-concept model initially used Active Directory for centralized identity and key distribution.
* While beneficial for scalability and automation, this was not included in the final project due to environment constraints.
* All user accounts and keys were **manually provisioned** to ensure complete visibility and control.
* This decision increased complexity but ensured **maximum security isolation** and simplified debugging.

## Backend implementation

The Python backend leverages the paramiko library to:

* Establish SSH sessions.
* Authenticate using the private key.
* Spawn interactive terminal shells.
* Stream STDIN/STDOUT between the agent and the dashboard.

## Frontend implementation

* The live terminal interface is built using modern web technologies.
* It mimics a full-featured TTY shell with:

1. Input echo
2. Scrollback buffer
3. Neon-green retro theme for cyber-ops aesthetics
4. The interface is click-to-connect: the SOC analyst selects an agent IP and is immediately dropped into a shell—no passwords, no prompts**.**

|  |  |
| --- | --- |
| Measure | Status |
| |  | | --- | | Password SSH login |  |  | | --- | |  | | Disabled |
| |  | | --- | | Key-only authentication |  |  | | --- | |  | | Enforced |
| |  | | --- | | Firewall restrictions |  |  | | --- | |  | | |  | | --- | | Agent-specific |  |  | | --- | |  | |
| |  | | --- | | Private key exposure |  |  | | --- | |  | | |  | | --- | | Dashboard only |  |  | | --- | |  | |
| |  | | --- | | Logging unauthorized access |  |  | | --- | |  | | Enabled |
| |  | | --- | | Brute-force resistance |  |  | | --- | |  | | Very high |

Conclusion

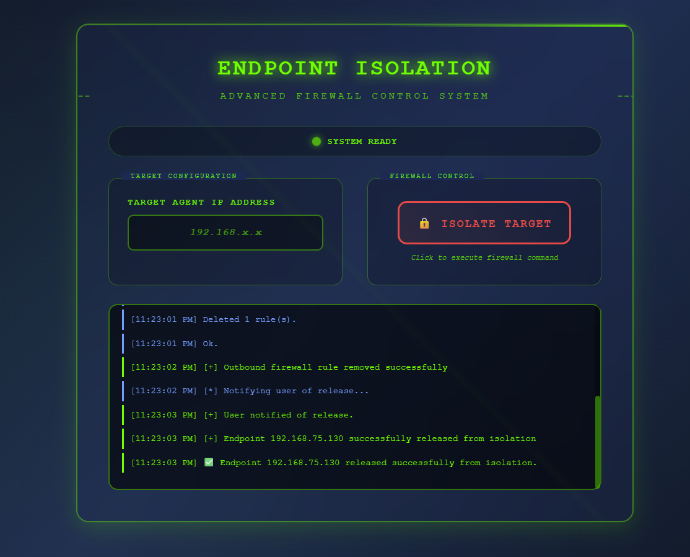
The live terminal feature offers zero-friction, secure remote access for SOC analysts without compromising endpoint security. By disabling password authentication and using public key cryptography and network-layer controls, we’ve created a hardened yet highly usable remote management capability.This live terminal is a cornerstone feature of Wendigo Wraith, enabling real-time response to incidents, malware detonation, and remote forensics at analyst fingertips.

# Endpoint Isolation

The **Endpoint Isolation** feature provides a powerful mechanism for remote containment of compromised or suspicious systems directly from the SOC dashboard. This functionality is critical during incident response to prevent lateral movement, data exfiltration, or communication with external threat actors.

* Upon selecting an endpoint in the dashboard, an **SSH connection** is established from the dashboard server to the target agent.
* A **PowerShell script** is then remotely deployed and executed on the agent machine. This script modifies the **Microsoft Defender Firewall rules** to:

1. Block **all inbound and outbound network traffic**, effectively cutting the device off from the network.
2. Allow **only the SSH session** with the dashboard to remain active, ensuring SOC analysts retain remote control.



Key Features of Isolation

* Analysts can **isolate** a device with a single click.
* Analysts can also **restore network access** to previously isolated agents using the same interface.
* After isolation, the agent device automatically displays a **pop-up alert** to the user:   
  "Your device has been isolated from the network. Please contact your system administrator."
* Isolation commands are executed under a privileged account.
* Password authentication is disabled across all agents; only **SSH key-based authentication** is permitted.
* Each agent has a local “.ssh/authorized\_keys” file pre-configured with the dashboard's public key, while the dashboard uses the matching private key.

# File Fetching / Virus Total Analysis

In our SOC dashboard, certain incidents are generated based on custom detection rules which may involve suspicious files or executables. To enhance our incident triage and response capabilities, we implemented a feature named

**“File Analysis and Fetching”**. This feature is available for incidents that include file path indicators, and it automates the process of validating a file’s threat level and fetching it securely for further analysis.

## Workflow

1. **Triggering the process**

When an analyst clicks on an incident that contains a suspicious file path, the dashboard interface displays a button labelled “File Analysis and Fetching”.

1. **SSH-Based Remote Access**

Upon clicking the button, the backend (written in Python using paramiko) initiates an SSH connection to the corresponding agent where the file resides.This connection is secure and uses pre-configured public-key authentication. Password authentication is disabled on all agents, and SSH access is restricted via firewall to only allow dashboard-originated connections.

1. **Hash Extraction**

Once connected, the system locates the file using the given path from the incident description. It then calculates the file's cryptographic hash (e.g., SHA-256) using command-line utilities on the agent machine.

1. **VirusTotal integration**

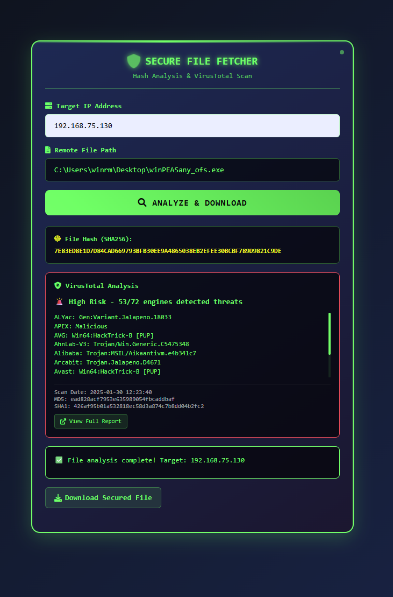
The computed hash is sent to **VirusTotal** via their public API using a predefined API key. If the hash exists in the VirusTotal database, a concise result is returned and displayed on the dashboard. This result includes:

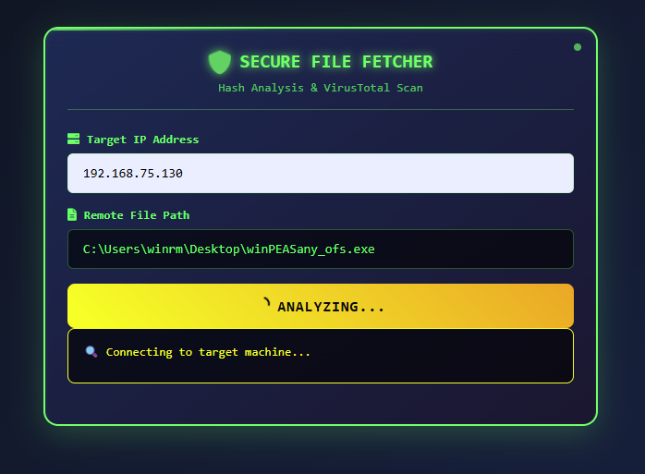
* Whether the file is considered malicious or clean.

1. A direct link to the full VirusTotal analysis page for further review  
     
     
   **Secure File Retrieval**

If the analyst chooses to fetch the file, it is automatically:

* Zipped (compressed) to minimize risk of accidental execution or spreading.
* Transferred over the existing SSH connection and stored in the dashboard’s local ./download directory.
* Logged for auditing and traceability**.**





1. **Offline Reverse-engineering**

* Analysts can later use local tools (e.g., **Ghidra**, **xxd**, **ExifTool**) within a sandbox environment to manually inspect the downloaded file.
* This offline approach ensures no additional risk is introduced to the live network environment.

Benefits:

1. **Rapid Threat Assessment:** Hash lookup provides near-instant insight without needing full file upload.
2. **Operational Safety:** Files are transferred securely and zipped to reduce execution risk.
3. **Actionable Intelligence:** Enables deeper offline reverse engineering and reporting.

# Final Conclusion

The Wendigo Wraith project represents a comprehensive, custom-built Security Operations Center (SOC) solution, merging open-source tools, real-time threat monitoring, and AI-powered incident response into a single platform. Our team successfully integrated Wazuh, Sysmon, Suricata, and a custom-built React/Tailwind dashboard to deliver a feature-rich interface for modern security operations.

Key accomplishments :

- Real-time alert ingestion and AI-assisted analysis

- Secure live terminal access for rapid response

- One-click endpoint isolation and file fetching

- Seamless integration with VirusTotal and Discord/Teams for notifications

This project not only demonstrates technical proficiency in cybersecurity and full-stack development, but also showcases our ability to collaborate and deliver an operational tool that mirrors enterprise-grade SOC workflows. With modular design and scalability in mind, Wendigo Wraith serves as a strong foundation for further enhancements, such as machine learning analytics and broader endpoint compatibility. The result is a responsive, secure, and intelligent SOC dashboard built for the evolving cybersecurity landscape.

# Literature Review and Reference to Sources Used

Literature Review and Reference to Sources Used  
  
Multi-Factor Authentication (MFA) is a well-established security mechanism that enhances traditional password-based authentication by requiring additional verification factors. According to NIST SP 800-63B, MFA is considered essential for reducing the risk of unauthorized access due to compromised credentials. Numerous studies and implementations have demonstrated that adding a second authentication factor—typically something the user has or is—greatly increases the difficulty for attackers attempting to breach user accounts.  
  
The implementation model followed in this project was influenced by widely adopted techniques in web security practices, especially those outlined in:  
- OWASP Authentication Cheat Sheet (OWASP, 2023): Emphasizes defense against brute-force attacks and session hijacking by implementing TOTP-based MFA.  
- RFC 6238 (IETF): Provides the technical foundation for Time-Based One-Time Passwords (TOTP), used in applications like Google Authenticator.  
- Course materials from "Web Application Security" and "Secure Software Development" (2024 modules, university-provided).  
- Various Flask and Django MFA implementation guides available through platforms such as GeeksForGeeks, Stack Overflow, and MDN Web Docs.  
  
During the research, existing authentication flows in enterprise-grade platforms like Microsoft 365 and AWS Cognito were studied to understand how MFA is integrated into real-world applications. This led to the choice of combining username/password (knowledge factor) with TOTP/OTP code (possession factor) in the Wendigo Wraith platform.

# Project Plan Reflection

Project Plan Reflection  
  
The original project plan involved three main phases:  
1. Research and Design  
2. Development and Integration  
3. Testing and Finalization  
  
The MFA module was initially scheduled for completion by Week 4. However, integration challenges, especially around secure session token handling and synchronizing frontend validation with backend OTP expiration logic, caused slight delays. These were not fully anticipated during planning.  
  
Time management was overall decent, but more buffer time should have been allocated for UI/UX refinements and testing edge cases (e.g., expired tokens, concurrent sessions). One key realization was the importance of early-stage prototyping — had the login and MFA flows been mocked earlier, many frontend integration issues could have been identified sooner.  
  
What could be improved:  
- Better estimation of development vs. testing workload  
- Earlier focus on user experience (e.g., handling error messages gracefully)  
- Automating test cases for MFA code validation to save manual time

# Personal Reflection

Personal Reflection  
  
Working on the MFA module for the Wendigo Wraith platform significantly enhanced my understanding of real-world security practices. Implementing a secure login system was not just about writing forms—it involved thinking like an attacker, securing endpoints, validating tokens, handling race conditions, and managing user session states.  
  
From a technical standpoint, I learned:  
- The complexity behind seemingly simple authentication flows  
- How to use tools like Flask, JWT, bcrypt, and TOTP libraries  
- How front-end and back-end must cooperate precisely in security-critical features  
  
I also gained an appreciation for security-through-usability — for example, creating a clean, distraction-free interface for MFA entry that doesn’t frustrate the user but also doesn’t compromise security.  
  
In the future, I would:  
- Use a test-driven approach earlier in development  
- Implement rate-limiting and alerting for repeated MFA failures  
- Research more on biometric or push-notification-based MFA systems for even higher usability and security  
  
Overall, the project bridged theoretical knowledge with real-world implementation, giving me hands-on skills in secure development practices and authentication systems.