# Software Requirements Specification

**for**

# AUTOPENT

**Version 1.0 approved**

**Prepared by Himanshu Gaur (23bcy10127)**
**Abhinav Mehra (23bcy10015)**
**Supreety Jha(23bcy10150)**
**Ishaan Shrivastava (23BCY10326)**

**VIT BHOPAL UNIVERSITY**

**July 25, 2025**

# Table of Contents



# Revision History

| Name | Date | Reason For Changes | Version |
|---|---|---|---|
| | | | |
| | | | |

## 1. Introduction

This section provides a comprehensive overview of the Software Requirements Specification (SRS) for the product designated as AutoPent v1.0. It outlines the primary purpose, scope, intended audience, and foundational context of the system, serving as the authoritative guide for its development and evaluation.

### 1.1 Purpose

The primary purpose of this document is to provide a detailed, unambiguous specification of the functional and non-functional requirements for AutoPent v1.0. This system is conceived as an AI-augmented security automation platform designed to streamline and accelerate the penetration testing lifecycle. In an environment where cyber threats are constantly evolving and the demand for skilled security professionals outstrips supply, AutoPent aims to bridge the gap by automating laborious tasks and providing intelligent, actionable insights.

It achieves this by orchestrating a suite of industry-standard reconnaissance and vulnerability scanning tools, then channeling the output into a proprietary analysis engine powered by the LLaMA 3.1 Large Language Model. This integration allows AutoPent to move beyond simple data aggregation and perform contextual analysis, prioritize vulnerabilities based on exploitability, and drastically reduce the noise of false positives.

This SRS serves as the single source of truth for all project participants. It is intended to:

- Guide Development: Provide the engineering team with a clear set of requirements to architect, design, and implement the system.
- Inform Testing: Supply the quality assurance team with a concrete basis for creating test cases and validation scripts.
- Align Stakeholders: Ensure all stakeholders, from project managers to end-users, have a shared and consistent understanding of the system's scope, capabilities, and limitations.

### 1.2 Document Conventions

To ensure clarity and consistency throughout this document, the following conventions have been adopted.

- Requirement ID: All requirements are assigned a unique identifier for traceability. The format is [MOD]-[TYPE]-[###], where MOD represents the module, TYPE signifies the requirement type (F for Functional, NF for Non-Functional), and ### is a sequential number. For example, REC-F-001 refers to the first functional requirement in the Reconnaissance module.
- Requirement Priority: Each requirement is assigned a priority level to guide development efforts. These levels are High (critical for the Minimum Viable Product), Medium (important but can be deferred post-MVP), and Low (a "nice-to-have" feature for future releases).

- Bold Text: This formatting is used to highlight section headings, key terms, and requirement priority levels for emphasis, such as the Reconnaissance Module.
- Monospace Font: This font is used to denote system commands, file paths, code snippets, tool names, or API endpoints. For example, the system will execute commands like nmap -sV -p- <target>.
- *Italic Text:* This formatting is used for notes, comments, or to introduce new terms. For example, *This feature depends on the availability of an external API.*

### 1.3 Intended Audience and Reading Suggestions

This document has been prepared for a diverse audience, and each role will have a different focus.

- For Developers and Engineers: This group is concerned with the technical implementation details, API contracts, system architecture, and constraints. They should read the entire document, paying close attention to Sections 3, 4, and 5 for the most detailed technical and functional requirements.
- For QA and Test Engineers: The primary focus for this group is the validation of functional and non-functional requirements and the development of comprehensive test plans. They should concentrate on Sections 3 and 5 to create test cases that cover all specified system behaviors and performance benchmarks.
- For Project Managers: Their interest lies in the overall project scope, feature prioritization, deliverables, and tracking development progress. Sections 1 and 2 will provide a high-level overview, while the requirement IDs throughout Section 3 will be crucial for progress tracking.
- For Stakeholders and Clients: This audience is focused on understanding the business goals, high-level system capabilities, and the product's overall value proposition. Sections 1.4 (Product Scope) and 2 (Overall Description) are most relevant as they provide a clear, non-technical summary of the system.

### 1.4 Product Scope

AutoPent v1.0 is an integrated cybersecurity framework engineered to automate up to 70% of the manual effort involved in a typical network and web application penetration test. Its core mission is to empower organizations—particularly those without dedicated, full-time security teams—to conduct frequent, consistent, and intelligent security audits.

**Key Capabilities:**

- Automated Reconnaissance: The system will autonomously perform comprehensive information gathering, including subdomain enumeration, port scanning, service identification, and technology stack fingerprinting.
- Intelligent Vulnerability Scanning: It will orchestrate leading scanning tools to identify a wide range of vulnerabilities, with a specific focus on the OWASP Top 10, misconfigurations, and outdated software.
- AI-Powered Analysis & Triage: Leveraging the LLaMA 3.1 model, AutoPent will analyze raw scanner output to correlate findings, eliminate false positives, and prioritize vulnerabilities based on contextual factors like exploit availability and potential business impact.
- Actionable Reporting: The system will generate comprehensive, human-readable reports that not only list vulnerabilities but also provide AI-driven explanations of the risks and suggest clear, actionable remediation steps.

**Out of Scope for v1.0:**

To ensure a focused and achievable initial release, the following functionalities are explicitly out of scope for AutoPent v1.0: automated exploitation or post-exploitation activities; physical security assessments; social engineering attack simulations; Denial of Service (DoS/DDoS) testing; and mobile application (iOS/Android) penetration testing.

**1.5 References**

This document is based on and refers to the following industry standards, tools, and technologies which provide foundational knowledge for understanding the system's operational context.

1. OWASP Top 10 Project: This resource serves as the primary checklist for web application vulnerabilities that AutoPent must be able to identify and report on.
2. Metasploit Framework: This is a key reference for understanding exploit payloads and methodologies, which informs the AI's analysis of a vulnerability's severity and real-world exploitability.
3. Nmap Documentation: As the primary tool for network discovery and security auditing, its scripting engine (NSE) is a core component of the Reconnaissance module.
4. SQLMap Documentation: This is the designated tool for detecting and exploiting SQL injection flaws. AutoPent will integrate and automate its usage for comprehensive database security testing.
5. LLaMA 3.1 - Meta AI: This document refers to the foundational Large Language Model used for the AI Analysis module, which is responsible for data interpretation, contextual analysis, and report generation.
6. IEEE Std 830-1998: This is the IEEE Recommended Practice for Software Requirements Specifications, which provides the structural and content guidelines followed in this document.

## 2. Overall Description

This section provides a high-level, non-technical overview of the AutoPent system. It describes the product's core perspective, its key functions, the intended user base, the operational environment, and the principal constraints and assumptions that will shape its design and development.

### 2.1 Product Perspective

**AutoPent v1.0** is engineered as a new, fully **standalone cybersecurity automation tool** designed for secure, **on-premise deployment**. It is not an iteration of a previous system but a novel framework created to address the inefficiencies of traditional, disjointed penetration testing workflows. The system's core philosophy is to create a unified, self-contained environment that brings together best-in-class open-source security tools—such as **Nmap**, **SQLMap**, **Nikto**, and the **Metasploit framework**—with the advanced analytical power of a locally hosted Large Language Model, specifically **LLaMA 3.1**.

In essence, AutoPent acts as an intelligent orchestration layer. It bridges the gap between the raw, technical output of traditional scanners and the need for human-readable, actionable insights. By managing the end-to-end testing process within a single interface, it eliminates the need for security professionals to manually context-switch between different tools and terminals, streamlining the process from initial reconnaissance to final reporting. This integrated approach ensures that all sensitive scan data and analysis remain within the user's control, reinforcing data privacy and security.

### 2.2 Product Functions

The AutoPent system is designed to perform several major functions that constitute a complete, streamlined penetration testing workflow.

- **Automated Reconnaissance:** The system initiates its process by building a comprehensive "attack surface map" of the target. It performs extensive domain and subdomain enumeration by leveraging multiple data sources, including **Shodan** and **BinaryEdge**. This stage is critical for discovering all potential entry points associated with a target organization.
- **Comprehensive Vulnerability Scanning:** Following reconnaissance, AutoPent launches a multi-pronged scanning phase. It systematically tests for the **OWASP Top 10** web application vulnerabilities using specialized tools like **Nikto** for web server misconfigurations, **Nmap** with its extensive library of NSE scripts for network-level issues, and **SQLMap** for detecting and analyzing SQL injection flaws.
- **Intelligent Exploit Mapping:** Beyond just identifying vulnerabilities, the system provides crucial context by mapping discovered flaws to relevant exploit modules within the **Metasploit framework**. This function helps users understand the real-world risk and potential impact of a vulnerability by suggesting a direct path to exploitation.
- **Advanced AI Analysis:** This is the core innovative function of AutoPent. All

findings from the previous stages are fed into the local **LLaMA 3.1** model. The AI engine processes this technical data to summarize complex findings, correlate related vulnerabilities, and generate remediation advice in clear, human-readable language, effectively translating technical jargon into actionable business risk.

- **Professional Security Report Generation:** Upon completion of the analysis, the system compiles all information—reconnaissance data, a detailed list of vulnerabilities with severity ratings, and the AI's recommendations—into a single, clean, and professional **PDF report**. This downloadable artifact is structured to be easily shared with both technical teams and management stakeholders.

### 2.3 User Classes and Characteristics

AutoPent is designed to serve a diverse set of users, each with unique needs and technical expertise.

- **Cybersecurity Engineers:** This is the primary power-user group. These technically proficient users are comfortable with penetration testing tools and methodologies. They will use AutoPent to accelerate their workflow, automate repetitive tasks, and leverage the AI analysis as a "second opinion" to validate their own findings and speed up report writing.
- **Startup Founders and DevOps Professionals:** This user class often operates without a dedicated security team. They require a tool that provides quick, clear, and actionable insights into their web applications' security posture. For them, AutoPent serves as an accessible first line of defense, with a strong emphasis on the AI-generated summaries and straightforward remediation steps.
- **IT Service Providers and MSPs:** These firms manage IT infrastructure and security for multiple clients. They require a scalable and efficient solution to conduct regular security audits across their client portfolio. AutoPent enables them to offer standardized, repeatable, and high-quality penetration testing services without a proportional increase in manual labor.
- **Cybersecurity Educators and Students:** In an academic or training setting, AutoPent serves as a valuable teaching tool. It allows educators to demonstrate a modern, AI-integrated penetration testing workflow in a controlled lab environment, helping students grasp both the classic toolchains and the future of AI in cybersecurity.

### 2.4 Operating Environment

The system is designed to run in a specific on-premise environment. The minimum **hardware requirements** include at least **8GB of RAM** and a mid-range, CUDA-compatible GPU, such as an **NVIDIA RTX 3050** or better, to handle the processing demands of the language model. Approximately **10GB of disk space** is required to store the model files, the integrated tools, and generated logs and reports.
For the **operating system**, AutoPent is optimized for **Linux** distributions (such as

Ubuntu or Kali Linux), which provide native support for most of the integrated security tools. However, it will also be supported on **Windows** for broader accessibility.
The system has several key **software dependencies**. It requires **Python 3.10** or a newer version as its core runtime. The user must have local installations of the essential security tools: **Nikto**, **SQLMap**, **Nmap**, and the **Metasploit framework**. Finally, specific Python libraries are needed, including llama_cpp for running the model, pdfkit for report generation, and Flask for the user interface. The local language model itself is a GGUF-formatted version of **LLaMA 3.1**.

### 2.5 Design and Implementation Constraints

Several key constraints will govern the architecture and development of AutoPent to ensure it meets its core objectives.

- **Strict Offline Capability:** To guarantee user privacy and data security, the AI analysis module must operate entirely offline. After the initial setup and model download, the system must not rely on any external APIs or internet connectivity for its core functions.
- **Reliance on Open-Source Tools:** The system's functionality is heavily dependent on the integration of established open-source tools. This introduces a constraint to honor all associated licenses and to design the system flexibly enough to accommodate potential updates or changes in these external dependencies.
- **GPU Dependency for AI Features:** While other parts of the system can run on a standard CPU, the AI analysis and summarization features are computationally intensive and require a CUDA-compatible GPU for acceptable performance. This is a significant hardware constraint for users wishing to leverage the full power of the system.
- **Local Execution Mandate:** Reinforcing the offline constraint, no cloud integration for AI processing is permitted. All data, from target information to vulnerability findings, must remain within the local execution environment controlled by the user.
- **Modular and Testable Architecture:** The system must be designed with a highly modular architecture. Each core module—Reconnaissance, Scanning, AI Analysis, and Reporting—must be developed as a separable component that can be tested and maintained independently. This ensures system robustness and simplifies future upgrades.

### 2.6 User Documentation

A comprehensive suite of user documentation will be provided with the product to ensure a smooth user experience from installation to daily operation. This includes an **Installation Guide**, which will provide clear, platform-specific instructions for setting up the environment, installing the required third-party tools, and managing all software dependencies. A detailed **User Manual** will offer step-by-step instructions for operating each module, running a complete scan, and interpreting the results.

For advanced users, a **Model Deployment Guide** will explain the process of downloading, configuring, and updating the local LLaMA 3.1 model. Finally, a **Troubleshooting Guide** will serve as a first point of reference for common errors, providing known solutions and contact information for further support.

### 2.7 Assumptions and Dependencies

The successful operation of AutoPent relies on several key assumptions and dependencies regarding the user and the operating environment. It is **assumed** that users will follow the setup guide to correctly install all required tools and dependencies before running the application. The system is **dependent** on internet access during the initial configuration phase solely for downloading the necessary tools and the LLaMA 3.1 model files. A critical assumption is that the user has obtained proper authorization to perform penetration testing on the target systems; AutoPent is a tool for ethical and legal security auditing only. For users operating the tool via its command-line interface, a basic knowledge of Linux commands is assumed. Finally, the system's long-term compatibility is **dependent** on the continued maintenance and updates of the integrated open-source tools by their respective communities.

## 3. External Interface Requirements

This section defines the interfaces between the AutoPent system and the outside world. It details how the software interacts with users, underlying hardware, other software components, and communication protocols.

### 3.1 User Interfaces

AutoPent is designed with a dual-interface approach to cater to users with different technical preferences and use cases. It provides both a powerful Command-Line Interface (CLI) for technical experts and an intuitive Graphical User Interface (GUI) for users who prefer a visual workflow.

- **Command-Line Interface (CLI):** The CLI is the core interface for AutoPent, designed for power users, security engineers, and for integration into automated scripts and larger security workflows. It operates through command-driven execution, providing detailed, real-time log outputs directly in the terminal. The interface is interactive, guiding the user through a series of prompts to specify the target domain, select the desired scan types (e.g., Reconnaissance only, Full Scan), and configure report generation options. Its design prioritizes efficiency, speed, and scriptability, with clear error messages and concise summary outputs to ensure that both successes and failures are communicated effectively.
- **Graphical User Interface (GUI):** Built as an optional frontend layer using Streamlit, the GUI provides an accessible and user-friendly way to interact with AutoPent's powerful backend. This interface is optimized for clarity and ease of use, making it ideal for non-security experts like DevOps engineers or startup founders. The main view presents a minimalistic layout with large, clearly-labeled buttons for each core module: **Recon**, **Vulnerability Scan**, **AI Analysis**, and **Report Download**. A central text area displays the live output from the currently running module, giving users visual feedback on the system's progress. Upon completion, a "Download Report" button becomes active, allowing the user to easily save the final PDF analysis.

Across the GUI, standard components are used to ensure a consistent and predictable user experience. Input fields for target URLs or domains include built-in validation to prevent common entry errors. Helpful tooltips are available for each module to explain its function. The overall visual design, while documented separately in a UI specification document, is guided by principles of minimalism and responsiveness.

### 3.2 Hardware Interfaces

AutoPent is a software system that interfaces with the host machine's hardware through high-level operating system abstractions and specialized libraries. It does not perform any direct low-level hardware communication.

The most significant hardware interaction is with the system's **Graphics**

**Processing Unit (GPU)**. To perform its advanced AI analysis, AutoPent is designed to leverage **CUDA-enabled NVIDIA GPUs** (with a recommended minimum of an RTX 3050). The software interfaces with the GPU via Python libraries that offload the computationally intensive matrix operations of the LLaMA 3.1 model. This results in dramatically accelerated processing for vulnerability analysis and report generation. For systems without a compatible GPU, a **CPU fallback** mechanism is available. This ensures the software can run on a wider range of hardware, though it comes with the trade-off of significantly slower performance during the AI analysis phase.
The system also interfaces with the machine's storage hardware for **Disk I/O**. This includes reading the large LLaMA model files (approximately 10GB) into memory, writing detailed logs during operation, and saving the final PDF reports to the user's specified location. Standard interactions with network adapters are also necessary for any scanning or API-based reconnaissance tasks.

### 3.3 Software Interfaces

As an orchestration tool, AutoPent's primary function is to interface with a suite of third-party software tools and libraries to perform its tasks.

- **Third-Party Tool Interfaces:** The system executes and manages several external security tools. It interfaces with **Nmap (v7.94)** for network scanning and NSE scripting, **Nikto (v2.5.0)** for web server vulnerability scanning, and the latest version of **SQLMap** for SQL injection testing. These tools are typically invoked as separate processes via the command line, and AutoPent is responsible for passing the correct arguments and parsing their structured output. The interface with the **Metasploit** framework is handled programmatically via the pymetasploit3 library for exploit suggestion. The core AI capability is achieved by interfacing with **LLaMA 3.1** through the llama-cpp-python library, which acts as a bridge between the Python application and the locally running language model.
- **Python Library Interfaces:** The system is built upon a foundation of key Python libraries. System-level interactions, command execution, and text parsing are handled by standard libraries like subprocess, re, json, and os. The llama_cpp library provides the critical integration with the LLaMA 3.1 model. Final reports are generated using the pdfkit library, which converts HTML content to PDF. The optional GUI is made possible through libraries like Flask or Streamlit. For reconnaissance, libraries such as requests, shodan, and binaryedge are used to communicate with external APIs.
- **Shared Data Format:** To ensure seamless communication between its internal modules, AutoPent uses **JSON (JavaScript Object Notation)** as its standard data serialization format. The output from one module (e.g., Reconnaissance) is packaged into a JSON object and passed to the next module (e.g., Scanning). This lightweight, human-readable format was chosen for its universal compatibility and ease of parsing, creating a robust and maintainable internal data pipeline.

### 3.4 Communications Interfaces

AutoPent's communication with external networks is intentionally limited and strictly controlled to ensure user privacy and data security.

- **Optional API Communication:** For passive reconnaissance, the system can optionally communicate with the public APIs of services like **Shodan**, **BinaryEdge**, and **Onyphe**. All such communication is conducted securely over **HTTPS**, ensuring that the data exchanged between AutoPent and the API servers is encrypted with TLS. User-provided API keys and tokens for these services are stored locally on the user's machine and are never transmitted to any other external service.
- **Networking Requirements for Scanning:** During active scanning, the system requires outbound network access to send probes and requests to the specified target systems. However, the core vulnerability scanning and AI analysis modules are designed to operate in a fully **offline mode**. Once the LLM is loaded, no internet connection is required for the system to analyze findings and generate a report.
- **Core Security Standards:** The system is built on a "local-first" principle. No sensitive data, including target information, scan results, or vulnerability reports, is ever sent to the cloud or any third-party server. All operations are, by design, performed locally on the user's machine. This commitment to local execution is a fundamental security constraint of the product.

## 4. System Features

This section provides a detailed breakdown of the core functional modules of AutoPent. Each feature is described in terms of its purpose, priority, user interaction, and a set of specific, testable functional requirements.

### 4.1 Reconnaissance Automation Module

#### 4.1.1 Description and Priority

This module represents the foundational stage of the AutoPent workflow, automating the critical process of **Open-Source Intelligence (OSINT)** gathering. Its purpose is to build a comprehensive "attack surface map" of a given target domain by collecting publicly available information from sources like **Shodan**, **BinaryEdge**, and **WHOIS** lookups. This initial data collection is paramount, as the quality and completeness of its findings directly influence the effectiveness of all subsequent scanning and analysis phases.

**Priority: High.** The strategic benefit of this module is immense, as it provides the essential context needed to conduct an intelligent, targeted security assessment. The penalty for its failure is equally high; without a clear picture of the target's assets, any subsequent vulnerability scans would be incomplete and potentially ineffective. The implementation cost is relatively low as it leverages existing APIs, and the risk is minimal.

#### 4.1.2 Stimulus/Response Sequences

The interaction begins when the **user provides a target domain** (e.g., example.com) or IP address as input. In response, the system initiates a sequence of automated data collection tasks. It queries the configured APIs and other sources to gather a wide array of information, including but not limited to: associated IP addresses, open network ports, the names and versions of services running on those ports, DNS records (A, MX, TXT), known subdomains, and the underlying technologies used by the web applications. This data is then aggregated and structured for the next stage.

#### 4.1.3 Functional Requirements

- **REQ-1: Input Sanitization.** The system must accept user input in the form of a fully qualified domain name (FQDN) or an IPv4 address. It must sanitize this input by stripping any protocol prefixes (e.g., http://) and removing invalid characters to prevent errors in downstream tools.
- **REQ-2: Secure API Integration.** The system must securely interface with the Shodan and BinaryEdge APIs. User-provided API keys must be stored in a secure manner, such as in a configuration file or as environment variables, and never hardcoded into the source. All communications with these APIs must be conducted over HTTPS.
- **REQ-3: Structured Data Output.** Upon completion, the module must consolidate all collected reconnaissance data into a single, structured **JSON**

object. This machine-readable format ensures that the data can be reliably and efficiently passed to the Vulnerability Scanning module.

- **REQ-4: Request Timeout and Retry Logic.** The system must implement robust error handling for API requests. Each request shall have a timeout of **15 seconds**. If a request fails or times out, the system shall retry up to two more times. If the API remains unresponsive, the system will log the failure and continue, ensuring that the entire process does not hang on a single unresponsive source.
- **REQ-5: Offline Reconnaissance Fallback.** In the event that no API keys are provided or the system is offline, the module must fall back to using locally available tools (e.g., dig, whois) to perform basic DNS and WHOIS lookups. This provides a baseline level of reconnaissance data even without internet connectivity.

### 4.2 Vulnerability Scanning Module

#### 4.2.1 Description and Priority

This module performs the active investigation phase of the penetration test. It takes the information gathered during reconnaissance and uses it to actively probe the target for known vulnerabilities and security misconfigurations. The module orchestrates a suite of powerful, industry-standard tools, including **Nikto**, **Nmap (using the Nmap Scripting Engine)**, and **SQLMap**, to conduct a thorough assessment focused on network-level issues and the **OWASP Top 10** web application vulnerabilities.

**Priority: High.** The benefit of this module is core to the product's value proposition—it is the engine that finds the actual security flaws. The penalty for its failure is severe, as it would render the entire tool ineffective. The implementation cost is moderate due to the complexity of integrating and parsing output from multiple tools, and the risk is moderate, as active scanning can sometimes impact system stability if not configured properly.

#### 4.2.2 Stimulus/Response Sequences

The module is triggered by the structured data passed from the reconnaissance phase or by direct user input of a target URL. Upon activation, the system **executes the configured scanners** against the target. It then parses the raw output from each tool, filters the results to remove informational and low-priority findings, and flags any identified critical issues.

#### 4.2.3 Functional Requirements

- **REQ-6: Filtered Nikto Execution.** The system shall run the **Nikto** web scanner with configurations designed to filter its output, focusing the results on the **top 25 most common and critical CVEs** to reduce noise and improve the signal-to-noise ratio.
- **REQ-7: Scoped SQLMap Execution.** The system's **SQLMap** scans shall be limited to testing for the top 10 most common database backends (e.g., MySQL, PostgreSQL, MSSQL) to ensure scan times remain reasonable. The final summary of its findings must be condensed to avoid verbose, multi-page

raw outputs.

- **REQ-8: Service Version Extraction.** During Nmap scans, the system must be configured to perform service version detection and extract version numbers for all discovered services (e.g., Apache 2.4.51, OpenSSH 8.2). This information is critical for the subsequent exploit mapping process.
- **REQ-9: Standardized JSON Results.** All results from all scanners must be parsed and stored in a unified **JSON** format. The raw, verbose log files from the individual tools must be excluded from the final user-facing report to maintain clarity and conciseness.
- **REQ-10: Exploit Identification.** The system shall use an internal, predefined signature-to-exploit map. This map will be used to cross-reference the identified vulnerabilities (e.g., by CVE number or specific signature) with known, corresponding exploit modules in the Metasploit framework.

### 4.3 AI-Powered Security Analysis Module

#### 4.3.1 Description and Priority

This module is AutoPent's key differentiator, providing the intelligent synthesis that transforms raw technical data into actionable security insights. It uses a locally hosted **LLaMA 3.1** Large Language Model to process all the findings from the reconnaissance and scanning stages. The AI engine generates high-level summaries of the target's security posture and provides clear, context-aware remediation recommendations. Operating entirely locally, this feature ensures that sensitive vulnerability data never leaves the user's machine.

**Priority: High.** The benefit is transformative, making complex security data accessible to non-experts and significantly accelerating the analysis and reporting time for professionals. The penalty for failure is a loss of the product's primary unique selling proposition. The cost of implementation is higher due to the complexities of prompt engineering and model integration, but the operational risk is low.

#### 4.3.2 Stimulus/Response Sequences

This module requires **no direct user input**. It is automatically triggered once the Vulnerability Scanning module has completed its tasks and generated the final JSON output of all findings. The system feeds this structured data into the LLaMA 3.1 model. In response, the AI **generates a plain English summary** of the most critical vulnerabilities, their potential business impact, and a series of suggested steps for remediation.

#### 4.3.3 Functional Requirements

- **REQ-11: Constrained AI Token Generation.** All prompts sent to the **LLaMA 3.1** model must be configured with a max_tokens limit of **800** or less. This technical constraint is necessary to manage memory usage, ensure a fast response time, and prevent the model from generating overly long or irrelevant output.
- **REQ-12: Context-Rich AI Prompting.** The system must dynamically construct a detailed prompt for the AI. This prompt must include a condensed

summary of all key findings, including reconnaissance data (technologies identified) and a list of discovered OWASP vulnerabilities, to provide the AI with sufficient context to generate a holistic and accurate analysis.

- **REQ-13: Plain Text AI Output.** The AI model must be instructed to return its analysis in **plain text only**. The output must not contain any code, JSON, or other structured formatting that could interfere with its direct inclusion in the final report.
- **REQ-14: Non-Technical Remediation Language.** The prompt engineering shall guide the AI to generate remediation suggestions that are clear, actionable, and largely non-technical. The goal is for a developer or system administrator to understand the required actions without needing to be a security expert.

### 4.4 Report Generation Module

#### 4.4.1 Description and Priority

This module produces the final, tangible deliverable of the AutoPent workflow. It compiles all data from the preceding modules—reconnaissance, scanning results, and the AI-powered analysis—into a single, professional, and easy-to-read document. The report is generated in both HTML for easy viewing in a browser and as a downloadable **PDF** for formal distribution and archiving.

**Priority: High.** The benefit of this module lies in its ability to effectively communicate the value of the entire security assessment to the end-user. The penalty for its failure is high, as without a report, the results of the scan are inaccessible. The implementation cost and risk are both very low.

#### 4.4.2 Stimulus/Response Sequences

The user can trigger this module manually by **clicking a "Generate Report" button** in the GUI after an analysis is complete. Alternatively, the system can be configured to automatically trigger this module as the final step in the workflow. In response, the system builds the structured report from a template, populates it with the collected data, and saves the final output as a **.pdf** file.

#### 4.3.3 Functional Requirements

- **REQ-15: Two-Stage Report Rendering.** The system must first render all report content into an **HTML** document. This allows for flexible styling and templating. It will then use the **pdfkit** library to convert this final HTML document into a PDF.
- **REQ-16: Preserve Code Formatting.** To ensure that any technical details, code snippets, or command examples are displayed correctly, the HTML template must use <pre> tags to wrap these sections, preserving their original formatting and whitespace in the final PDF.
- **REQ-17: Timestamped File Storage.** All generated reports must be stored in a dedicated /reports directory within the application's folder structure. Filenames must be automatically generated with a timestamp to ensure uniqueness and chronological organization (e.g., Report-example.com-20250918-183000.pdf).

- **REQ-18: Optional User-Defined Filename.** The system shall provide an option for the user to specify a custom filename for the report before it is generated, overriding the default timestamped name.

## 5. Other Nonfunctional Requirements

This section details the nonfunctional requirements (NFRs) that define the quality attributes, operational characteristics, and overall performance of the AutoPent system. These requirements specify *how* the system should perform its functions, rather than *what* it should do.

### 5.1 Performance Requirements

Performance is a critical attribute for ensuring the system is efficient and usable in real-world scenarios.

- **Overall Scan Duration:** The system must complete a full, end-to-end workflow—including reconnaissance, a standard vulnerability scan, and AI-based report generation—within a **7 to 10-minute** timeframe. This benchmark is defined for mid-tier hardware systems (e.g., a machine with 8GB RAM and an NVIDIA RTX 3050 GPU) and is crucial for ensuring the tool is practical for regular, iterative security testing.
- **AI Analysis Latency:** The LLaMA 3.1 model must generate its analysis and remediation suggestions within **15 seconds** per analysis block after receiving the consolidated scan data. This ensures the AI analysis phase feels responsive and does not become a significant bottleneck in the user's workflow.
- **System Concurrency:** On appropriately provisioned hardware (e.g., a server with a high-end GPU and sufficient RAM), the system must be capable of handling up to **5 simultaneous scan and analysis sessions** without significant performance degradation. This is a key requirement for IT service providers and larger security teams who need to conduct parallel assessments.
- **Offline Performance Parity:** The system's core scanning and report generation performance must remain consistent in a fully offline environment. There shall be no dependency on external cloud APIs or network latency for these primary functions, ensuring reliable and predictable performance regardless of internet connectivity.

### 5.2 Safety Requirements

As a powerful security tool, AutoPent must operate under strict safety protocols to prevent accidental damage or misuse.

- **Non-Exploitation by Default:** The system is designed for assessment, not attack. It must not perform any actual exploitation of discovered vulnerabilities. Any code related to exploitation must be commented out or disabled by default and require deliberate, explicit user action to be enabled.
- **Data Exfiltration Prevention:** Under no circumstances shall the system automatically upload or transmit logs, scan results, or reports to any external server. All generated data must be stored locally on the user's machine by default to ensure complete data sovereignty and privacy.
- **Confirmation for High-Impact Modules:** Any module or function that is

considered potentially destructive or high-impact (such as those that interact with Metasploit) must require explicit user confirmation via a command-line prompt before execution. This acts as a critical safety interlock against unintended actions.
- **Process and Environment Isolation:** When running on a shared server or multi-tenant environment, the system's processes must be properly contained. They must not be able to access or interfere with other users' files, processes, or environments on the same machine, which can be achieved through containerization or strict user permissions.

### 5.3 Security Requirements

This section outlines the requirements for securing the AutoPent application itself against unauthorized access and tampering.

- **Secure Credential Storage:** All user-provided API keys (for Shodan, BinaryEdge, etc.) must be stored in a secure manner, such as in a restricted-access configuration file or as environment variables. Credentials must never be hardcoded in the application's source code.
- **Encryption of Sensitive Results:** The system shall provide an option to encrypt the final report files locally, particularly if they contain sensitive data like discovered credentials or detailed vulnerability information. Access to the encrypted report shall be protected by a user-configurable password.
- **User Authentication Layer:** If the system is deployed with a web-based GUI, it must include a mandatory user authentication mechanism (username and password). The system must enforce strong password policies and store user credentials securely using industry-standard hashing and salting techniques.
- **Audit Logging:** To ensure accountability, especially in team environments, the system must maintain audit logs. These logs shall track key events, including user logins, the targets of scans, and the specific commands or modules that were executed, along with timestamps.
- **Internal Security Principles:** The application's own web interface (if deployed) must be developed in compliance with **OWASP Top 10** security principles to protect it from common web vulnerabilities such as Cross-Site Scripting (XSS) and SQL Injection.

### 5.4 Software Quality Attributes

- **Portability:** The application must be highly portable, designed to run on all major modern operating systems, including common Linux distributions, Windows 10 and newer, and macOS. This will be achieved by avoiding platform-specific code and providing platform-appropriate installation methods.
- **Reliability:** The system must produce highly accurate and consistent reports. The tolerance for errors in data parsing and processing from scanner output to the final report shall be less than **2%**. The system should be stable and not crash during a standard scan cycle.
- **Maintainability:** The codebase must be highly modular. The logic for

reconnaissance, scanning, AI analysis, and reporting must be separated into distinct files or packages. This modular architecture is essential for facilitating easy updates, bug fixes, and future feature extensions.
- **Usability:** The system must be usable by both technical and non-technical audiences. This is achieved through a dual-interface approach: a comprehensive command-line version for experts and an optional, guided GUI for users who are less familiar with security tools.
- **Testability:** Each major module of the application must be designed to be independently testable. This allows for the creation of targeted unit tests, which are crucial for ensuring the quality and reliability of each component in isolation and for enabling automated testing in a CI/CD pipeline.

## 6. Other Requirements

This section covers other miscellaneous requirements, dependencies, and rules governing the system.

### 5.5 Business Rules

- **User Authorization:** The initiation of any scan must be restricted to authorized personnel only, enforced through the system's authentication and access control mechanisms.
- **Report Traceability:** Every report generated by the system must be embedded with metadata, including a timestamp of its creation and the user ID of the individual who initiated the scan, to ensure full traceability.
- **Ethical AI Recommendations:** The AI analysis module must be constrained to never suggest illegal or offensive actions. Its recommendations must remain strictly within the context of defensive security and ethical hacking principles.
- **Software Licensing:** The software will be governed by a licensing agreement that is one-time per organization. The agreement will restrict the resale, modification, or redistribution of the software unless explicitly permitted.

### 6.1 System and Legal Prerequisites

- **LLM Dependency:** The core AI functionality is dependent on the local availability of the **LLaMA 3.1** model checkpoint file. Users will be required to download this file (approximately 4GB) during the initial setup process as a prerequisite for using the AI analysis features.
- **Legal Use Disclaimer:** The software must display a prominent disclaimer upon first use and within its documentation. This disclaimer will state that the tool is intended for authorized security testing and ethical hacking purposes only, and that the user is solely responsible for its legal and ethical use.
- **Deployment Flexibility:** The system shall offer multiple deployment modes to suit different environments. This includes direct installation via a CLI or GUI installer, as well as a pre-configured **Docker container** for easy, isolated deployment on servers.

- **User Documentation:** A comprehensive set of documentation, including a deployment guide and a detailed user manual, will be bundled with the software to guide users through setup and operation.
- **Internationalization Support:** While the initial release will default to English for all report outputs and UI text, the application shall be architected for future internationalization. All user-facing strings shall be stored in external resource files rather than being hardcoded, allowing for straightforward translation into other languages in future versions.

## Appendix A: Glossary

This glossary defines key terms and acronyms used throughout the AutoPent Software Requirements Specification to ensure a common understanding.

**API (Application Programming Interface)** A set of rules, protocols, and tools that allows different software applications to communicate with each other. AutoPent uses APIs to gather data from external reconnaissance services like Shodan.

**AutoPent** The name of the software product described in this document. It is an integrated software framework designed to automate the penetration testing workflow by orchestrating various security tools and leveraging a Large Language Model for intelligent analysis and reporting.

**CLI (Command-Line Interface)** A text-based user interface used to run programs by typing commands into a terminal. The CLI is AutoPent's primary interface, designed for technical users and automation scripting.

**CVE (Common Vulnerabilities and Exposures)** A system that provides a reference-method for publicly known information-security vulnerabilities and exposures. A CVE ID (e.g., CVE-2021-44228) provides a standard identifier for a specific flaw.

**Exploit** A piece of code, a sequence of commands, or a method that takes advantage of a software **vulnerability** or security flaw to cause unintended or unanticipated behavior on a computer system.

**GUI (Graphical User Interface)** A visual-based user interface that allows users to interact with electronic devices through graphical icons and visual indicators, as opposed to a CLI. AutoPent's optional GUI is designed for ease of use.

**JSON (JavaScript Object Notation)** A lightweight, text-based, human-readable data-interchange format. AutoPent uses JSON as the standard internal format for passing structured data between its different modules.

**LLM (Large Language Model)** An advanced type of artificial intelligence trained on vast amounts of text data to understand, generate, and summarize human language. In this project, **LLaMA 3.1** is the specific LLM used to analyze technical scan results and produce plain-English summaries.

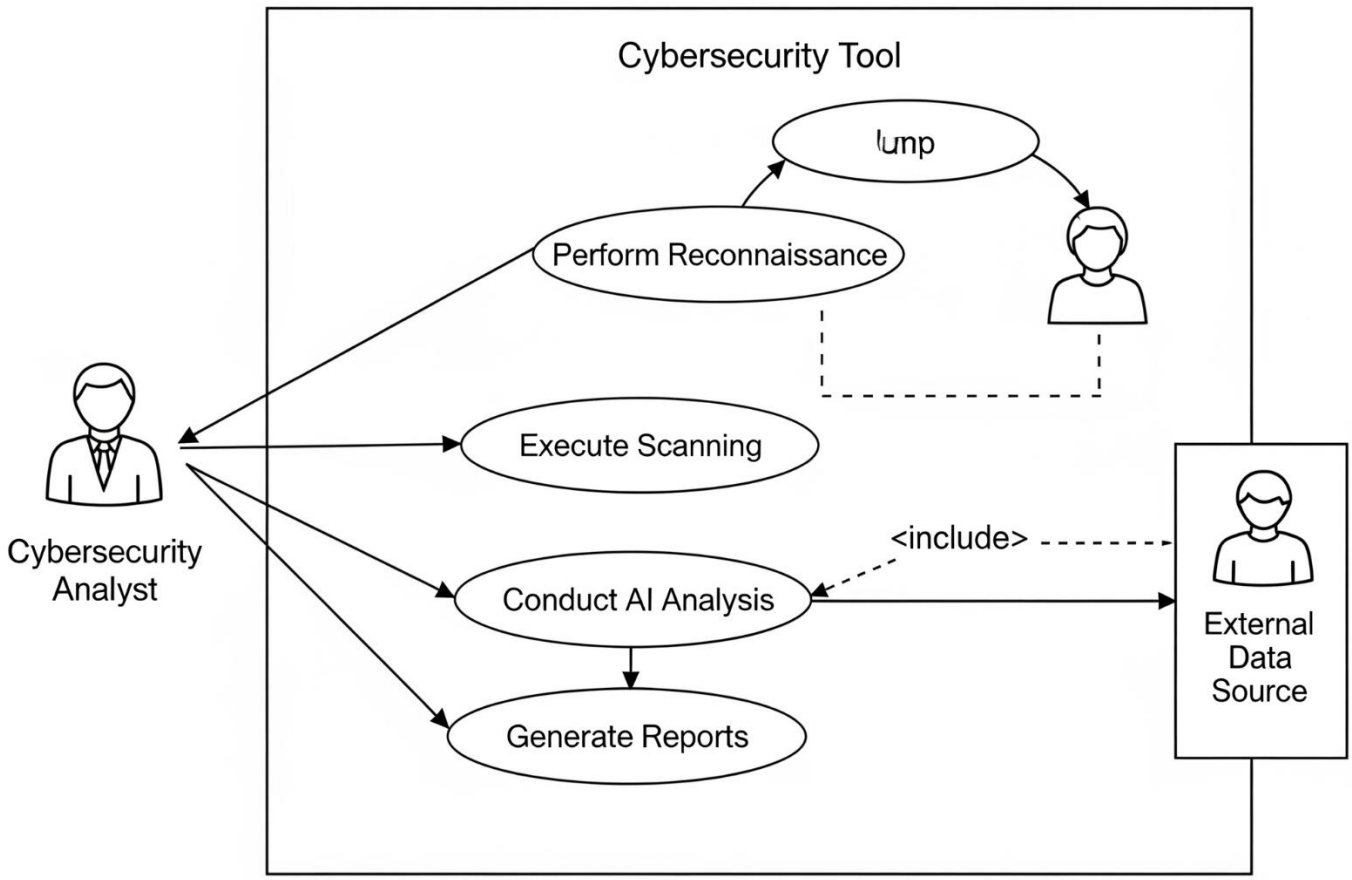
**OSINT (Open-Source Intelligence)** The practice of collecting and analyzing data from publicly available sources to produce actionable intelligence. The **Reconnaissance** module heavily relies on OSINT techniques.

**OWASP (Open Web Application Security Project)** A non-profit foundation dedicated to improving the security of software. The "OWASP Top 10" is a widely recognized awareness document representing a broad consensus about the most critical security risks to web applications.

**Penetration Test (Pentest)** An authorized, simulated cyberattack performed on a computer system to evaluate its security. The goal is to identify and fix security weaknesses before a malicious actor can exploit them. AutoPent is designed to automate this process.

**Reconnaissance** The initial phase of a penetration test, focused on gathering information about a target system. This can be **passive** (using public sources like OSINT) or **active** (directly probing the target with techniques like port scanning).

**Vulnerability** A weakness, flaw, or misconfiguration in a system's design, implementation, or operation that could be exploited by a malicious actor to compromise the system's confidentiality, integrity, or availability.


Cybersecurity Tool
lump
Perform Reconnaissance
Execute Scanning
Conduct AI Analysis
Generate Reports
<include>
Cybersecurity Analyst
External Data Source

## Appendix C: To Be Determined (TBD) List

This appendix lists items that have been identified during the requirements gathering process but are not yet finalized. These items are acknowledged as necessary for the project but require further investigation, analysis, or business decisions before they can be specified in detail. Each item is assigned an ID for tracking purposes.

### TBD-01: Final Large Language Model (LLM) Selection

- **Description:** This item concerns the final selection of the core LLM for the AI Analysis module. While the current baseline for development and testing is **LLaMA 3.1**, the rapidly evolving landscape of open-source models necessitates a final evaluation before release. The primary alternative under consideration is a model from the more recent **Mistral 7B** family.
- **Resolution Criteria:** The final decision will be based on a formal trade-off study comparing the models on several key metrics:
  - **Performance:** Accuracy and relevance of the generated security analysis.
  - **Hardware Footprint:** VRAM and system RAM requirements for efficient inference.
  - **Inference Speed:** The time taken to generate a complete analysis.
  - **Licensing:** A thorough review of the models' licenses to ensure compliance with commercial use.
- **Status:** A decision will be made after the completion of the prototyping phase, where both models will be benchmarked with real-world scan data.

### TBD-02: Detailed UI/UX Specifications for GUI

- **Description:** While the requirement for an optional **Graphical User Interface (GUI)** is established, the specific design, layout, and user experience (UX) flows are yet to be determined. This includes the visual design and interactive behavior of the frontend application.
- **Resolution Criteria:** This will be resolved upon the completion of a dedicated UI/UX design phase. The deliverables from this phase will include:
  - **Wireframes:** Low-fidelity structural layouts of all screens.
  - **Mockups:** High-fidelity visual designs and color palettes.
  - **Interactive Prototypes:** Clickable prototypes demonstrating the user workflow.
- **Status:** This process will begin once the core backend features are stable to ensure the UI accurately reflects the system's capabilities.

**TBD-03: Enterprise Licensing and Support Packages**

- **Description:** The pricing model and terms for enterprise-level customers are currently undefined. This includes potential offerings for large-scale deployments, dedicated support, or custom feature development.
- **Resolution Criteria:** The business and sales teams will define a comprehensive enterprise strategy, which will outline:
  - **Pricing Tiers:** Details on licensing models (e.g., per-user, per-instance, site-wide).
  - **Support Level Agreements (SLAs):** Guaranteed response times for technical support.
  - **Customization Options:** Packages for custom branding or feature integrations.
- **Status:** These business decisions will be finalized closer to the product's official launch date.

**TBD-04: Containerization Deployment Specifications**

- **Description:** The requirement to support deployment via containers is confirmed, but the exact technical specifications for the container images and orchestration are TBD.
- **Resolution Criteria:** The DevOps team will create and finalize the deployment artifacts, which will include:
  - **Dockerfile:** An optimized Dockerfile for building a secure and efficient production image.
  - **Docker Compose:** A docker-compose.yml file to simplify local and single-server deployments.
  - **Kubernetes (K8s) Manifests:** Potential creation of Helm charts or K8s YAML files for scalable cluster deployments.
- **Status:** This will be addressed during the release engineering phase of the project lifecycle.

**TBD-05: Future Language and Localization Support**

- **Description:** The initial version of AutoPent will be released in **English** only. The decision to add support for other languages (internationalization and localization) is pending.
- **Resolution Criteria:** The decision to localize the product for specific regions will be based on post-launch data, including:
  - **Market Demand:** Analysis of sales data and user demographics.
  - **User Feedback:** Direct requests from the customer base for language support.
  - **Business Case:** A cost-benefit analysis for entering new linguistic markets.
- **Status:** While the initial architecture will be designed to be "localization-ready" (i.e., no hardcoded strings), the actual translation effort is a future consideration.