Requirements Document

Version 1

4/16/2025

Team Name: Cyber Recon

Sponsor: HighViz Security LLC

Team Mentor: Veerendernath Surendernath Komala

Team Members: Zachary Garza, Sean Weston, Jared Kagie, Christian Butler

Accepted as baseline requirements for the project

For The Client: _____ Date: _____

For the Team: _____ Date: _____

Table of Contents

Introduction

Statistics show that 60% of small businesses shut down six months after a cyberattack. In the US alone, the average cost of a data breach is currently $9.44 million. The majority of companies lack in-house capabilities to scan and prioritize vulnerabilities since they are understaffed and underfunded. This deficiency has made the majority of companies over-reliant on third-party cybersecurity professionals like HighViz Security LLC.

HighViz possesses the skills to discover and document vulnerabilities with automatic scanners like Nessus. Although these scans detect a wide array of issues, they also produce false positives, they give irrelevant information, and there is just too much information—manual evaluation being highly inefficient and time-consuming. Cyber Recon has a solution that utilizes artificial intelligence in automating vulnerability scanning and prioritization.

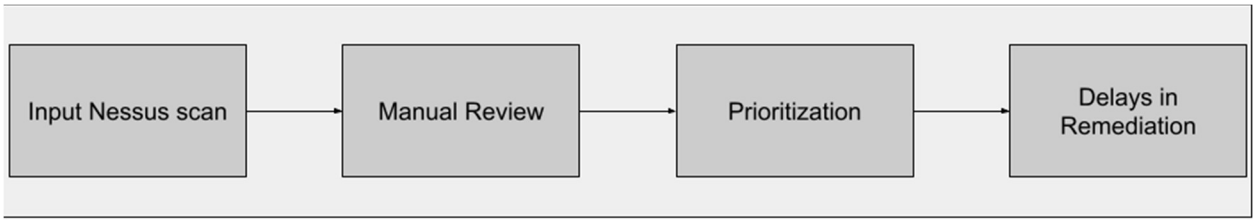
In the dynamically changing threat landscape of today, the imperative to refresh cybersecurity tactics is clearer than ever. With high-speed technology advancements have arrived mysterious cyberattacks aimed at even the smallest vulnerabilities, risking organizations financially and reputationally. Using artificial intelligence and deep analytics, the proposed solution will translate raw vulnerability information into actionable guidance for speeded-up decision-making and enhanced risk mitigation. This preemptive approach not only enhances the overall security position but also ensures that resources are directed to the most critical issues first.

Also, the incorporation of machine learning and natural language processing in the assessment process is a paradigm shift for vulnerability management. The ability of the system to scan complex Nessus scan files, provide context to information with up-to-date threat intelligence, and prioritize risks judiciously enables security teams to achieve greater accuracy and efficacy. This local deployment hybrid architecture, tailored precisely for MacBook Pro M2 hardware with safe cloud collaboration, is a union of cutting-edge technology and user-centric functionality that delivers scalability as well as data privacy for small, medium, as well as large enterprises.

This spec document outlines the architecture of an AI-enabled tool that scans Nessus files, applies machine learning and natural language processing to analyze vulnerabilities, adds external threat feeds, and creates a prioritized list of threats. The focus is on installing this solution locally on MacBook Pro M2 laptops used by members of the HighViz team with a facility for secure cloud collaboration. This blended method delivers efficiency, scalability, and data privacy.

Problem Statement

Today's vulnerability scanners like Nessus produce masses of valuable security information, but in most cases, this information is overwhelmed by huge amounts of noise. Large data sets created consist of thousands of entries with low-risk discoveries and false positives filling many of the entries. So, human analysts are left spending significant amounts of time manually reviewing scan outputs. This not only delays critical decision-making but also creates a climate where highest-priority threats might be overlooked due to the din of irrelevant information.



Main issues with the current process:

- Excessive Time Consumption
- Slow Response Time
- Additional Resource Allocation
- Human Error

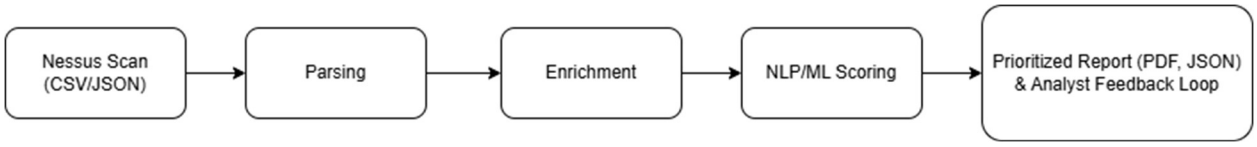
The current manual process has several shortcomings. First, high usage of time is an ongoing issue; analysts spend many hours looking through scan results, which reduces overall operational efficiency. Second, prioritization using CVSS scores alone is not sufficient to measure the real-world impact and exploitability of vulnerabilities. This shortfall leads to resource misallocation, where potentially severe issues are handled with lower priority, and benign outcomes are over prioritized. Furthermore, the presence of false positives results in a noisy and unorganized dataset, which becomes challenging to generate actionable insights.

Resource strain is a major issue. Small teams with limited budgets are overwhelmed by complex data, slowing decision-making and weakening security. Traditional methods fall short, highlighting the need for a smarter, automated approach.

HighViz Security recognizes the pressing demand for a productive solution. The goal is to enhance vulnerability scanning with the use of contextual risk rating powered by artificial intelligence and robust rule-based systems. This enables the system to filter noise, prioritize threats by real-world severity, and ease analyst workload. It must run locally on MacBook Pros with optional secure cloud support for scalable, privacy-respecting deployment. Such a futuristic approach not only ensures improved operation efficiency but also greater precision of prioritization of vulnerabilities in an increasingly complex threat landscape.

Solution Vision

Cyber Recon's product is a hybrid AI-enabled platform that streamlines vulnerability prioritization by transforming raw Nessus outputs into actionable risk insights. The platform parses scan files, enriches them with up-to-date threat intelligence (e.g., KEV, EPSS, Exploit-DB), and applies both natural language processing and machine learning to generate risk scores. This system improves both the speed and accuracy of risk assessments by combining textual context from vulnerability descriptions with structured threat data.



This architecture runs efficiently on Apple M2 hardware, supporting local-first execution and secure optional cloud updates. The system leverages open-source tools including PyTorch, HuggingFace Transformers, and Pandas, ensuring cost-effective deployment and adaptability. Model synchronization with the cloud occurs securely via HTTPS and is optional, enabling centralized model updates and collaboration across deployments while preserving local data control.

The dual-model design—BERT for natural language analysis and Random Forest for structured scoring—enables deeper vulnerability understanding. Rule-based overrides (e.g., KEV auto-prioritization) ensure critical issues are not missed. Reports clearly present prioritized vulnerabilities along with rationale, supporting quick, confident remediation decisions by security teams.

	Before	**After (with Cyber Recon platform)**
Process	Manual, time-consuming scan reviews	Automated, AI-assisted triage and prioritization
Accuracy	Prone to false positives and overlooked threats	Enhanced by enriched context and AI-driven scoring
Speed	Hours to days for triage	Minutes on local hardware for 10,000+ entries
Scalability	Limited by team size and manual capacity	Horizontally scalable and containerized for cloud future
Security	Risk of cloud dependence	Local-first model with optional secure cloud sync

Function Requirements

1. FR1 – Nessus File Parsing

Scenario:
As a security engineer, I need to ensure that Nessus scan outputs in CSV and JSON formats are properly ingested, parsed, and normalized, so that I can effectively analyze vulnerabilities and risk factors.

Overview:
The platform must ingest, and process Nessus scan outputs provided in both CSV and JSON formats. The parser should account for schema differences across Nessus versions and extract critical fields for risk evaluation.

Key Details:

- **CSV Files:**
  - **Variability:** Handle differences in header names (e.g., "plugin_id" vs. "pluginID") using configurable mapping rules.
  - **Validation:** Check each row for required fields (plugin ID, vulnerability title, CVSS score, CVE identifiers) and log any discrepancies.
- **JSON Files:**
  - **Normalization:** Flatten nested structures to extract key fields (vulnerability title, affected hosts, CVE IDs, CVSS scores).
  - **Filtering:** Exclude unnecessary fields to streamline downstream processing.
- **Error Handling:**
  - Validate file integrity; throw meaningful errors (e.g., missing CVE data in a specific record) and implement robust exception handling.

2. FR2 – Severity Normalization

Scenario:
As a developer, I need to normalize raw CVSS scores into consistent severity labels so that tools and analysts can easily interpret and act on the data accordingly.

Overview:
The system must normalize severity levels by mapping numerical CVSS scores to standard severity labels.

Key Details:

- **Mapping Logic:**
  - Define thresholds: CVSS 9.0–10.0 as Critical, 7.0–8.9 as High, 4.0–6.9 as Medium, and below 4.0 as Low.
- **Flexibility & Error Management:**
  - Allow threshold values to be adjusted through configuration.
  - Handle missing or non-numeric values by assigning a default category or flagging them for review.

3. FR3 – CVE Enrichment

Scenario:
As a vulnerability engineer, I want to enrich CVE records with metadata from the NVD so that I can provide the most complete and up-to-date context for risk evaluation.

Overview:
Enhance parsed vulnerability data with metadata from the National Vulnerability Database (NVD) to improve the quality and depth of vulnerability intelligence.

Key Details:

- **Data Integration:**
  - Enrich each record using CVE IDs to fetch vulnerability summaries, CWE classifications, updated CVSS metrics, and patch information.
- **Modes of Operation:**
  - **Real-Time API:** Enable immediate queries with error and timeout handling.
  - **Bulk Download:** Support offline enrichment by scheduling periodic downloads and validating file integrity via checksums.
- **Fallback Procedures:**
  - If enrichment fails, log errors and optionally use the latest available enrichment data

4. FR4 – EPSS Integration

Scenario:
As a security analyst, I need to have predictive exploitation scores incorporated into the vulnerability data so that I can assess the likelihood of exploitation and prioritize issues on real-world risks.

Overview:
This system incorporates the Exploit Prediction Scoring System (EPSS) to enrich each CVE with a probabilistic risk score, enhancing threat prioritization based on exploit likelihood.

Key Details:

- **Mechanisms:**
  - **Local CSV Lookups:** Use pre-downloaded EPSS data with efficient indexing on CVE IDs.
  - **Live API Queries:** Support real-time data fetches with built-in retry and rate-limit strategies.
- **Default Handling:**
  - Specify default EPSS values if no score is found or flag the record for further review.
- **Data Appending:**
  - Normalize and attach EPSS scores to vulnerability records to assist in composite risk calculations.

5. FR5 – KEV Cross-Reference

Scenario:
As a vulnerabilities manager, I need to automatically flag vulnerabilities listed in the catalog so that I can prioritize threats with known exploits and respond more quickly to real-world risks.

Overview:
This system cross-references vulnerabilities against the CISA Known Exploited Vulnerabilities (KEV) catalog to elevate risk prioritization.

Key Details:

- **Automation:**
  - Regularly download and update the KEV feed (daily/weekly) into a local database.
- **Cross-Reference Logic:**
  - For each CVE, if a match is found in KEV, mark the record with a "KEV flag" and automatically increase its risk level.
- **Audit Trail:**
  - Log all cross-referencing events for transparency and compliance.

6. FR6 – NLP-Based Risk Inference

Scenario:
As a threat analyst, I need an NLP model to interpret CVE descriptions and estimate risk or exploitability so that I can flag high-risk vulnerabilities.

Overview:
Leverage natural language processing with a fine-tuned BERT model to analyze vulnerability descriptions and predict exploitability or risk level.

Key Details:

- **Model Training:**
  - Fine-tune a pre-trained BERT model using labeled historical CVE descriptions. Train the model to interpret key phrases (e.g., "public exploits exist").
- **Processing Pipeline:**
  - Tokenize and pre-clean descriptions; output both a categorical risk label and a numerical confidence score.
- **Robustness:**
  - Implement fallback handling for cases where the model inference fails or returns ambiguous results.

7. FR7 – Structured ML Scoring

Scenario:
As an engineer for security, I need a machine learning model to evaluate vulnerability data and output a risk score so that I can quickly prioritize issues based on data insights.

Overview:
This system leverages a structured ML model, such as Random Forest or XGBoost, to analyze vulnerability characteristics and produce a numerical risk assessment.

Key Details:

- **Features:**
  - Utilize inputs such as CVSS scores, attack vectors, privileges required, EPSS scores, KEV flags, and CWE tags.
  - Normalize numerical inputs and encode categorical variables appropriately.
- **Output & Explainability:**
  - Generate either a numerical risk score (e.g., 0–100) or direct risk labels (High/Medium/Low), along with feature importance insights for transparency.

8. FR8 – Composite Risk Score

Scenario:
As a threat analyst, I need a risk scoring system that combines outputs of both NLP and structured ML models so that I can make faster, more confident decisions using a single, interpreted risk score.

Overview:
This system combines outputs from natural language processing and structured machine learning models into a single score, providing both standardized scaling and easy to understand explanations.

Key Details:

- **Fusion Techniques:**
  - Integrate outputs using methods like weighted averages, stacking via logistic regression, or rule-based fusion.
- **Normalization:**
  - Ensure the composite score is standardized on a predefined scale (e.g., 0–100 or categorical such as Low, Medium, High, Critical).
- **Explanation Module:**
  - Generate explanations detailing how inputs from each model contributed to the final score.

9. FR9 – Risk-Based Output Generation

Scenario:
As a risk team leader, I need clear output from the AI in both human-friendly and machine-consumable formats so that I can quickly assess threats, inform stakeholders, and integrate findings into operational workflows.

Overview:
This system generates comprehensive risk reports that are both readable for analysts and executives and understandable for automated systems, enabling fast decision making and simplistic integration.

Key Details:

- **Output Formats:**
  - Generate PDF reports with executive summaries and detailed sections; produce structured JSON and CSV files.
- **Content:**
  - Include vulnerability names, affected hosts, risk levels, CVE summaries, EPSS scores, KEV flags, and AI justification details.
- **Integration:**
  - Ensure seamless compatibility with ticketing systems and security dashboards.

10. FR10 – Local Threat Intelligence Database

Scenario:
As a data engineer, I need a reliable local database to store and manage vulnerability data, feeds, AI model outputs, and feedback so that I can have secure storage and full traceability for logging and analysis.

Overview:
This system uses a local SQLite database to serve as a centralized, secure storage for vulnerability data and human feedback, supporting both performance needs and traceability.

Key Details:

- **Schema Design:**
  - Create tables for CVE records, enrichment feeds (NVD, KEV, EPSS), AI predictions, and analyst feedback.
- **Performance & Security:**
  - Index key fields for quick queries; enforce role-based access control (RBAC) and optionally encrypt sensitive fields using SQLCipher.
- **Audit Logging:**
  - Record all modifications and updates to support traceability and forensic analysis.

11. FR11 – Incremental Data Updates

Scenario:
As a systems engineer, I need to automate the retrieval and validation of threat intelligence feeds so that I can ensure the latest data is always available without risking integrity or losing important context.

Overview:
This system automates the scheduled updating of external security feeds (NVD, EPSS, KEV) ensuring data is the newest, previous data backed up for emergencies, and all changes are logged, all while maintaining integrity through validation mechanisms.

Key Details:

- **Scheduling:**
  - Use a cron-compatible CLI or scheduler to fetch data at predetermined intervals.
- **Integrity & Backup:**
  - Validate downloads via checksums; back up older data versions before updating.
- **Audit Logs:**
  - Maintain logs of all update events for future audits and troubleshooting.

12. FR12 – Analyst Feedback Loop

Scenario:
As a security analyst, I need an intuitive interface to review and correct AI-generated recommendations so that I can provide feedback that improves the model's accuracy.

Overview:
This system enables the validation and refining of AI outputs through a dedicated interface, capturing feedback that directly informs model of improvements and incorporates retraining.

Key Details:

- **Interface Options:**
  - Offer both a secure web interface and a CLI for submitting feedback.
- **Feedback Capture:**
  - Record annotations with CVE/plugin IDs, user identifiers, timestamps, and brief explanations.
- **Model Integration:**
  - Use the feedback to drive periodic retraining, ensuring that the models evolve based on real-world validations.

13. FR13 – Rule Enforcement System

Scenario:
As a risk analyst, I need to override AI-generated outputs based on custom company policies so that I can ensure critical cases are flagged accurately, even when the model's predictions fall short.

Overview:
This system integrates a flexible ruling definition that can override AI outputs based on predefined conditions, allowing for human-in-the-loop control and real-time policy enforcement without interrupting system uptime.

Key Details:

- **Rule Definition:**
  - Use human-readable formats (such as YAML or Python) for defining override rules (e.g., "flag all KEV entries as Critical").
- **Dynamic Updates:**
  - Allow rules to be added or modified on the fly without downtime.
- **Conflict Resolution:**
  - Implement predefined precedence and conflict resolution mechanisms for overlapping rules.

14. FR14 – Cloud Model Synchronization

Scenario:
As a security analyst, I need to synchronize local models with a private cloud server so that I can keep the models current while ensuring data privacy, version control, and auditability.

Overview:
This system securely manages the upload and download of machine learning models between local environments and a designated private cloud, using secure communication and detailed logging to ensure safe synchronization.

Key Details:

- **Secure Transmission:**
  - Use HTTPS with robust authentication (tokens or certificates) for model uploads and downloads.
- **Versioning & Anonymization:**
  - Strip sensitive details before uploading; maintain version control to support rollbacks if needed.
- **Logging:**
  - Log all synchronization events to ensure auditability and compliance.

15. FR15 – End-to-End Automation

Scenario:
As a cybersecurity analyst, I want to automate the processing of risk data through a structured pipeline, so that I can generate consistent, auditable reports with minimal manual effort.

Overview:
A CLI-based pipeline orchestrates multiple phases—from parsing input files to producing final risk reports—allowing customizable execution and robust integration into CI/CD workflows.

Key Details:

- **Pipeline Structure:**
  - Develop a master CLI tool or script that organizes the workflow into distinct phases:
    - File Parsing
    - Enrichment
    - Risk Inference (NLP & ML)
    - Composite Score Generation
    - Report Output
- **Customizability & Robustness:**
  - Support command-line parameters for risk thresholds, output directories, and verbosity settings.
  - Include detailed logging and error handling at each stage to ensure repeatability and facilitate integration into CI/CD pipelines.

Performance (Non-Functional) Requirements

To ensure seamless and effective operation in resource-demanding cybersecurity environments, the performance requirements of the vulnerability prioritization system are established along various dimensions. These non-functional requirements are processing speed, inference latency, resource usage, database query responsiveness, boot time, error tolerance, scalability, and execution security. The following are the criteria and supporting considerations for achieving these performance objectives.

PR1 – Processing Speed

Objective:

The system must be able to parse and process an entire Nessus file containing up to 10,000 vulnerability entries in five minutes on a MacBook Pro M2 with 16GB RAM. This will help with FR1.

Implementation Considerations:

- I/O Efficiency: Minimize file reading and parsing operations by utilizing asynchronous I/O methods and buffered data processing. Since the initial loading and preprocessing operation lays the foundation for subsequent operations, it is important to use efficient file reading libraries that don't use a lot of memory when handling large files.
- Preprocessing Pipelines: Leverage multi-threaded or multi-process methods wherever feasible. For example, splitting the dataset into chunks and processing them in parallel can really reduce the data normalization time, error checking time, and initial formatting time.
- Profiling and Optimization: Use performance profiling tools (such as Python's cProfile, Py-Spy) during development. Identify bottlenecks at critical stages—such as parsing nested JSON arrays or normalizing CSV fields—then optimize these code sections with more efficient data structures (such as arrays or dictionaries) and using compiled extensions if necessary.
- Resource Scheduling: Reduce unnecessary background services at runtime and use caching mechanisms to avoid duplicate processing when dealing with repetitive or similar datasets.

PR2 – Model Inference Time

Objective:

The goal is to ensure that both the Natural Language Processing (NLP) and structured Machine Learning (ML) models can compute and return a risk score for each individual vulnerability in under 100 milliseconds. Achieving this rapid inference time is critical for enabling large-scale batch processing—specifically, processing 10,000 vulnerability entries in under 2 minutes on a GPU-accelerated Apple M2 system. This performance benchmark directly supports functionality described in FR6, FR7, FR8, FR9, FR12, and FR13, making it essential for meeting the broader system requirements.

Implementation Considerations:

- Efficient Batching: To maximize throughput and minimize redundant computation, inference routines should be designed to process vulnerabilities in batches instead of one at a time. This approach helps capitalize on parallelism provided by modern hardware, significantly reducing overall processing time. Efficient batching also minimizes the cost associated with repeatedly loading model weights into memory.
- GPU Acceleration: The models should be trained and optimized using GPU-friendly frameworks such as PyTorch or TensorFlow. Leveraging GPU resources can drastically reduce inference latency. In addition, applying advanced techniques such as mixed-precision training or model quantization can help further reduce computational demand, allowing models to operate faster with minimal loss of accuracy.
- Latency Monitoring: To ensure system performance remains within acceptable bounds, real-time latency monitoring should be embedded into the production environment. This system will automatically log and track instances where inference time exceeds the 100-millisecond threshold. If such events occur frequently, automated responses—such as adjusting batch sizes, reallocating resources, or triggering fallback models—should be initiated to preserve overall performance and reliability.
- Edge Case Handling: Some inputs may be unusually complex or difficult for the model to process within the target latency. These edge cases should be handled with care. Strategies include routing such cases to a separate processing thread to avoid blocking the main pipeline or flagging them for manual review if delays exceed acceptable thresholds. Graceful degradation in these situations ensures system robustness without compromising throughput for standard cases.

PR3 – Memory Usage

Objective:

Maintain maximum RAM usage at 2GB under typical circumstances, despite working with huge data sets and intricate models. Most closely relates to FR15.

Implementation Considerations:

- Memory Profiling: Use intense memory profiling as a portion of the development process through tools such as Valgrind, memory_profiler for Python, or the built-in debugging capabilities in the choice programming language. Monitor memory usage during normal and peak operations to ensure adherence to the 2GB limit.
- In-Memory Data Processing: Minimize data structures' overhead. For instance, we prefer streaming data processing methods that don't require loading entire files in memory simultaneously.
- Garbage Collection Tuning: Tune the runtime's garbage collection (GC) settings aggressively to free objects promptly, especially for languages like Python where GC behavior can be tweaked for more aggressive memory cleanup.
- Cache Management: Implement cache control mechanisms that either flush ephemeral data stores periodically or apply least-recently-used (LRU) policies to prevent memory bloat from highly accessed but non-prioritized data.

PR4 – Database Query Latency

Objective:

Ensure any query of the local threat intelligence database (e.g., cross-referencing CVE data with KEV or EPSS datasets) completes within 50 milliseconds. This corresponds with FR2, FR3, FR4, and FR5.

Implementation Considerations:

- Indexing Strategies: Organize the database schema with proper indexing of significant columns, such as CVE IDs, timestamps, and foreign keys linking threat intelligence datasets.
- Efficient Joins: Optimize SQL queries and design join strategies with little overhead, potentially leveraging in-memory databases or caching layers for frequently executed queries.
- Benchmarking: Run regular performance benchmarks with real queries under a variety of load conditions. Use these benchmarks to continually tune query performance and adjust indexes as the dataset grows.
- Query Optimization Tools: Use database native optimization tools (e.g., SQLite's EXPLAIN command) to identify and correct slow queries.

PR5 – Boot Time

Objective:

The application should fully initialize and be ready for user interaction within 15 seconds of launch. This most closely relates to FR15.

Implementation Considerations:

- Startup Sequencing: Order the loading of critical components during startup to facilitate immediate use of core features. Delay less important tasks (like background updates or diagnostics) until the main interface is available.
- Lazy Loading: Employ lazy loading in modules or libraries not immediately required are loaded only when actually used for the first time. This reduces program effort at startup.
- Optimized Dependency Management: Minimize external dependencies and use compiled binaries where possible to prevent startup overhead.
- Startup Testing: Occasionally test boot times in varied environments and optimize the code path executed during the initial boot process. Document the configuration requirements backing the fastest possible boot times.

PR6 – Error Resilience

Objective:

The system must handle corrupted or truncated Nessus files, with verbose error messages and recovery mechanisms to continue processing valid records. This relates to FR1.

Implementation Considerations:

- Exception Handling: Implement a strict exception handling mechanism that catches file I/O errors, data validation failures, and parsing anomalies at each stage of the data processing workflow. When a corrupted record is encountered, log an error with sufficient context to facilitate debugging.
- Fallback Strategies: Use fallback strategies such as skipping faulty records without terminating the whole process. Maintain a record of skipped entries so these can be reviewed and corrected as necessary.
- User Notifications: Have a user notification system that warns of critical errors in a manner that is actionable and clear. This should include recommendations for data cleansing or external verification in case of repeated issues.
- Self-Healing Capabilities: Investigate adding automated recovery capabilities that attempt to repair minor data inconsistencies (e.g., missing closing braces in JSON files) and log the repairs undertaken.

PR7 – Scalability Baseline

Objective:

The system should support future cloud deployment despite being designed for local use. Key components must be modular and containerized, with horizontal scaling capabilities. This aligns with FR10, FR11, FR14, and FR15.

Implementation Considerations:

- Modular Design: Organize the system into loosely coupled modules (e.g., ingestion, inference, reporting) with well-defined interfaces to allow independent scaling.
- Containerization: Containerize critical components (e.g., with Docker) for easy deployment. Use orchestration tools like Kubernetes for managing containers.
- Distributed Processing: Design with future support for distributed tools (e.g., Spark, Dask) to handle large data volumes efficiently. Document interface standards for easy integration.
- Load Testing: Perform local load tests to identify bottlenecks and use findings to inform scalable cloud-ready designs.
-

PR8 – Security of Execution

Objective:

At runtime, the platform must avoid storing sensitive data unencrypted, including hostnames and IPs. Temporary files should be securely managed and deleted after use. This aligns with FR15.

Implementation Considerations:

- Data Encryption: Encrypt sensitive data during processing at both the file and application levels using secure libraries.
- Secure File Handling: Open temporary files with restricted permissions and ensure they're deleted promptly after use.
- Audit Trails: Log all sensitive operations, and protect logs with proper access controls to support traceability.
- Compliance Testing: Regularly audit for vulnerabilities and update security practices based on findings.
- Runtime Isolation: Use containers or VMs to isolate runtime memory and prevent leaks from dumps or snapshots. Document these in the security policy.

Environmental Requirements

Requirement	Constraint & Key Tools/Libraries
ER1 – Platform Support	macOS 13+, Apple M2 (Metal, Core ML, MPS)
ER2 – Offline Capability	Pure-local execution (SQLite 3, Python feed-updater scripts)
ER3 – Open-Source Libraries	Permissive-license only (MIT/BSD/Apache 2.0)
ER4 – Local Execution	No cloud; Core ML & Metal inference on-device
ER5 – Storage	≤10 GB total (Zstandard/LZ4 compression)
ER6 – Encrypted Disk	FileVault + SQLCipher + NSFileProtectionComplete

ER1 – Platform Support :

The platform should natively support macOS 13 and later and be fully optimized to run on Apple M2 processors. Native support delivers native integration with macOS-exclusive features and enhances performance by leveraging the hardware features intrinsic to the Apple M2 processor. The solution should also utilize Apple Metal to accelerate computationally intensive operations such as model training and inference. Utilizing Metal not only accelerates these operations by going directly to the GPU but also maintains the platform responsive during real-time analysis. With Apple Silicon optimization, the system runs with increased energy efficiency and speed, which are crucial in the scenario of time-sensitive vulnerability analysis.

ER2 – Offline Capability :

The system should be capable of running completely offline under normal conditions in order to offer maximum security and privacy for data. The central functionality—vulnerability parsing, model inference, and report generation—must be functional locally where there is no internet available. Offline support is necessary in the case of situations where threats to data exfiltration must be minimal or where network utilization is constrained by security limitations. Background updates for critical threat feeds such as CVE, KEV, and EPSS can be run autonomously and are designed particularly for supporting critical air-gapped environments. These updates are planned and controlled so that the system remains in offline integrity while periodically benefiting from data enrichment and threat intelligence enhancements.

ER3 – Open-Source Libraries:

All libraries and tools included in the platform must be open-source and licensed on permissive terms such as MIT, BSD, or Apache 2.0 to promote transparency, flexibility, and community development. This requirement enables developers to have access to the source code in its entirety, modify it as necessary, and benefit from peer review and community contribution. Proprietary APIs or dependencies are permitted only if thoroughly reviewed and personally approved by the project sponsor. By embracing open-source software, the platform minimizes legal liabilities and avoids vendor lock-in, thereby providing a robust and economically sustainable development platform for continued improvement.

ER4 – Local Execution Requirements:

The architecture demands that no cloud-based compute resources should be employed in executing the application. The computational operations—like model inference, data parsing, and storage—should be executed on the local machine. This local execution model is crucial for ensuring that sensitive information will remain within the secure confines of the deployment environment. In practice, all necessary data as well as the necessary models and support libraries must be preloaded locally. This not only reduces latency significantly but also enhances data security by eliminating the risk of transmitting sensitive information over external networks.

ER5 – Storage Requirements

The total disk usage of the system should not exceed 10GB, even with the addition of complete NVD mirrors and enhanced local databases. This is a threshold that requires careful planning of data storage plans and adherence to good data management practices. Compression procedures, for instance, might be utilized to compress older, less recent data to make room on the disks without compromising data integrity or accessibility. Restricting the storage space footprint ensures the platform will continue to be suitable for deployment on moderate storage device offerings while providing maximum performance. Long-term system sustainability will demand effective data cleanup and archiving strategies.

ER6 – Encrypted Disk Access:

Security at the storage level is critical in protecting confidential information. FileVault encryption must be enabled on the deployment devices to protect local drives from intrusion. Also, all databases that store enriched vulnerability information must have optional SQLCipher encryption. In addition, strict adherence to macOS file permission requirements should be observed to restrict access and further protect data stored. This security model with multiple layers ensures that sensitive information is encrypted both in transit and in any temporary storage, hence avoiding potential breaches and conforming to industry best practices in data protection.

Potential Risks:

R1 – Model Bias

AI models used for prioritizing vulnerabilities can be biased if the training data is small or skewed. For example, if less popular CVEs are underrepresented, the model can downrate their risk. This bias can lead to skewed threat assessments that undermine overall security.

Mitigation Strategies:

- Data Augmentation: Supplement the training dataset with synthetic or more real-world examples to cover a broader set of vulnerabilities.
- Rule-Based Overrides: Use rules to trigger alerts on specific vulnerabilities (e.g., the ones in the KEV list) independent of model output.
- Continuous Evaluation: Regularly retrain the model using newer data and performance metrics to detect and rectify bias over time.

R2 – Misclassification

Misclassification is a significant threat because the system can misrank vulnerabilities—overestimating low-risk threats or underestimating high-impact threats. Such errors can cause resource misallocation and increased exposure to threats.

Mitigation Strategies

- Human-in-the-Loop: Have a feedback loop to enable security analysts to edit and validate risk scores, using their input in further model tuning.
- Explainable AI: Provide transparent, understandable results that describe the factors behind each risk score, enabling analysts to verify and trust the automated result.

R3 – Dependency Vulnerabilities

The system relies on a collection of third-party libraries and tools, each of which may have vulnerabilities within them that can be exploited. Such dependency issues can leave the system open to attack even if its code is secure.

Mitigation Strategies:

- Regular Security Scans: Include Bandit or Safety in the development cycle to scan dependencies for known vulnerabilities on a regular basis.
- Dependency Management: Maintain an updated list of all libraries and patch or update them as soon as they are made available to minimize exposure.

R4 – Performance Bottlenecks

Performance bottlenecks could cause the system to slow, particularly on legacy hardware. Inefficiencies in parsing, model inference inefficiencies, or database querying inefficiencies could decelerate critical operations and make timely remediation challenging.

Mitigation Strategies:

- Profiling and Optimization: Profile with tools to identify and correct slow code segments or inefficient queries.
- Load Testing: Perform periodic stress and load testing in order to simulate peak usage so that the system will be kept within acceptable responsiveness limits.
- Resource Monitoring: Use constant monitoring of resource usage and adjust algorithms or hardware configuration as needed.

R5 – Threat Feed Staleness

Accuracy of risk assessment depends on fresh threat intelligence. Unless CVE, EPSS, or KEV feeds are periodically updated, the system may return stale recommendations without considering new threats.

Mitigation Strategies:

- Automatic Schedule Updates: Install provisions for periodic, scheduled updates of all threat feeds, even in air-gapped infrastructures.
- Alert Systems: Install notifications to indicate failed updates or data freshness inconsistencies, necessitating immediate corrective action.

R6 – Cloud Security Breach

Although local execution is promoted by the platform, voluntary integration of the model updates or teamwork in the cloud introduces the danger of cloud security breaches. When the cloud link is compromised, sensitive vulnerability details are revealed.

Mitigation Strategies:

- Secure Transmission: Restrict cloud interaction to HTTPS mutual authentication, whereby data exchange would be encrypted.
- Strict Access Controls: Limit cloud interactions to only what is necessary and perform regular audits to verify that the security posture of the cloud infrastructure is aligned with strict standards.

R7 – Legal Compliance Risk

Failure to comply with licensing agreements or data retention policies may result in significant legal liabilities, financial penalties, and reputational harm. Open-source components necessitate strict compliance with their license terms.

Mitigation Strategies:

- Compliance Audits: Regularly audit the software modules of the platform to ensure all open-source licenses (e.g., MIT, BSD, Apache 2.0) are being implemented correctly.
- Documentation and Policies: Maintain proper documentation of dealing with data, retention policies, and licensing compliance. Internal audits and legal reviews must be scheduled to ensure ongoing compliance.

Project Plan:

The project is structured into six principal phases; each aligned with specific functional specifications and objectives to deliver a working proof-of-concept by week 12. Throughout the project execution, each phase will have weekly internal discussion sessions, sponsor comment sessions, and continuous unit testing to ascertain that each unit meets performance and quality objectives.

Phase 1 (Weeks 1–2): Parser and Data Normalizer Prototyped

In this initial phase, the priority is on developing a prototype of the data parser. The parser will convert both CSV and JSON formats of Nessus files and normalize them to determine main data elements. This milestone paves the way for the rest of the project by giving confidence that raw data is correctly transformed and formatted for analysis.

Phase 2 (Weeks 3–5): Model Selection and Training

In the second phase, the emphasis is on selecting and training both structured machine learning models and NLP models. The team will tune a BERT model for text-based inference, compare various algorithms, and apply models such as Random Forest for structured data. The team will conduct weekly model performance tests against specified latency and accuracy goals.

Phase 3 (Weeks 6–7): Risk Prioritization Logic Implemented

This stage combines the trained models with risk scoring logic. The system will use both AI-based risk scores and rule-based overrides to generate a prioritized list of vulnerabilities. The goal is to make sure that important vulnerabilities are properly flagged, considering context from both the NLP and structured ML methods.

Phase 4 (Weeks 8–9): Report Generator and Local DB Developed

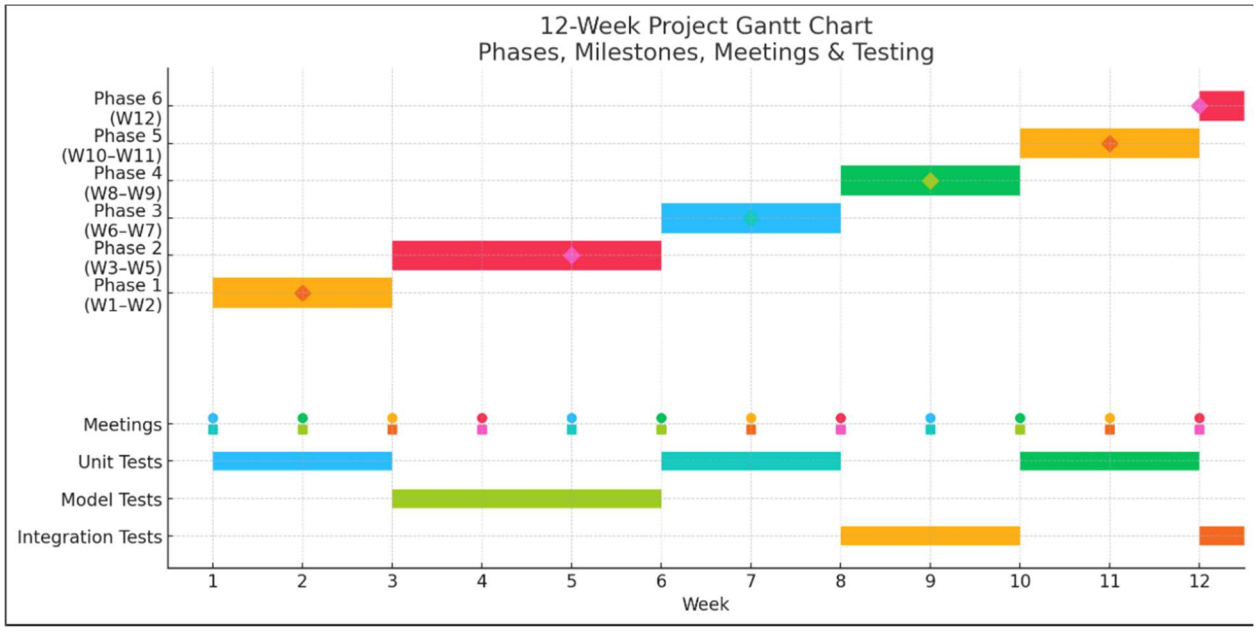
Once risk scores and data are set up, focus is directed towards implementing the report generator and local database. The report generator will consume prioritized vulnerabilities and generate human-readable outputs (e.g., PDF reports and CSV/JSON exports), while the local database will store raw and enriched data for fast lookups and audit trails. This integration will be required for operational usability and scalability in the future.

Phase 5 (Weeks 10–11): UI + CLI Interfaces Finalized

User interfaces become the focus of attention as both graphical user interface and command-line interface are finished. These interfaces will allow analysts to use the system, view reports, and provide feedback. Iterative design refinement and usability testing are extremely crucial activities in this stage.

Phase 6 (Week 12): Full System Integration and Testing

The final phase is for the integration of all components into one system. End-to-end testing and rigorous integration will make sure that the system meets all the functional and non-functional requirements. The final deliverable includes complete documentation, a working proof-of-concept, and a demonstration for key stakeholders.



Conclusion

The AI-Powered Vulnerability Mapping System developed by Team Cyber Recon addresses an acute need in modern cybersecurity. In today's rapidly evolving world, data breaches and threats are increasing at unbelievable rates. Companies, like HighViz Security, require cutting-edge tools with the ability to significantly reduce the workload for resource-limited in-house teams. Traditional vulnerability analyses approaches are often slow from manual overhead and error prone. The project responds directly to these challenges, offering a transformative approach that not only improves detection and prioritization but also alleviates the inefficiency and inaccuracy of traditional analyses that leave critical threats unmediated during false positives and long, manual review cycles.

In this document, we have demonstrated an end-to-end solution leveraging a hybrid AI approach of natural language processing and structured machine learning. The system is able to successfully parse and normalize very large quantities of Nessus scan data sets as well, as enrich them with real-time threat intelligence, and rank the vulnerabilities in a meaningful and complete manner. This evaluative nature of the AI is backed by the training from previous data sets to offer both a faster analysis to the traditional counterpart as well as accurate review that strays from human error.

Understanding of hardware of companies like HighViz Security is a focus for the system due to the constraints of such limitations. Their Apple M2-based systems are used company-wide and have been the defined software that is well known. By running natively on the hardware with the possibility of secure cloud collaboration, the platform meets stringent performance, security, and budget demands. Each aspect of the system—from parser prototyping all the way through model training, risk prioritization, and final report generation—has been carefully designed to provide actionable intelligence that guides quicker, smarter remediation efforts.

Implementation into the pre-existing system is a solution to the issues plaguing these in-house and resource-limited teams. It promises revolutionary benefits through the reduction of analyst overhead, improved customer satisfaction, and a scalable technical foundation for future artificial intelligence advances in security. The developments presented in this report not only demonstrate good project planning and technological discipline but also chart a clear path to deployment and long-term system evolution. We feel that this solution will not only solve current issues in vulnerability management today but also lay the groundwork for continued improvement and evolution in the ever-changing cybersecurity landscape.

Glossary

- **CVSS** – Common Vulnerability Scoring System
- **CVE** – Common Vulnerabilities and Exposures
- **KEV** – Known Exploited Vulnerabilities (by CISA)
- **EPSS** – Exploit Prediction Scoring System
- **BERT** – Bidirectional Encoder Representations from Transformers
- **ML** – Machine Learning
- **NLP** – Natural Language Processing
- **SQLCipher** – An SQLite extension for database encryption
- **CLI** – Command Line Interface
- **UI** – User Interface

Appendices

Appendix A – Sample Nessus Export Snippet (CSV & JSON)
Appendix B – Risk Score Calculation Example
Appendix C – Performance Benchmark Results
Appendix D – Training Dataset Description
Appendix E – Technology Stack and License Table
Appendix F – Screenshots of UI Mockup
Appendix G – Risk Prioritization Rule File Template