**Logo, company name

Description automatically generated**

**COMSATS University Islamabad (CUI)**

Software Design Description   
(SDS DOCUMENT)

for

**AEGIS**

**(Real-time Malware Detection Prevention and Patching)**

Version 1.0

***By***

**Muhammad Faizan Haider CIIT/SP22-BCT-030/ISB**

**Muhammad Abdullah CIIT/SP22-BCT-026/ISB**

**Salman Ahmed CIIT/SP22-BCT-042/ISB**

***Supervisor*Dr. Farhana Jabeen**

*Bachelor of Science in Cybersecurity (2022-2026)*

**Table of Contents**

[1. Introduction 1](#_Toc199328390)

[1.1 1.3 Modules 1](#_Toc199328391)

[1.1.1 1.3.1 System Call Monitoring Module 1](#_Toc199328392)

[1.1.2 1.3.2 Behavioural Analysis & Malware Detection Module 2](#_Toc199328393)

[1.1.3 1.3.3 Malware Intent Classification Module 2](#_Toc199328394)

[1.1.4 1.3.4 Process Containment & Prevention Module 2](#_Toc199328395)

[1.1.5 1.3.5 Malicious Code Extraction Module 3](#_Toc199328396)

[1.1.6 1.3.6 LLM-Based Code Repair Module 3](#_Toc199328397)

[1.1.7 1.3.7 Code Formatting & Optimization Module 3](#_Toc199328398)

[1.1.8 1.3.8 Secure Code Execution Module 3](#_Toc199328399)

[1.1.9 1.3.9 Malware Signature Generation Module 3](#_Toc199328400)

[1.1.10 1.3.10 Logging & Reporting Module 4](#_Toc199328401)

[1.1.11 1.3.11 Visualization & Dashboard Module 4](#_Toc199328402)

[1.1.12 1.3.12 API & Communication Module 4](#_Toc199328403)

[1.1.13 1.3.13 Training Dataset Preparation Module 4](#_Toc199328404)

[1.1.14 1.3.14 Fine-Tuning the LLM for Code Repair Module 4](#_Toc199328405)

[2. Design Methodology and Software Process Model 5](#_Toc199328406)

[2.1.1 Domain Representation 5](#_Toc199328407)

[2.1.2 Complex Behavior Encapsulation 5](#_Toc199328408)

[2.1.3 Extensibility Requirements 5](#_Toc199328409)

[2.1.4 Modularity and Separation of Concerns 5](#_Toc199328410)

[2.1.5 Reusability of Components 6](#_Toc199328411)

[2.2 Process Model: Incremental and Iterative Development 6](#_Toc199328412)

[2.2.1 Risk Mitigation 6](#_Toc199328413)

[2.2.2 Complexity Management 6](#_Toc199328414)

[2.2.3 Feedback Incorporation 6](#_Toc199328415)

[2.2.4 Adaptability to Changing Threats 7](#_Toc199328416)

[2.2.5 Early Delivery of Core Value 7](#_Toc199328417)

[3. System Overview 7](#_Toc199328418)

[3.1 Box and Line: 7](#_Toc199328419)

[3.2 Architectural Design: 8](#_Toc199328420)

[4. Design Models 9](#_Toc199328421)

[4.1 Activity Diagram: 9](#_Toc199328422)

[4.1.1 Visualization and Dashboard: 9](#_Toc199328423)

[4.1.2 System call monitoring: 10](#_Toc199328424)

[*4.1.3* Filter and Analyze System Calls 11](#_Toc199328425)

[4.1.4 Behavioral Analysis: 12](#_Toc199328426)

[4.1.5 Malware Intent Classification: 13](#_Toc199328427)

[4.1.6 Halt Malicious Process Execution 14](#_Toc199328428)

[*4.1.7* Log Containment Event 15](#_Toc199328429)

[*4.1.8* Extract Malicious Code from Halted Process 16](#_Toc199328430)

[*4.1.9* Identify Malicious Sections of Malicious Code 17](#_Toc199328431)

[4.1.10 Send Extracted Code to LLM for Automated Patch Generation 18](#_Toc199328432)

[*4.1.11* Convert Patched Code into a Ready-to-Run Executable Format 19](#_Toc199328433)

[*4.1.12* Clean and Structure Dataset Entries for LLM Training 20](#_Toc199328434)

[*4.1.13* Fine-Tune Model 21](#_Toc199328435)

[4.2 Class diagram 22](#_Toc199328436)

[4.3 Sequence diagram: 23](#_Toc199328437)

[4.3.1 System Call Monitoring: 23](#_Toc199328438)

[4.3.2 Behavioral Analysis & Detection: 24](#_Toc199328439)

[4.3.3 Process Containment 25](#_Toc199328440)

[4.3.4 Code Extraction 25](#_Toc199328441)

[4.3.5 LLM-Based Code Repair: 26](#_Toc199328442)

[4.3.6 Visualization and Dashboard 27](#_Toc199328443)

[4.3.7 LLM Fine-Tuning: 28](#_Toc199328444)

[4.3.8 Logging and Reporting 29](#_Toc199328445)

[4.3.9 API integration 30](#_Toc199328446)

[4.3.10 Dashboard updating 31](#_Toc199328447)

[4.4 Behavioral State Diagram 31](#_Toc199328448)

[4.4.1 Overall System 31](#_Toc199328449)

[4.4.2 HMM based Anomaly Detection 32](#_Toc199328450)

[4.4.3 LLM Patch Generation 32](#_Toc199328451)

[4.4.4 Process Resurrection 32](#_Toc199328452)

[4.4.5 API Handling 33](#_Toc199328453)

[5. Data Design 33](#_Toc199328454)

[5.1 Data Dictionary 33](#_Toc199328455)

[Functions 36](#_Toc199328456)

[6. Human Interface Design 38](#_Toc199328457)

[6.1 Screen Images 38](#_Toc199328458)

[6.1.1 Dashboards 40](#_Toc199328459)

[6.1.2 Home Page 41](#_Toc199328460)

[6.1.3 About Page 42](#_Toc199328461)

[7. Implementation 43](#_Toc199328462)

[7.1 Algorithms 43](#_Toc199328463)

[7.2 External API/SDKs 51](#_Toc199328464)

[7.3 User Interface 52](#_Toc199328465)

[7.3.1 Screen Images 52](#_Toc199328466)

[7.3.2 Dashboards 54](#_Toc199328467)

[7.3.3 Home Page 55](#_Toc199328468)

[7.3.4 About Page 56](#_Toc199328469)

[7.4 Deployment 57](#_Toc199328470)

[8. Testing and Evaluation 61](#_Toc199328471)

[8.1 Unit Testing 61](#_Toc199328472)

[8.2 Functional Testing 63](#_Toc199328473)

[8.3Integration Testing 65](#_Toc199328474)

[9. References 66](#_Toc199328475)

[10. Plagiarism Report 68](#_Toc199328476)

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason for changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

Application Evaluation History

|  |  |
| --- | --- |
| **Comments (by committee)**  **\*include the ones given at scope time both in doc and presentation** | **Action Taken** |
|  |  |
|  |  |

Supervised by

Dr. Farhana Jabeen

Signature\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Introduction

The Malware Detection, Prevention, and Automated Code Repair System is poised to offer an active real-time cybersecurity solution that utilizes artificial intelligence-driven threat detection, automated containment, and self-healing. The system will constantly monitor system calls, detect malicious behavior through behavioral analysis, and label threats according to severity. Upon detection of a high-risk process, the system will suspend execution, eliminate the malicious code, and forward it to a specially trained large language model for automated vulnerability patching. After generation of the patched code, the system will authenticate and safely apply it to overwrite the vulnerable process, guaranteeing uninterrupted recovery.

The project will have essential security modules like system call monitoring, behavior analysis, process containment, LLM-based patching, malware signature generation, and safe code execution. The system will also have advanced logging, reporting, and visualization, giving security analysts in-depth forensic information on threats found and response taken. There will also be integration via a RESTful API to third-party security products like SIEMs, IDS/IPS, and threat intelligence systems.

However, the project will not feature such functions as endpoint protection (such as classic antivirus capabilities) or human inspection of malware because it primarily deals with automated detection, containment, and repair of code. The platform will first be implemented for Linux-based systems with enhancements in the future targeted towards Windows and cloud-centric deployments.

By malware response and patching automation, this project will minimize security incident response times, lower the need for human effort, and enhance overall cybersecurity resilience against zero-day attacks and advanced threats.

## 1.3 Modules

### 1.3.1 System Call Monitoring Module

* ***FE-1:*** *Captures all system calls made by running processes in real-time.*
* ***FE-2:*** *Logs syscall metadata, including process ID, timestamps, and arguments.*
* ***FE-3:*** *Sends detected syscall sequences to the behavioral analysis module.*
* ***FE-4****: Preprocesses syscall sequences before feeding them into the for anomaly detection. HMM model*
* ***UI-FE-1:*** *Provides a log viewer for system administrators to analyse syscall activity.*
* ***SEC-FE-1:*** *Ensures syscall logs are encrypted and accessible only to authorized users.*
* ***PERF-FE-1:*** *Monitors system calls with minimal overhead to avoid system slowdowns*

### 1.3.2 Behavioural Analysis & Malware Detection Module

* ***FE-1:*** *Analyzes system call patterns using rule-based and ML-based anomaly detection.*
* ***FE-2:*** *Flags unusual behaviors such as file encryption, unauthorized privilege escalation, or network access.*
* ***FE-3:*** *Classifies detected malware based on predefined behavior patterns.*
* ***FE-4****: Uses an HMM-based probabilistic model to calculate the likelihood of syscall sequences being malicious.*
* ***FE-5****: Maintains a database of HMM-trained normal and malicious syscall sequences for comparison.*
* ***UI-FE-1:*** *Displays detection results in a graphical dashboard with risk indicators.*
* ***SEC-FE-1:*** *Prevents unauthorized modification of detection rules.*
* ***PERF-FE-1:*** *Ensures real-time detection without excessive CPU/memory usage.*

### 1.3.3 Malware Intent Classification Module

* ***FE-1:*** *Identifies the intent of malicious activities based on syscall analysis and execution patterns.*
* ***FE-2:*** *Categorizes detected threats into ransomware, spyware, rootkits, or privilege escalation attempts.*
* ***FE-3:*** *Assigns a risk level (low, medium, high) to each detected malware based on HMM anomaly scores.*
* ***UI-FE-1:*** *Provides an interactive interface for administrators to view malware classification results.*
* ***SEC-FE-1:*** *Restricts modification of malware classification results to authorized personnel.*
* ***PERF-FE-1:*** *Ensures rapid classification within milliseconds of detection.*

### 1.3.4 Process Containment & Prevention Module

* ***FE-1:*** *Immediately halts execution of flagged malware processes.*
* ***FE-2:*** *Applies system-level security restrictions to prevent further malicious actions.*
* ***FE-3:*** *Logs the containment event for forensic analysis and investigation.*
* ***UI-FE-1:*** *Displays a containment notification with detailed process information.*
* ***SEC-FE-1:*** *Ensures that once a process is contained, it cannot restart automatically.*
* ***PERF-FE-1:*** *Enforces containment measures without affecting benign processes.*

### 1.3.5 Malicious Code Extraction Module

* ***FE-1:*** *Extracts the relevant portion of malicious code from the halted process.*
* ***FE-2:*** *Formats the extracted code into a structured representation for LLM analysis.*
* ***FE-3:*** *Identifies the malicious sections of the code that need patching.*
* ***UI-FE-1:*** *Displays extracted code snippets for review by security analysts.*
* ***PERF-FE-1:*** *Optimizes code extraction speed to avoid system delays.*

### 1.3.6 LLM-Based Code Repair Module

* ***FE-1:*** *Sends the extracted vulnerable code to a fine-tuned Code Llama model, while a secondary LLM ChatGPT provides contextual CWE knowledge for enhanced patch generation.*
* ***FE-2:*** *Replaces malicious sections with benign code.*
* ***FE-3:*** *Outputs a patched version of the code, ready for execution.*
* ***UI-FE-1:*** *Provides an interface for users to manually review LLM-generated patches.*
* ***SEC-FE-1:*** *Prevents unauthorized modifications to generated patches before execution.*

### 1.3.7 Code Formatting & Optimization Module

* ***FE-1:*** *Ensures that the patched code maintains proper syntax and structure.*
* ***FE-2:*** *Improves readability and optimizes execution performance.*
* ***FE-3:*** *Converts patched code into a ready-to-run executable format.*
* ***UI-FE-1:*** *Displays formatted code with syntax highlighting for user review.*
* ***PERF-FE-1:*** *Ensures that formatting does not introduce unnecessary overhead****.***

### 1.3.8 Secure Code Execution Module

* ***FE-1:*** *Monitors execution to detect unexpected behavior or failures.*
* ***FE-2:*** *Logs execution results for future analysis and debugging.*
* ***UI-FE-1:*** *Provides execution logs.*
* ***PERF-FE-1:*** *Minimizes execution delays while maintaining security*.

### 1.3.9 Malware Signature Generation Module

* ***FE-1:*** *Extracts key characteristics of detected malware to generate a unique signature.*
* ***FE-2:*** *Stores generated signatures in a local database for future reference.*
* ***FE-3:*** *Uses stored signatures to enhance future detection capabilities.*
* ***UI-FE-1:*** *Displays newly generated signatures for manual review.*
* ***SEC-FE-1:*** *Prevents unauthorized access to the malware signature database.*
* ***PERF-FE-1:*** *Optimizes signature matching for real-time detection.*

### 1.3.10 Logging & Reporting Module

* ***FE-1:*** *Logs all detected malware events, containment actions, and patching results.*
* ***FE-2:*** *Generates security reports summarizing system threats and responses.*
* ***FE-3:*** *Allows exporting reports in various formats for forensic analysis.*
* ***UI-FE-1:*** *Provides interactive filtering and search options for logs.*
* ***SEC-FE-1:*** *Ensures logs are immutable and cannot be modified by unauthorized users.*
* ***PERF-FE-1:*** *Compresses log data for efficient storage.*

### 1.3.11 Visualization & Dashboard Module

* ***FE-1:*** *Provides an interactive dashboard displaying malware detection statistics.*
* ***FE-2:*** *Visualizes system call trends, detected threats, and patching success rates.*
* ***FE-3:*** *Allows administrators to filter and analyse threat data over time.*
* ***UI-FE-1:*** *Supports customizable dashboard layouts and views.*
* ***SEC-FE-1:*** *Restricts access to dashboard data based on user roles.*
* ***PERF-FE-1:*** *Optimizes real-time data updates without performance lag.*

### 1.3.12 API & Communication Module

* ***FE-1:*** *Provides RESTful API access for communication between agent and LLM model.*
* ***FE-2:*** *Enables automated integration with SIEM tools.*
* ***UI-FE-1:*** *Displays API usage metrics and logs.*
* ***SEC-FE-1:*** *Requires authentication and authorization for API access.*
* ***PERF-FE-1:*** *Ensures low-latency API responses.*

### 1.3.13 Training Dataset Preparation Module

* ***FE-1:*** *Collects real-world vulnerability and malware samples.*
* ***FE-2:*** *Cleans and structures dataset entries for HMM training.*
* ***FE-3:*** *Ensures no overlap between training and test datasets.*

### 1.3.14 Fine-Tuning the LLM for Code Repair Module

* ***FE-1:*** *Fine-tunes a Transformer-based model (CodeT5, Mistral, CodeLlama) using novel collaborative approach, where a secondary LLM provides CWE-based vulnerability insights.*
* ***FE-2:*** *Uses QLoRA for efficient training.*
* ***FE-3:*** *Continuously updates the model by integrating new vulnerabilities (from CVE databases, exploit reports, etc.) into its training pipeline.*

# Design Methodology and Software Process Model

For this malware detection and protection system, I've applied an Object-Oriented Programming (OOP) methodology, which is well-justified for several reasons:

### Domain Representation

**Justification:** The malware detection system deals with real-world entities that naturally map to objects such as system calls, processes, malware signatures, and behaviors. OOP allows us to model these domain entities directly as classes, making the system design intuitive and semantically accurate.

For example, SyscallEvent, MalwareSignature, and ProcessContainer are direct representations of real-world concepts in the security domain.

### Complex Behavior Encapsulation

**Justification:** The system requires complex algorithms (HMM for anomaly detection, LLM for code repair) that need to maintain state and exhibit sophisticated behaviors. OOP's encapsulation allows us to hide implementation details while exposing only necessary interfaces.

The HiddenMarkovModel and CodeRepairLLM classes encapsulate complex mathematical and AI algorithms while providing simple interfaces like analyzeSyscallSequence() and generatePatch().

### Extensibility Requirements

**Justification:** Malware evolves constantly, requiring the system to adapt to new threats. OOP's inheritance and polymorphism facilitate extending the system by adding new malware categories, detection rules, and containment strategies without modifying existing code.

The inheritance hierarchies for DetectionRule, MalwareCategory, and ContainmentStrategy demonstrate this extensibility.

### Modularity and Separation of Concerns

**Justification:** The system has distinct responsibilities (monitoring, analysis, containment, repair) that benefit from strong separation of concerns. OOP enables us to create classes with single responsibilities that collaborate through well-defined interfaces.

Each class in the design handles one primary responsibility, like SystemCallMonitor focusing on syscall capture or MaliciousCodeExtractor specializing in code extraction.

### Reusability of Components

**Justification:** Many components in the system (like logging, dashboard visualization, API management) can be reused across various security applications. OOP's class-based structure promotes reusable components through inheritance and composition.

Components like SecurityLogger, DashboardManager, and APIManager can be reused in other security applications with minimal modification.

## Process Model: Incremental and Iterative Development

For the development process, I'm following an **Incremental and Iterative Development** model, which is well-suited for this security application for the following reasons:

### Risk Mitigation

**Justification:** Security systems involve high risks of vulnerabilities or missed detection capabilities. The iterative approach allows early testing of critical components like the detection engine before the entire system is built.

The core components (syscall monitoring, behavioral analysis) can be developed and tested first before adding advanced features like code repair.

### Complexity Management

**Justification:** The system involves complex algorithms (HMM, LLM) and integrations. Building incrementally allows the team to manage this complexity by focusing on smaller, functional pieces before integrating them.

Each module can be developed independently and integrated progressively, starting with system call monitoring, then adding analysis, classification, etc.

### Feedback Incorporation

**Justification:** Security effectiveness depends on continuous feedback about detection accuracy. The iterative model facilitates incorporating feedback from testing into subsequent iterations, improving the system's detection capabilities.

Detection rules and model parameters can be refined based on test results from each iteration.

### Adaptability to Changing Threats

**Justification:** The threat landscape evolves rapidly, and requirements may change during development. The iterative model accommodates changing requirements better than waterfall approaches.

New malware types discovered during development can be incorporated into the classification system in subsequent iterations.

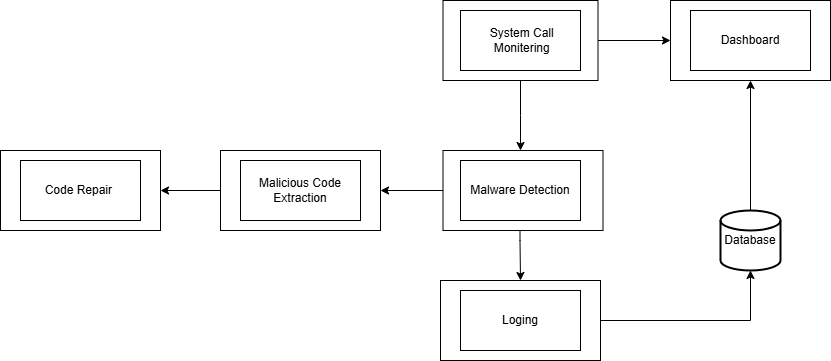
### Early Delivery of Core Value

**Justification:** Basic protection is valuable even before all advanced features are complete. The incremental approach delivers core detection capabilities early, providing value while more sophisticated features (like automatic code repair) are developed.

A minimal viable product with syscall monitoring and basic detection could be deployed early, with code repair capabilities added later.

# System Overview

## Box and Line:



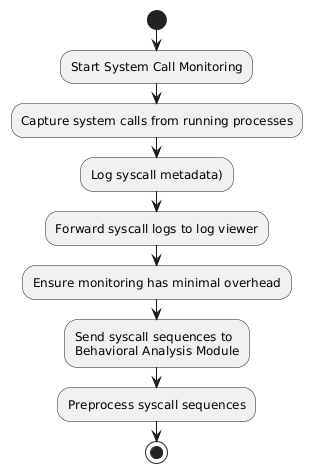
## A close-up of a document AI-generated content may be incorrect.Architectural Design:

# Design Models

## Activity Diagram:

### A diagram of a system AI-generated content may be incorrect.Visualization and Dashboard:

### System call monitoring:

****

### Filter and Analyze System Calls

A diagram of a system

AI-generated content may be incorrect.

### Behavioral Analysis:

**A flowchart of a computer

AI-generated content may be incorrect.**

### Malware Intent Classification:

A diagram of a process flow

AI-generated content may be incorrect.

### A diagram of malware process AI-generated content may be incorrect.Halt Malicious Process Execution

### Log Containment Event

A diagram of a process

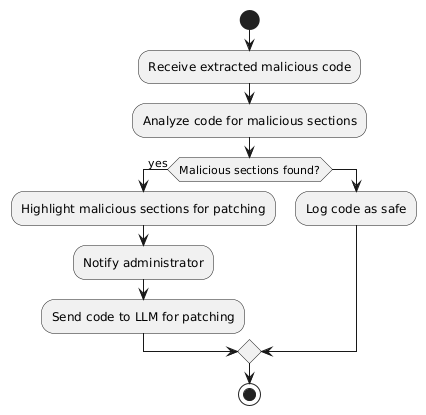
AI-generated content may be incorrect.

### Extract Malicious Code from Halted Process

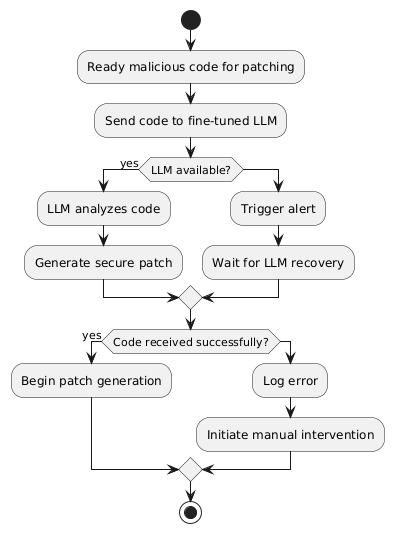
A diagram of a process

AI-generated content may be incorrect.

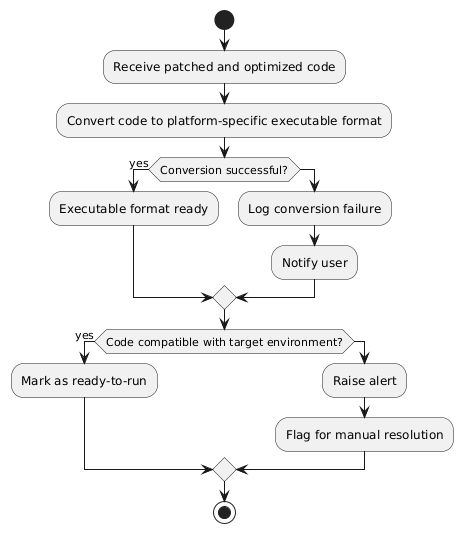
### Identify Malicious Sections of Malicious Code

****

### Send Extracted Code to LLM for Automated Patch Generation



### Convert Patched Code into a Ready-to-Run Executable Format



### Clean and Structure Dataset Entries for LLM Training

A diagram of a data processing process

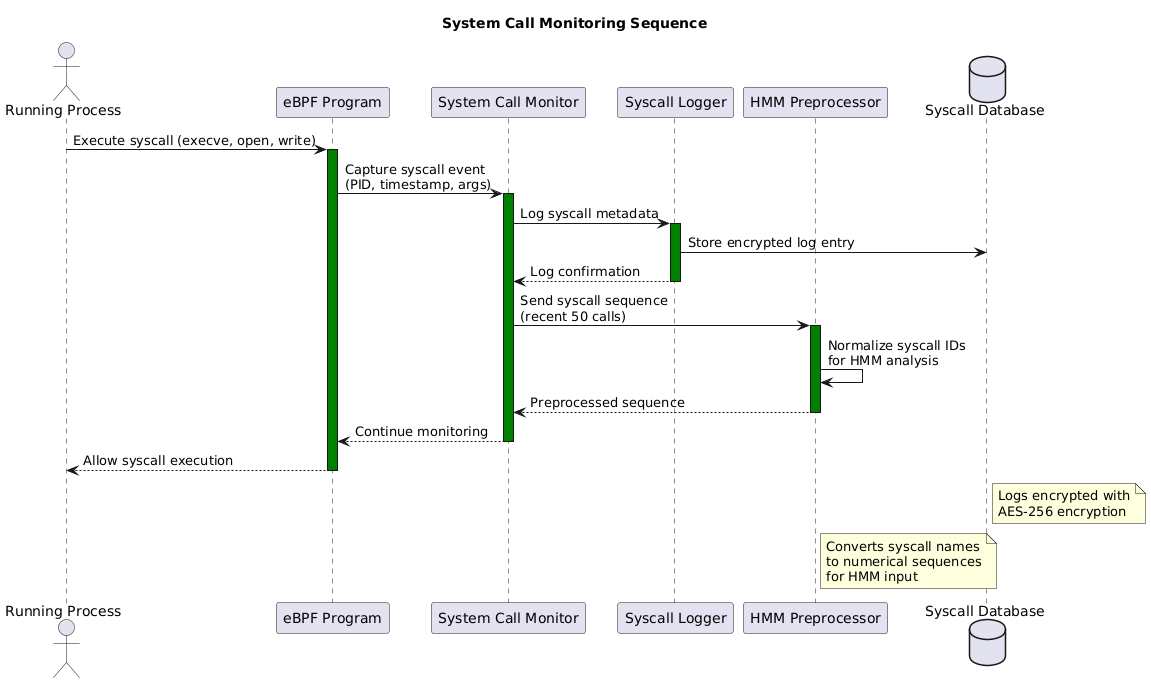
AI-generated content may be incorrect.

### A diagram of a software process AI-generated content may be incorrect.Fine-Tune Model

## Class diagram

## Sequence diagram:

### System Call Monitoring:

****

### Behavioral Analysis & Detection:

A screenshot of a computer screen

AI-generated content may be incorrect.

### Process Containment

A screenshot of a project management system

AI-generated content may be incorrect.

### A diagram of a program AI-generated content may be incorrect.Code Extraction

### LLM-Based Code Repair:

**A screenshot of a computer

AI-generated content may be incorrect.**

### Visualization and Dashboard

A diagram of a data processing process

AI-generated content may be incorrect.

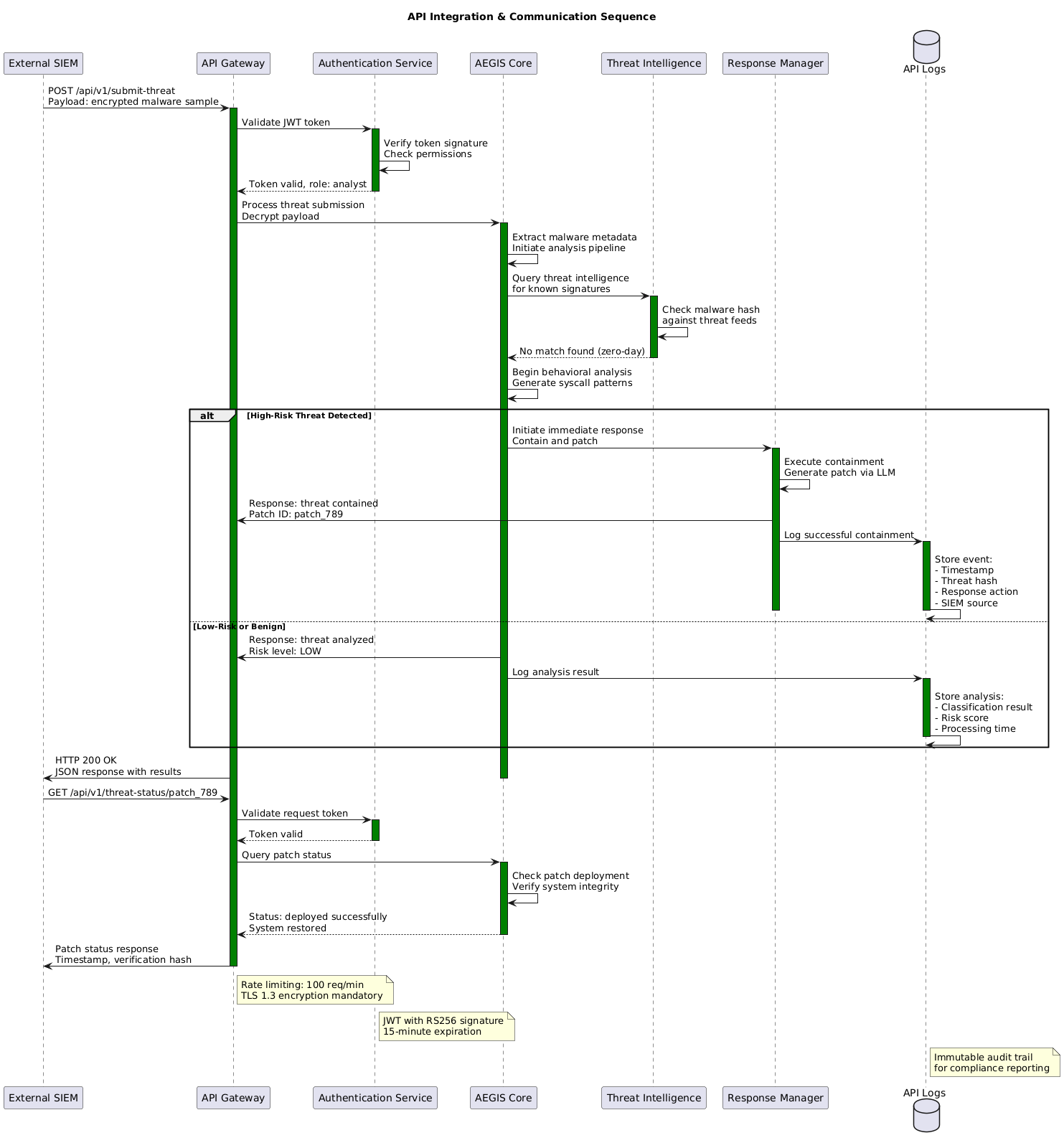
### A screenshot of a computer AI-generated content may be incorrect.LLM Fine-Tuning:

### Logging and Reporting

**A diagram of a computer program

AI-generated content may be incorrect.**

### API integration



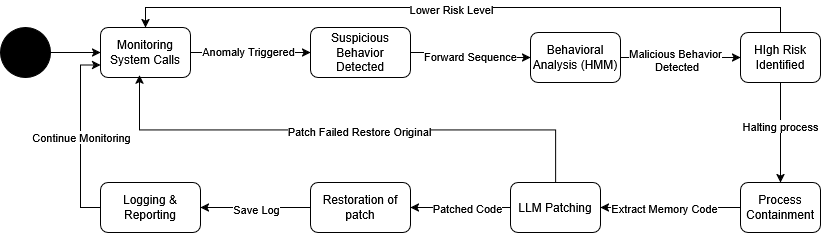
### Dashboard updating

A diagram of a software program

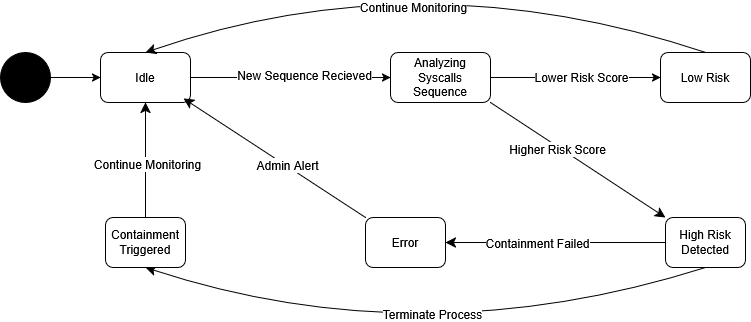
AI-generated content may be incorrect.

## Behavioral State Diagram

### Overall System



### HMM based Anomaly Detection



### LLM Patch Generation

A black background with white squares

AI-generated content may be incorrect.

### A black and white screen with white text AI-generated content may be incorrect.Process Resurrection

### API Handling

A black screen with white text

AI-generated content may be incorrect.

# Data Design

## Data Dictionary

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity/Function** | **Attribute/Parameter** | **Type** | **Description** | **Methods** | **Method Parameters** | **Return Type** |
| User | username | String | Unique identifier for the user. | login() | username (String), password (String) | Boolean (True if successful) |
|  | password | String | Secure password for authentication. | logout() | None | Void |
|  | role | String | Role of user (e.g., Admin, Analyst). | updatePassword() | old\_password (String), new\_password (String) | Boolean |
|  | last\_login | DateTime | Timestamp of last login. |  |  |  |
| SystemCallSequence | syscall\_id | String | Unique identifier for the syscall log. | captureCall() | process\_id (String), syscall\_name (String), args (List) | Boolean |
|  | process\_id | String | ID of the process making the syscall. | filterByPattern() | pattern (Regex String) | List (Filtered calls) |
|  | timestamp | DateTime | Time when the syscall occurred. | sendToAnalyzer() | syscall\_id (String) | Boolean |
|  | arguments | List | Arguments passed to the syscall. |  |  |  |
| ProcessClassification | classification\_id | String | Unique ID for a classification event. | classifyBehavior() | syscall\_sequence (List) | String (e.g., Ransomware) |
|  | process\_id | String | Process associated with the behavior. | assignRiskScore() | classification\_id (String) | Integer (0–100) |
|  | behavior\_type | String | Type: ransomware, rootkit, spyware, etc. |  |  |  |
|  | risk\_score | Integer | Severity score of threat. |  |  |  |
| MaliciousCode | code\_id | String | Unique ID for malicious code block. | extractCode() | process\_id (String) | String (Malicious snippet) |
|  | process\_id | String | Source process of the malicious code. | displayExtractedCode() | code\_id (String) | String (Formatted code) |
|  | code\_snippet | String | Extracted suspicious code. | identifyMaliciousSections() | code\_snippet (String) | List (Line numbers, patterns) |
| BenignPatch | patch\_id | String | Unique ID for the generated patch. | generatePatch() | malicious\_code (String) | String (Patched code) |
|  | source\_code\_id | String | Refers to associated malicious code. | validatePatch() | patch (String) | Boolean |
|  | patched\_code | String | Cleaned and safe version of original code. | deployPatch() | patch\_id (String) | Boolean |
| API | endpoint\_id | String | Unique identifier for API endpoint. | fetchThreatData() | auth\_token (String), filters (Dict) | JSON |
|  | route | String | URI path (e.g., /submit-patch). | sendPatchToLLM() | code\_id (String), model (String) | String (Response status) |
|  | method | String | Request method: GET, POST, PUT, DELETE. | logAPICall() | endpoint\_id (String), status\_code (Int) | Void |
|  | request\_payload | JSON | Data sent to the endpoint. |  |  |  |

## Functions

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Parameters** | **Description** | **Return Type** |
| login() | username (String), password (String) | Authenticates the user based on provided credentials. | Boolean (True if successful) |
| logout() | None | Logs out the current user. | Void |
| updatePassword() | old\_password (String), new\_password (String) | Updates user password securely. | Boolean (True if successful) |
| captureCall() | process\_id (String), syscall\_name (String), args (List) | Captures a system call made by a process for monitoring. | Boolean |
| filterByPattern() | pattern (Regex String) | Filters system call logs based on a defined pattern. | List (Filtered calls) |
| sendToAnalyzer() | syscall\_id (String) | Sends syscall for anomaly analysis. | Boolean |
| classifyBehavior() | syscall\_sequence (List) | Analyzes syscall sequence to classify process behavior. | String (Threat type) |
| haltMaliciousProcess() | process\_id (String) | Halts a flagged malicious process to prevent further execution. | Boolean |
| extractCode() | process\_id (String) | Extracts malicious code snippet from the flagged process. | String (Code snippet) |
| identifyMaliciousSections() | code\_snippet (String) | Identifies malicious sections within extracted code. | List (Malicious patterns/lines) |
| generatePatch() | malicious\_code (String) | Sends malicious code to LLM to generate secure patch. | String (Patched code) |
| deployPatchIntoMemory() | patch\_id (String) | Deploys the validated patch directly into memory for immediate use. | Boolean |
| logAPICall() | endpoint\_id (String), status\_code (Int) | Logs usage and response of API calls. | Void |

# Human Interface Design

## Screen Images

Allows User to register new account or sign in with a new account in the system. Allows multiple

users to use the tool on the same system.

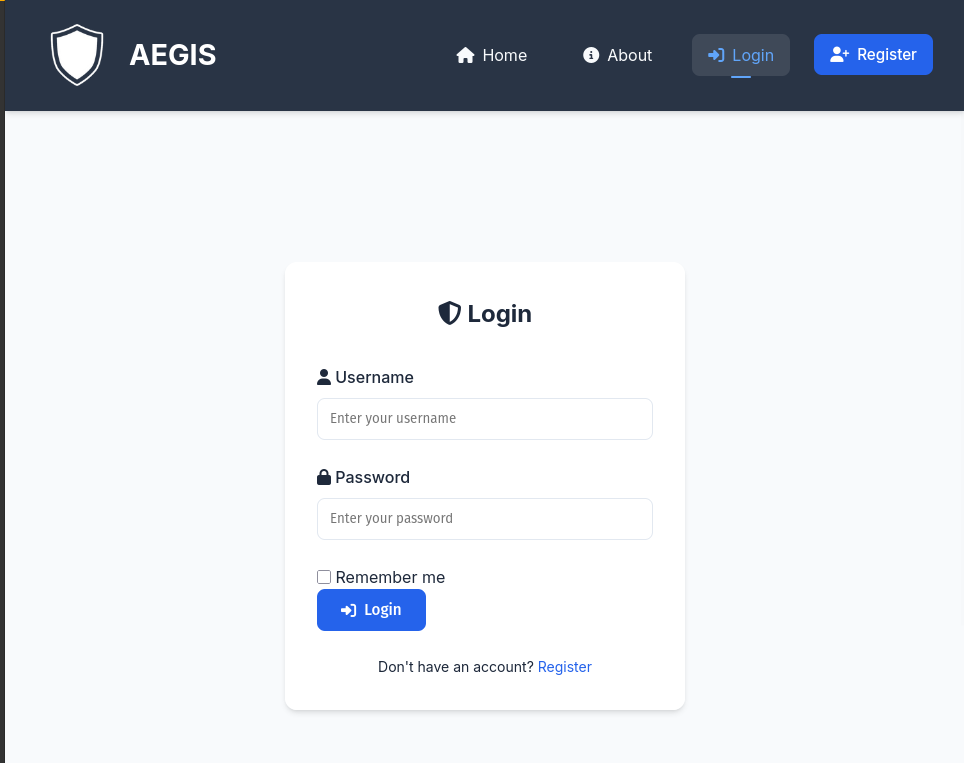


Figure Login Page

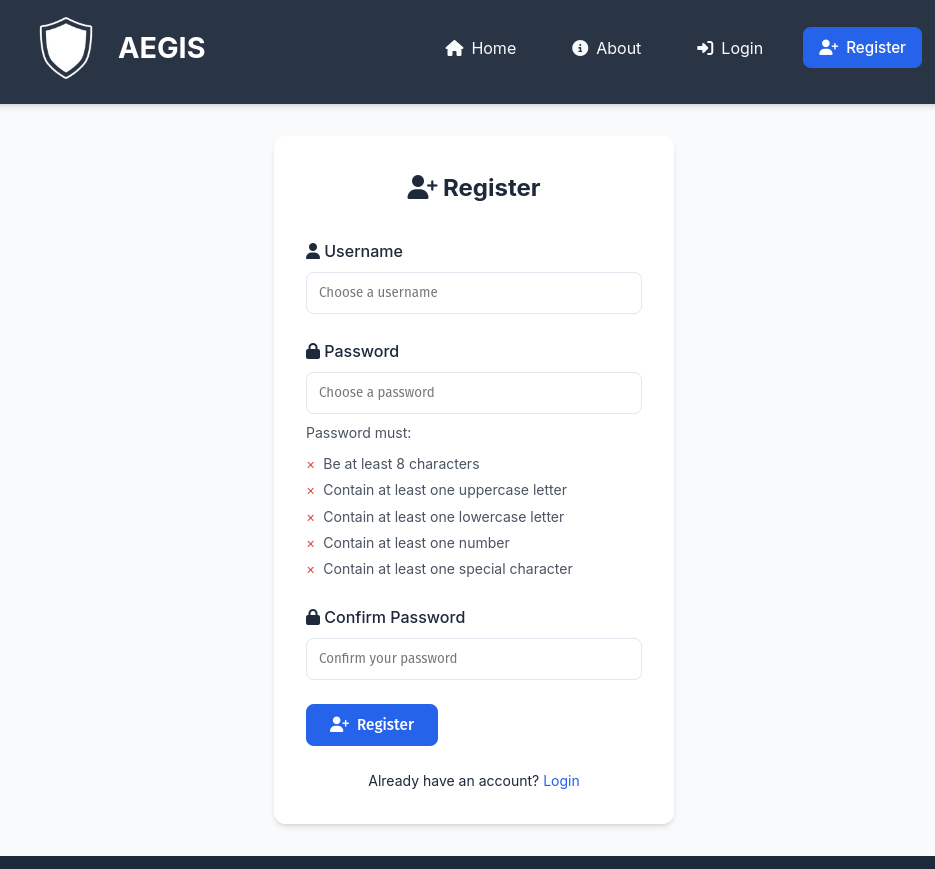


Figure Register Page

### Dashboards

Allows users to view recent activity, and review findings.

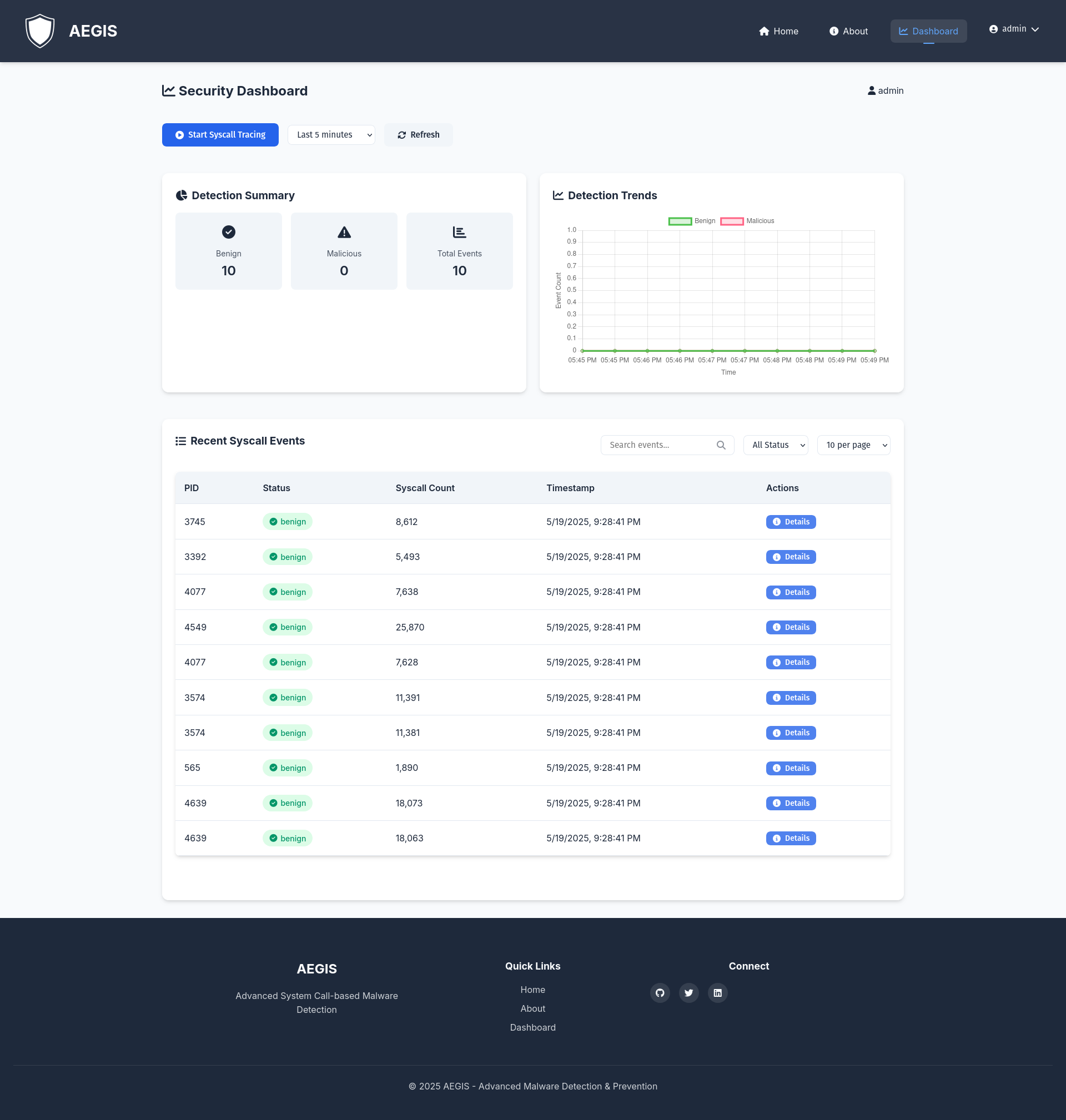


Figure Dashboard Page

### Home Page



Figure Home Page

### About Page

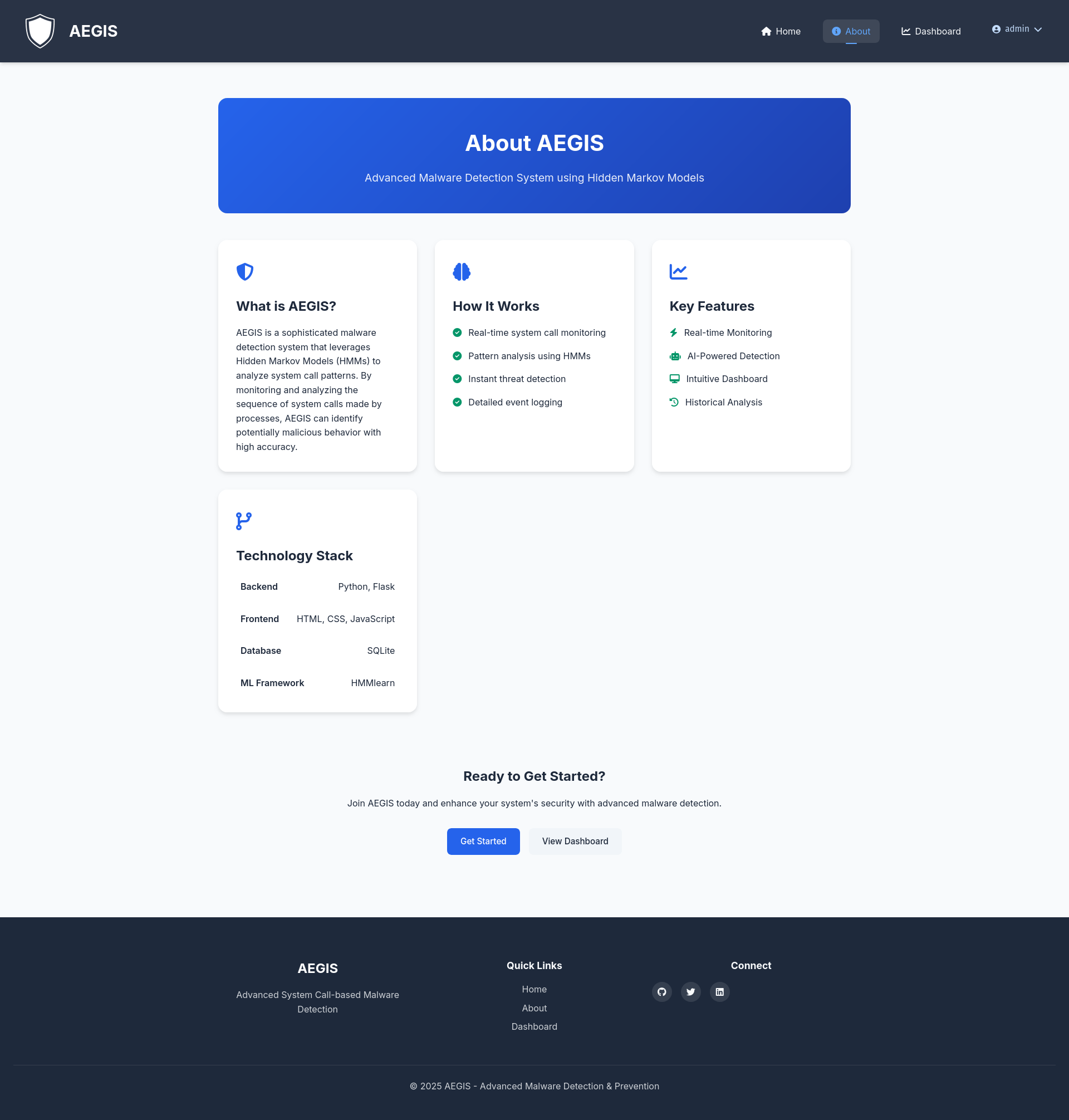


Figure About Page

# Implementation

## Algorithms

|  |
| --- |
| **Algorithm 1: Syscall Monitoring and Classification** |
| **Input: C/C++ source code file Output: Identified vulnerabilities and corresponding Address Sanitizer analysis** |
| **Output: System Call sequences based on corresponding PID** |
| Input: syscall\_mapping\_file, hmm\_model Output: process\_classifications  1: mapping\_df ← read\_csv(syscall\_mapping\_file) 2: name\_to\_id ← create\_dict(mapping\_df["Syscall Name"], mapping\_df["Syscall Number"]) 3: hmm\_benign, hmm\_malicious, classify\_sequence ← get\_hmm\_models() 4: bpf\_code ← build\_bpf\_code(syscall\_names) 5: function\_map ← create\_function\_mapping(syscall\_names) 6: bpf\_program ← compile\_bpf\_program(bpf\_code) 7: attach\_kprobes(bpf\_program, syscall\_names, function\_map) 8: syscall\_log ← initialize\_empty\_dict() 9: syscall\_ids\_log ← initialize\_empty\_dict() 10: start\_perf\_buffer\_monitoring() 11: return process\_classifications  **Function build\_bpf\_code(syscall\_names):** Input: syscall\_names, bpf\_header, bpf\_function\_template Output: final\_bpf\_code, function\_map 1: final\_bpf\_code ← bpf\_header 2: function\_map ← initialize\_empty\_dict() 3: for each syscall in syscall\_names: 4: func\_name ← "trace\_" + extract\_syscall\_name(syscall) 5: function\_map[syscall] ← func\_name 6: final\_bpf\_code ← final\_bpf\_code + format\_template(func\_name, syscall) 7: return final\_bpf\_code, function\_map  **Function print\_event(cpu, data, size):** Input: cpu, data, size Output: None 1: event ← extract\_event\_data(data) 2: pid ← event.pid 3: syscall ← decode\_syscall\_name(event.syscall) 4: syscall\_log[pid].append(syscall) 5: syscall\_id ← get\_syscall\_id(syscall, name\_to\_id) 6: syscall\_ids\_log[pid].append(syscall\_id) 7: write\_logs\_to\_files(syscall\_log, syscall\_ids\_log) 8: if sufficient\_syscalls\_for\_classification(syscall\_ids\_log[pid]): 9: recent\_seq ← get\_recent\_syscalls(syscall\_ids\_log[pid]) 10: prediction ← classify\_sequence(recent\_seq) 11: output\_classification\_result(pid, prediction) 12: return |
| **Algorithm 2: HMM Model for Syscall Classification** |
| **Input: Syscall sequences Output: Classification models and classification function** |
| **Output: Classification of sequence as benign or malicious** |
| **Function get\_hmm\_models():** Input: None Output: hmm\_benign, hmm\_malicious, classify\_function 1: hmm\_benign ← load\_trained\_hmm\_model("benign") 2: hmm\_malicious ← load\_trained\_hmm\_model("malicious") 3: classify\_function ← create\_classifier\_function(hmm\_benign, hmm\_malicious) 4: return hmm\_benign, hmm\_malicious, classify\_function  **Function classify\_sequence(sequence):** Input: sequence Output: classification 1: benign\_score ← hmm\_benign.score(sequence) 2: malicious\_score ← hmm\_malicious.score(sequence) 3: if malicious\_score > benign\_score: 4: return "malicious" 5: else: 6: return "benign" 7: return classification  **Function load\_trained\_hmm\_model(model\_type):** Input: model\_type Output: hmm\_model 1: if model\_type is "benign": 2: hmm\_model ← load\_model\_from\_file("benign\_model.pkl") 3: else if model\_type is "malicious": 4: hmm\_model ← load\_model\_from\_file("malicious\_model.pkl") 5: return hmm\_model |
| **Algorithm 3 Process Termination and Memory Extraction** |
| **Input: Process ID (PID)** |
| **Output: Memory dump file and terminated process** |
| Algorithm Input: pid Output: memory\_dump\_file  1: pid ← get\_process\_id\_from\_user() 2: output\_file ← generate\_output\_filename(pid) 3: success ← extract\_memory(pid, output\_file) 4: if success is true: 5: halt\_process(pid) 6: return output\_file  **Function halt\_process(pid):** Input: pid Output: termination\_success 1: try: 2: process ← get\_process\_object(pid) 3: process.terminate() 4: process.wait(timeout=3) 5: output\_success\_message(pid) 6: return true 7: except process\_not\_found\_exception: 8: output\_error\_message("no such process", pid) 9: except access\_denied\_exception: 10: output\_error\_message("access denied", pid) 11: except timeout\_exception: 12: output\_error\_message("timeout waiting for termination", pid) 13: return false  F**unction extract\_memory(pid, output\_file):** Input: pid, output\_file Output: extraction\_success 1: maps\_path ← "/proc/" + pid + "/maps" 2: mem\_path ← "/proc/" + pid + "/mem" 3: try: 4: maps\_file ← open(maps\_path, "read") 5: mem\_file ← open(mem\_path, "read\_binary") 6: out\_file ← open(output\_file, "write\_binary") 7: for each line in maps\_file: 8: parts ← split\_line\_by\_space(line) 9: addr ← parts[0] 10: perms ← parts[1] 11: if perms contains "r": 12: start, end ← parse\_address\_range(addr) 13: mem\_file.seek(start) 14: chunk ← mem\_file.read(end - start) 15: out\_file.write(chunk) 16: close\_all\_files(maps\_file, mem\_file, out\_file) 17: output\_success\_message(output\_file) 18: return true 19: except file\_not\_found\_exception: 20: output\_error\_message("proc files not found") 21: except permission\_error\_exception: 22: output\_error\_message("permission denied") 23: except other\_exception as e: 24: output\_error\_message(e) 25: return false |
| **Algorithm 4 Secure LLM Memory Patching** |
| **Input: Memory dump file and process ID** |
| **Output: Securely patched binary** |
| Algorithm Input: llm\_api\_url, encryption\_key Output: secure\_patch\_file  1: patcher ← initialize\_secure\_llm\_patcher(llm\_api\_url, encryption\_key) 2: try: 3: patch\_file ← patcher.send\_for\_patching(memory\_dump\_path, process\_id) 4: output\_success\_message(patch\_file) 5: except exception as e: 6: output\_error\_message(e) 7: return patch\_file  **Function initialize\_secure\_llm\_patcher(llm\_api\_url, encryption\_key):** Input: llm\_api\_url, encryption\_key Output: patcher 1: patcher.llm\_api\_url ← llm\_api\_url 2: patcher.cipher ← initialize\_fernet\_cipher(encryption\_key) 3: patcher.logger ← setup\_logger() 4: return patcher  **Function setup\_logger():** Input: None Output: logger 1: logger ← create\_logger('AEGIS\_Patcher') 2: logger.level ← INFO 3: handler ← create\_file\_handler('secure\_patching.log') 4: handler.formatter ← create\_formatter('timestamp - message') 5: logger.add\_handler(handler) 6: return logger  **Function send\_for\_patching(memory\_dump\_path, process\_id):** Input: memory\_dump\_path, process\_id Output: patch\_path 1: try: 2: raw\_data ← read\_binary\_file(memory\_dump\_path) 3: if size\_of(raw\_data) > 100MB: 4: raise value\_error("Memory dump too large") 5: encrypted\_data ← encrypt\_data(raw\_data) 6: checksum ← generate\_checksum(raw\_data) 7: response ← send\_secure\_api\_request(process\_id, checksum, encrypted\_data) 8: if response.status\_code equals 200: 9: decrypted\_patch ← decrypt\_data(response.content) 10: patch\_path ← generate\_patch\_filename(process\_id) 11: write\_secure\_file(patch\_path, decrypted\_patch) 12: log\_entry ← create\_success\_log\_entry(process\_id, checksum) 13: log\_encrypted\_message(log\_entry) 14: return patch\_path 15: else: 16: raise exception("LLM API Error: " + response.text) 17: except exception as e: 18: error\_msg ← create\_error\_message(process\_id, e) 19: log\_encrypted\_error(error\_msg) 20: raise exception 21: return null  **Function encrypt\_data(data):** Input: data Output: encrypted\_data 1: encrypted\_data ← cipher.encrypt(data) 2: return encrypted\_data  Function decrypt\_data(encrypted\_data): Input: encrypted\_data Output: decrypted\_data 1: decrypted\_data ← cipher.decrypt(encrypted\_data) 2: return decrypted\_data  Function generate\_checksum(data): Input: data Output: checksum 1: checksum ← sha256\_hash(data) 2: return checksum  **Function send\_secure\_api\_request(process\_id, checksum, encrypted\_data):** Input: process\_id, checksum, encrypted\_data Output: response 1: headers ← create\_headers(process\_id, checksum) 2: response ← post\_request(llm\_api\_url, headers, encrypted\_data, verify=true, timeout=30) 3: return response  **Function write\_secure\_file(file\_path, data):** Input: file\_path, data Output: None 1: file ← open\_file\_for\_writing(file\_path) 2: set\_file\_permissions(file\_path, owner\_only\_permission) 3: write\_data\_to\_file(file, data) 4: close\_file(file) 5: return |
| **Algorithm 5 Secure Memory Replacement** |
| **Input: Process ID, memory address, patched code, and original hash** |
| **Output: Process with memory replaced by secure patch** |
| **Algorithm** Input: encryption\_key, target\_pid, memory\_address, patched\_code, original\_hash Output: memory\_replacement\_success  1: memory\_replacer ← initialize\_secure\_memory\_replacer(encryption\_key) 2: try: 3: suspend\_success ← memory\_replacer.suspend\_process(target\_pid) 4: if suspend\_success is true: 5: replace\_success ← memory\_replacer.replace\_memory(target\_pid, memory\_address, patched\_code, original\_hash) 6: if replace\_success is true: 7: memory\_replacer.resume\_process(target\_pid) 8: return true 9: except exception as e: 10: output\_error\_message("Critical error", e) 11: return false  **Function initialize\_secure\_memory\_replacer(encryption\_key):** Input: encryption\_key Output: memory\_replacer 1: memory\_replacer.cipher ← initialize\_fernet\_cipher(encryption\_key) 2: memory\_replacer.logger ← setup\_logger() 3: memory\_replacer.original\_memory ← initialize\_empty\_dict() 4: return memory\_replacer  **Function setup\_logger():** Input: None Output: logger 1: logger ← create\_logger('AEGIS\_Memory\_Replacer') 2: logger.level ← INFO 3: handler ← create\_file\_handler('memory\_replacement.log') 4: handler.formatter ← create\_formatter('timestamp - message') 5: logger.add\_handler(handler) 6: return logger  **Function validate\_patch(original\_hash, patched\_code):** Input: original\_hash, patched\_code Output: is\_valid 1: current\_hash ← generate\_sha256\_hash(patched\_code) 2: is\_valid ← (current\_hash equals original\_hash) 3: return is\_valid  **Function suspend\_process(pid):** Input: pid Output: suspend\_success 1: try: 2: send\_signal\_to\_process(pid, SIGSTOP) 3: log\_info\_message("Process suspended", pid) 4: return true 5: except process\_lookup\_error: 6: log\_error\_message("Process not found", pid) 7: return false 8: return false  **Function replace\_memory(pid, base\_address, patched\_code, original\_hash):** Input: pid, base\_address, patched\_code, original\_hash Output: replace\_success 1: try: 2: validate\_result ← validate\_patch(original\_hash, patched\_code) 3: if validate\_result is false: 4: raise security\_error("Patch validation failed") 5: backup\_original\_memory(pid, base\_address, length\_of(patched\_code)) 6: write\_memory\_at\_address(pid, base\_address, patched\_code) 7: written\_data ← read\_memory\_at\_address(pid, base\_address, length\_of(patched\_code)) 8: if written\_data not\_equals patched\_code: 9: restore\_memory(pid, base\_address) 10: raise memory\_error("Memory write verification failed") 11: log\_info\_message("Memory replaced", pid) 12: return true 13: except exception as e: 14: log\_error\_message("Memory replacement failed", e) 15: restore\_memory(pid, base\_address) 16: return false 17: return false  **Function backup\_original\_memory(pid, base\_address, length):** Input: pid, base\_address, length Output: None 1: mem\_file ← open\_proc\_mem\_file(pid, "read\_binary") 2: mem\_file.seek(base\_address) 3: original\_memory[pid] ← mem\_file.read(length) 4: close\_file(mem\_file) 5: return  **Function write\_memory\_at\_address(pid, base\_address, data):** Input: pid, base\_address, data Output: None 1: mem\_file ← open\_proc\_mem\_file(pid, "read\_write\_binary") 2: mem\_file.seek(base\_address) 3: mem\_file.write(data) 4: close\_file(mem\_file) 5: return  **Function read\_memory\_at\_address(pid, base\_address, length):** Input: pid, base\_address, length Output: data 1: mem\_file ← open\_proc\_mem\_file(pid, "read\_binary") 2: mem\_file.seek(base\_address) 3: data ← mem\_file.read(length) 4: close\_file(mem\_file) 5: return data  **Function restore\_memory(pid, base\_address):** Input: pid, base\_address Output: None 1: if pid exists in original\_memory: 2: mem\_file ← open\_proc\_mem\_file(pid, "read\_write\_binary") 3: mem\_file.seek(base\_address) 4: mem\_file.write(original\_memory[pid]) 5: close\_file(mem\_file) 6: log\_warning\_message("Memory restored", pid) 7: return  **Function resume\_process(pid):** Input: pid Output: resume\_success 1: try: 2: send\_signal\_to\_process(pid, SIGCONT) 3: log\_info\_message("Process resumed", pid) 4: return true 5: except process\_lookup\_error: 6: log\_error\_message("Process not found", pid) 7: return false 8: return false |

## External API/SDKs

|  |  |  |  |
| --- | --- | --- | --- |
| **Name of API/SDK and Version** | **Description of API/SDK** | **Purpose of Usage** | **API Endpoint/Function/Class** |
| BPF Compiler Collection (BCC) v0.28 | Linux kernel tracing via eBPF | Real-time syscall monitoring and process tracing | BPF(text=...), bpf\_probe\_read() |
| Hugging Face Transformers v4.40 | Library for LLMs (Code Llama/Mistral) | LLM-based code repair and malware pattern analysis | AutoModelForCausalLM, TextStreamer |
| PostgreSQL v15 (psycopg2) | Relational database system | Secure storage of malware signatures and syscall logs | psycopg2.connect(), cursor.execute() |
| Cryptography v42.0 | Cryptographic primitives library | AES-256 encryption for logs/API communication | Fernet.generate\_key(), AESGCM |
| FastAPI v0.108 | Modern web framework | RESTful API endpoints for SIEM/IDS integration | @app.post("/detect"), APIRouter |
| PyTorch v2.2 (with CUDA) | ML framework with GPU acceleration | QLoRA fine-tuning of LLM models | peft.LoraConfig, optim.AdamW8bit |
| Linux Audit Framework | Kernel-level syscall monitoring | Complementary syscall analysis | auditd daemon, libaudit |

## User Interface

### Screen Images

Allows User to register new account or sign in with a new account in the system. Allows multiple

users to use the tool on the same system.

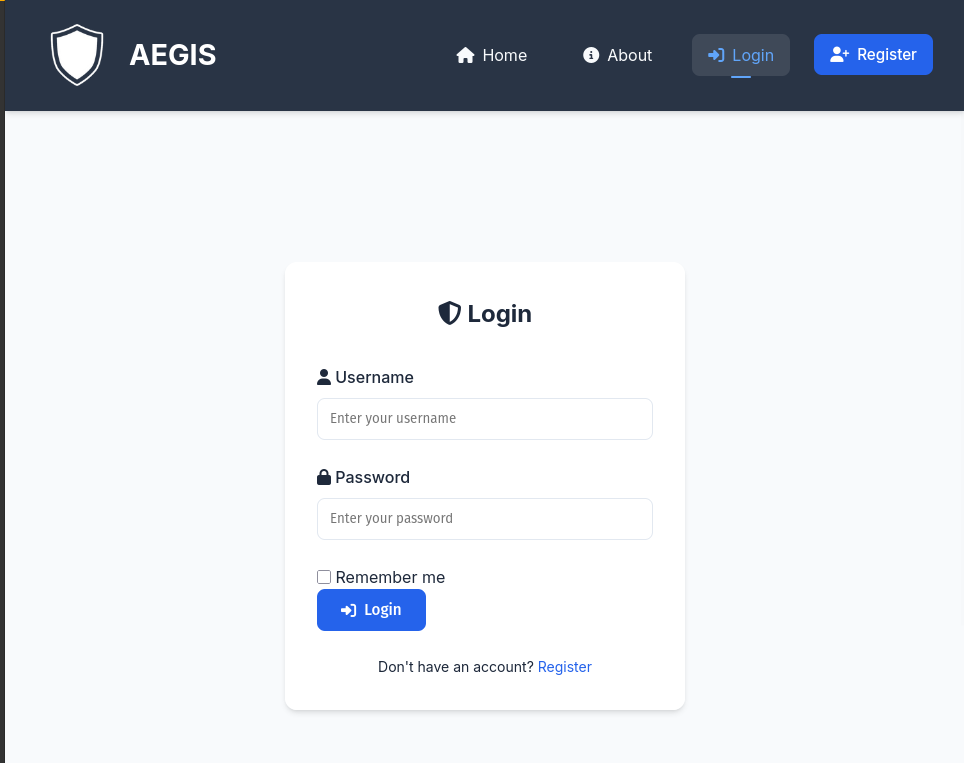


Figure Login Page

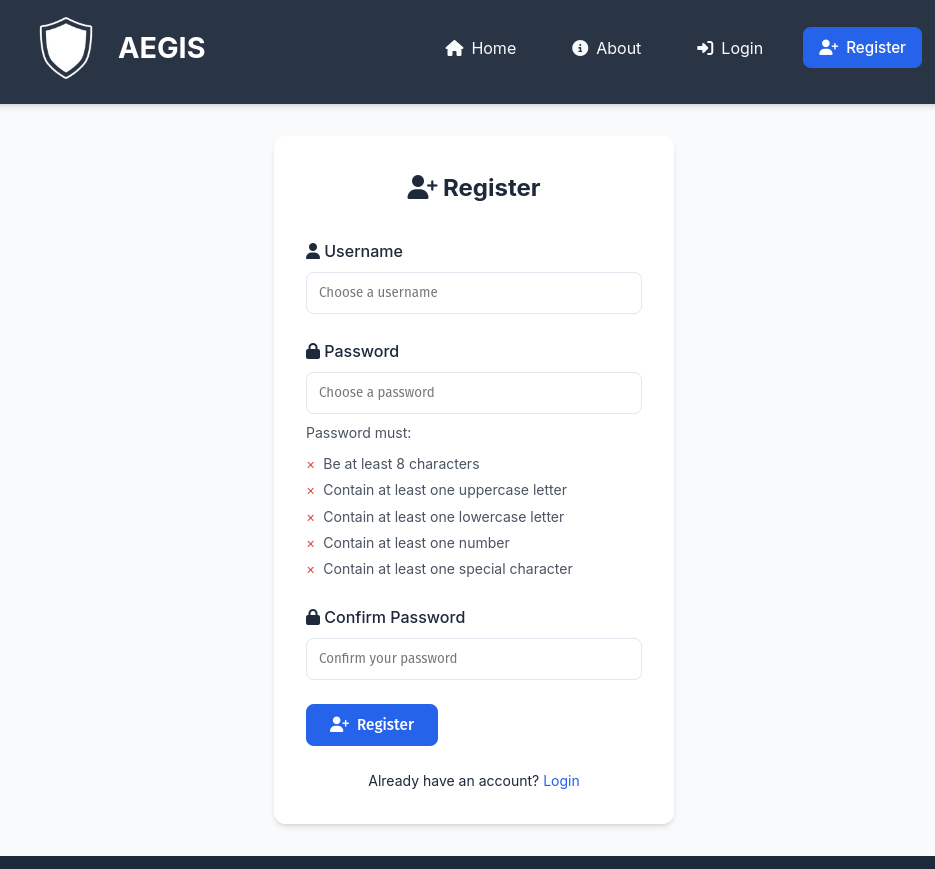


Figure Register Page

### Dashboards

Allows users to view recent activity, and review findings.

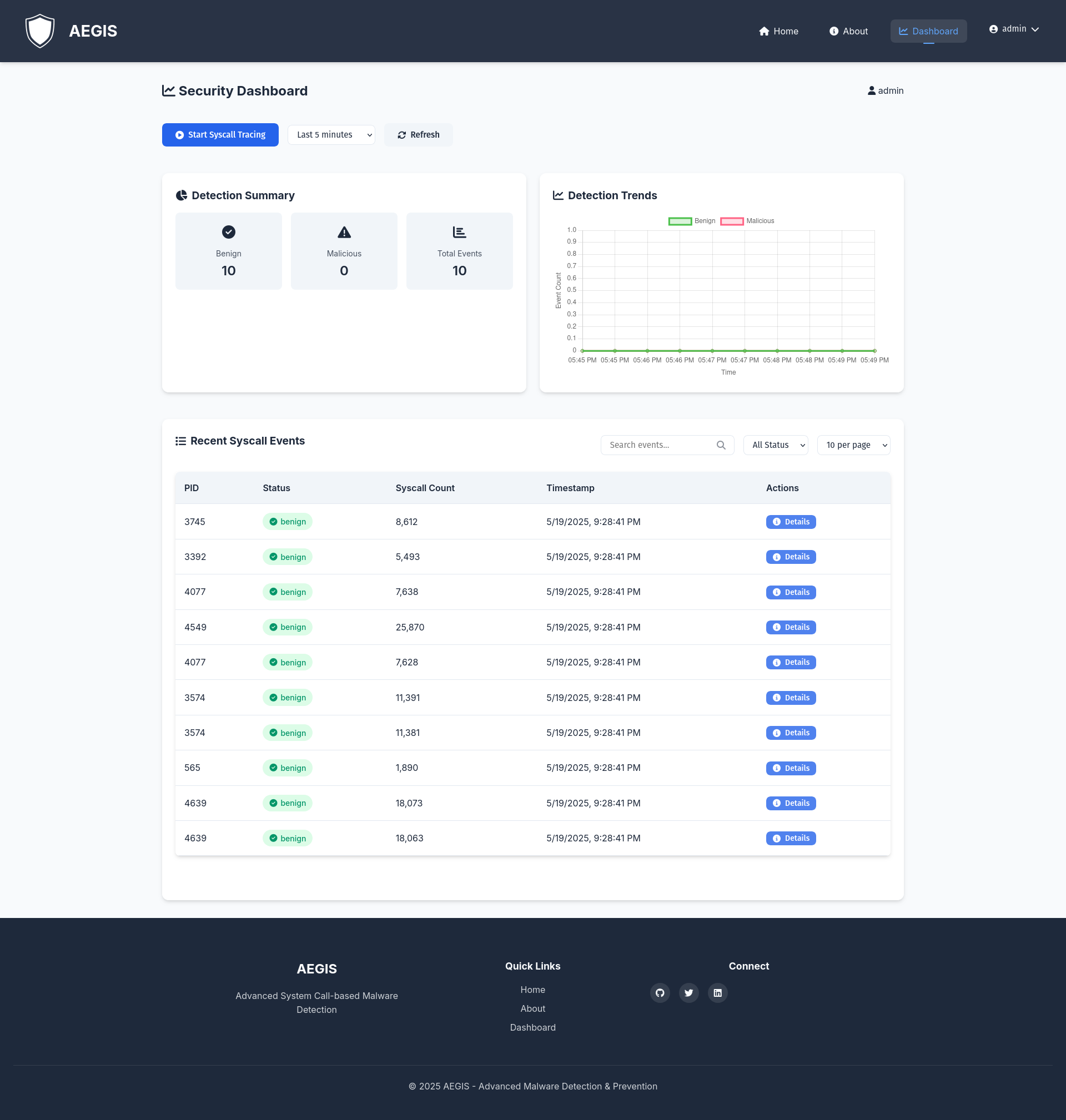


Figure Dashboard Page

### Home Page



Figure Home Page

### About Page

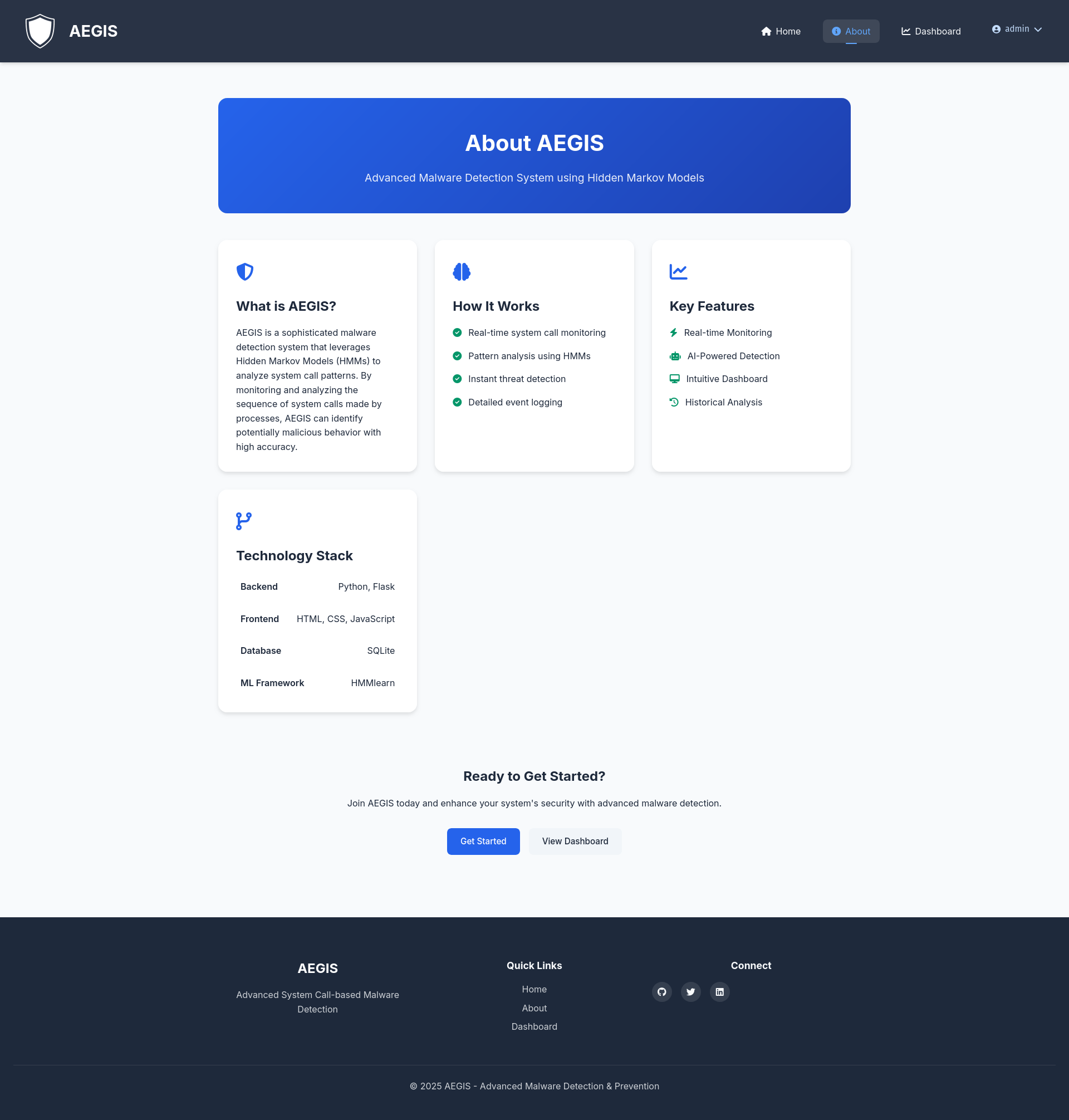


Figure About Page

## Deployment

* **Hosting Environment**

**Target Platform:**

Initially **Linux-based systems** (Ubuntu 22.04 LTS preferred)

* **Hosting Options:**

**On-premises server** (preferred for full control and low-latency syscall access)

**Private cloud (e.g., Proxmox, OpenStack)**

**Virtual machines (KVM, VirtualBox, VMware)**

* **Minimum Hardware Requirements:**

|  |  |  |
| --- | --- | --- |
| **Resource** | **Minimum** | **Recommended** |
| CPU | 4-core | 8-core |
| RAM | 8 GB | 16 GB |
| Storage | 100 GB SSD | 250 GB NVMe |
| GPU | Optional for LLM | NVIDIA RTX (if fine-tuning models) |
| OS | Linux (x86\_64) | Ubuntu 22.04 LTS |

* **Deployment Process**

1. Prepare **the Hosting Environment**

* Set up a Linux-based system (preferably Ubuntu 22.04 LTS) with internet connectivity.
* Ensure the system has sufficient resources like CPU, RAM, and storage as per project requirements.
* (Optional) Configure GPU support if local LLM fine-tuning is to be performed.

1. Install **Required Software and Dependencies**

* Install Python and essential packages for system monitoring, machine learning, and API handling.
* Install a suitable database system (PostgreSQL or MongoDB) to store logs, patches, and threat data.
* Set up supporting tools such as logging managers and web servers (e.g., Nginx) if needed.

1. Set **Up Virtual Environment for the Project**

* Create a virtual Python environment to isolate dependencies.
* Install project-specific Python libraries like psutil, scikit-learn, transformers, torch, and Flask.

1. Configure **the Project Structure**

* Clone or copy the AEGIS project source code to the server.
* Organize directories for models, logs, datasets, and runtime files.
* Ensure necessary configuration files like .env are created and properly set up with variables such as database URLs and log paths.

1. Integrate **the Large Language Model (LLM)**

* Prepare the LLM either locally or by integrating with an API-based LLM service.
* If using a local model, ensure it is placed in the correct directory and is accessible to the patching module.

1. Initialize **the System Components**

* Start the system call monitoring service.
* Launch the behavioral analysis and detection module.
* Connect all modules including process containment, code extraction, LLM-based repair, and reporting.

1. Test **the Entire Workflow**

* Run simulated scenarios with benign and malicious processes.
* Verify if detection, patching, and reporting are triggered as expected.
* Ensure all data flows correctly through the pipeline from monitoring to visualization.

1. Deploy **to Production Mode**

* Shift from development/testing mode to production by securing services.
* Enable persistent logging, database backups, and role-based access for the dashboard and API.
* Optimize for performance and minimize resource consumption.

1. Document **the System Configuration**

* Maintain documentation for deployment, environment variables, and architecture.
* Store backups of configuration files for disaster recovery.
* **Software Requirements**

|  |  |
| --- | --- |
| **Component** | **Version / Tool** |
| **Python** | **3.10+** |
| **Flask** | **REST API support** |
| **psutil** | **System process and syscall access** |
| **scikit-learn / torch** | **Behavioral model for anomaly detection** |
| **transformers / Huggingface** | **LLM patching (CodeT5, Mistral, CodeLlama)** |
| **PostgreSQL / MongoDB** | **Log storage and patch history** |
| **Web server (optional)** | **Nginx for Dashboard** |

* **Testing Environment**

**Unit Tests:**

* Write using pytest for modules like system call monitoring, patch generation, logging, etc.

**Integration Tests:**

* Test with mock malware scenarios
* Inject known bad syscall patterns and verify detection

**LLM Patch Tests:**

* Validate patch quality using static analysis tools
* Cross-check patch integrity via test harness

**Penetration Testing:**

* Tools: Metasploit, nmap, linenum, Ghidra for reverse engineering

**Simulated Malware Frameworks:**

* Use tools like **Caldera**, **Atomic Red Team**, **Infection Monkey**
* **Monitoring and Maintenance**

**Logging:**

All actions (syscalls, patches, responses) logged to database and local log files.

Use logrotate for managing log sizes.

* **Health Monitoring:**

Integrate **Prometheus + Grafana** for:

* System resource monitoring
* Alerting on CPU spikes or unusual syscall counts
* **Patch Updates:**

Automate fetching CVE updates via:

* NVD API
* GitHub security feeds

Schedule weekly model re-tuning

* **Backup Strategy:**

Use cron to automate DB and patch backup.

Store backups in encrypted format in a secure directory or S3-compatible storage.

* **Scheduled Maintenance:**

Weekly log cleanup

Monthly vulnerability audit and LLM update

Model retraining with latest exploit dataset (CVEs)

**Tools and Technologies**

|  |  |  |
| --- | --- | --- |
| **Module** | **Key Functions** | **Tools & Technologies Used** |
| **System Call Monitoring** | Capture & log syscalls | ETW, Sysmon, eBPF |
| **Detection (HMM Model)** | Classify syscalls as benign/malicious | hmmlearn, numpy, scikit-learn |
| **Memory Extraction** | Extract infected binary code | psutil, ctypes, WinDbg |
| **AI-Based Patching** | Repair malicious code using AI | Llama, GPT, QLora, requests |
| **Memory Replacement** | Restore patched code into memory | ctypes, mmap, VirtualProtectEx |
| **Logging & Web UI** | Monitor & visualize security events | Flask, React.js, PostgreSQL, WAZUH |

# Testing and Evaluation

## 8.1 Unit Testing

**Unit Testing 1: System Call Monitoring Module**

**Testing Objective:** Validate real-time syscall capture, logging, and preprocessing.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Capture syscalls from running process | Process: nginx Syscall: execve | Logs syscall metadata (PID, timestamp, args) | To be executed |
| 2 | Handle unsupported syscall type | Syscall: io\_uring\_enter (Blacklisted) | Flags as suspicious and triggers alert | To be executed |
| 3 | Validate log encryption | Log: syscall\_123.log | File encrypted with AES-256 | To be executed |
| 4 | Test preprocessing for HMM | Input: Raw syscall sequence Output: Normalized vectors | Vectors match HMM input requirements | To be executed |

**Unit Testing 2: Behavioural Analysis Module**

**Testing Objective:** Verify HMM anomaly detection and risk scoring.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Detect privilege escalation pattern | Syscall Sequence: setuid→execve→chmod | Risk score > 0.85 | To be executed |
| 2 | Validate benign process scoring | Process: systemd | Risk score < 0.2 | To be executed |
| 3 | Test rule-based detection | Syscall: ptrace with PROCESS\_VM\_WRITE | Immediate containment trigger | To be executed |
| 4 | Handle unknown syscall patterns | Input: Zero-day ransomware sequence | Flags as "Unknown Threat" | To be executed |

**Unit Testing 3: LLM-Based Code Repair Module**

**Testing Objective:** Validate malicious code extraction and secure patching.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Extract code from memory dump | Input: malware.dmp Offset: 0x7f34a3bcd000 | Isolates vulnerable code section | To be executed |
| 2 | Generate patch for buffer overflow | Vulnerable Code: strcpy(dest, src) | Replaces with strncpy(dest, src, len) | To be executed |
| 3 | Validate patch syntax | Code: LLM-generated patch | No compilation errors | To be executed |
| 4 | Test QLoRA fine-tuning | Training Data: CVE-2024 samples | Validation loss < 0.15 | To be executed |

**Unit Testing 4: Process Containment Module**

**Testing Objective:** Test process termination and security restrictions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Terminate malicious process | PID: 5678 Process: cryptominer | Process status: Terminated | To be executed |
| 2 | Apply network restrictions | Process: malware.exe | iptables rule added | To be executed |
| 3 | Prevent process resurrection | Malware: Fork bomb | Child processes blocked | To be executed |
| 4 | Validate containment logging | Event: Process termination | Log entry with CONTAINED status | To be executed |

## 8.2 Functional Testing

**Functional Testing 1: End-to-End Malware Detection**

**Testing Objective:** Validate detection→containment→patching workflow.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Ransomware detection & patching | Sample: WannaCry variant | 1. Detection <5s 2. Auto-patch applied | To be executed |
| 2 | Rootkit process containment | Process: Hidden kernel module | 1. Detection via syscall anomaly 2. Kernel module unloaded | To be executed |
| 3 | Multi-stage attack handling | Attack Chain: Phishing→C2→Lateral Movement | All stages detected & contained | To be executed |
| 4 | Validate system performance | Load: 10k processes | CPU usage < 30% | To be executed |

**Functional Testing 2: LLM Patching Workflow**

**Testing Objective:** Test code repair lifecycle

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Validate end-to-end patching | Input: Vulnerable libssl code | 1. Patch generated <10s 2. Passes ASAN checks | To be executed |
| 2 | Test patch rollback | Input: Invalid LLM patch | Original code restored | To be executed |
| 3 | Verify API communication | Test: MITM attack simulation | TLS 1.3 handshake fails | To be executed |
| 4 | Stress-test LLM module | Concurrent Requests: 100 | Response time <2s/req | To be executed |

**Functional Testing 3: API Integration**

**Testing Objective:** Validate Smooth flow of API requests integration.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Test authentication | Invalid JWT token | 401 Unauthorized | To be executed |
| 2 | Validate malware submission | Payload: Encrypted malware.bin | Returns X-Patch-ID header | To be executed |
| 3 | Test rate limiting | Requests: 100/sec | 429 Too Many Requests after 50 | To be executed |
| 4 | Verify log streaming | Duration: 1hr | Real-time updates to Splunk | To be executed |

**Functional Testing 4: Logging & Reporting**

**Testing Objective:** Validate secure logging and report generation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Test log immutability | Action: Modify containment.log | Permission denied error | To be executed |
| 2 | Validate report generation | Input: 24hr log data | PDF/CSV report with threat stats | To be executed |
| 3 | Test log compression | File: syscall.log (10GB) | Compressed size <2GB | To be executed |
| 4 | Verify RBAC for logs | User: Analyst role | Read-only access enforced | To be executed |

## 8.3Integration Testing

**Integration Testing 1: Detection-to-Containment Workflow**

**Testing Objective:** Validate seamless interaction between System Call Monitoring → Behavioural Analysis → Process Containment.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | Detect & contain ransomware | Syscall: File encryption sequence Process: malware.exe | 1. Detection <2s 2. Process terminated 3. Network blocked | To be executed |
| 2 | Handle false positives | Process: legitimate\_app | 1. Risk score <0.3 2. No containment | To be executed |
| 3 | Multi-process containment | Attack: Fork bomb | All child processes terminated | To be executed |
| 4 | Validate kernel-level isolation | Malware: Rootkit | 1. Kernel module unloaded 2. SELinux context enforced | To be executed |

**Integration Testing 2: Patching-to-Execution Workflow**

**Testing Objective: Validate integration between Malicious Code Extraction → LLM Repair → Secure Execution.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Test Case/Test Script** | **Attribute and Value** | **Expected Result** | **Result** |
| 1 | End-to-end buffer overflow repair | Code: strcpy(dest, src) LLM: CodeLlama-13b | 1. Patch: strncpy 2. ASAN validated 3. Process resumed | To be executed |
| 2 | Handle invalid LLM patches | Code: Corrupted response | 1. Rollback to original 2. Alert to analyst | To be executed |
| 3 | Validate encrypted API comms | Test: MITM attack | TLS 1.3 handshake failure | To be executed |
| 4 | Performance under load | Concurrent patches: 50 | Avg. latency <15s | To be executed |

# References

1. Fu, M. (2024). *A Case Study of LLM for Automated Vulnerability Repair: Assessing Impact of*

*Reasoning and Patch Validation Feedback*. ResearchGate.

[https://www.researchgate.net/publication/380895093\_A\_Case\_Study\_of\_LLM\_for\_Automated\_Vulne rability\_Repair\_Assessing\_Impact\_of\_Reasoning\_and\_Patch\_Validation\_Feedback](https://www.researchgate.net/publication/380895093_A_Case_Study_of_LLM_for_Automated_Vulnerability_Repair_Assessing_Impact_of_Reasoning_and_Patch_Validation_Feedback)

*This study evaluates how Large Language Models (LLMs) can automate vulnerability repair, emphasizing the importance of reasoning and feedback in improving patch quality.*

1. Mishra, A. (2024). *AI-Augmented Vulnerability Repair: A Thesis*. University of Michigan. [https://deepblue.lib.umich.edu/bitstream/handle/2027.42/195066/Mishra\_Thesis\_AI\_Augmented\_Vuln erability.pdf?sequence=1&isAllowed=y](https://deepblue.lib.umich.edu/bitstream/handle/2027.42/195066/Mishra_Thesis_AI_Augmented_Vulnerability.pdf?sequence=1&isAllowed=y)

*This thesis explores the integration of AI in vulnerability analysis and repair, focusing on augmenting traditional cybersecurity practices with machine learning techniques.*

1. Sahu, D. (n.d.). *System Call Analysis in the Age of AI: Revolutionizing Malware Detection with Our Own Machine*. Medium. [https://medium.com/@deepsahu1/system-call-analysis-in-the-age-of-airevolutionizing-malware-detection-with-our-own-machine-8ecfc638d803](https://medium.com/@deepsahu1/system-call-analysis-in-the-age-of-ai-revolutionizing-malware-detection-with-our-own-machine-8ecfc638d803)

*The article discusses the use of system call analysis combined with AI techniques to enhance malware detection accuracy and reduce false positives.*

1. ScienceDirect. (2024). *Automated Software Vulnerability Patching using Large Language*

*Models*. <https://www.sciencedirect.com/science/article/pii/S0952197624014490>*This research investigates how LLMs can automatically generate secure patches for identified software vulnerabilities, potentially improving software resilience.*

1. MichaelFu1998. (n.d.). *Vulrepair: AI-Driven Vulnerability Patching*. [https://michaelfu1998create.github.io/papers/vulrepair.pdf](https://michaelfu1998-create.github.io/papers/vulrepair.pdf)

*The paper presents Vulrepair, a tool that leverages LLMs for automated software vulnerability patching, focusing on code analysis and secure patch generation.*

1. Emory University. (2024). *AI-Driven Vulnerability Analysis: Case Studies and Frameworks*. <https://etd.library.emory.edu/concern/etds/4j03d103r>

*This academic work offers case studies and frameworks for applying AI techniques to software vulnerability detection and remediation.*

1. ArXiv. (2024). *Automated Software Vulnerability Patching using Large Language Models*. <https://arxiv.org/pdf/2409.18395>

*A preprint detailing method for using LLMs to automate the process of identifying and patching software vulnerabilities efficiently.*

1. ArXiv. (2024). *AI-Driven Malware Detection with System Call Analysis*. <https://arxiv.org/html/2404.01096v1>

*The study explores combining system call analysis with AI models to detect malware, aiming to enhance detection precision and reduce manual effort.*

1. WIPO. (2015). *Methods and System Handling Malware* (Patent No. WO2015029037A3). World Intellectual Property Organization. <https://patentscope.wipo.int/search/en/WO2015029037>*This patent by Intel Corporation describes a system for detecting and handling malware threats, focusing on real-time threat analysis and mitigation techniques.*

1. Google Patents. (2011). *System and Method for Detecting Malicious Code* (Patent No. US20110252468A1). <https://patents.google.com/patent/US20110252468A1/en>*The patent outlines a method for detecting malicious code by analyzing program behavior, contributing to proactive malware defense strategies.*

1. Google Patents. (2020). *Malware Detection Using Behavioral Analysis* (Patent No. US10789361B2). <https://patents.google.com/patent/US10789361B2/en>*This patent describes a method that uses behavioral analysis techniques to identify malware activity within computing environments.*

# Plagiarism Report