Faculty of Computers and Data Science

Alexandria National University

Integration of Penetration Testing and Threat Modeling for Continuous Monitoring

TIBSA Platform

Department of Cybersecurity

Graduation Project 2025–2026

Prepared by:

Nadine Rasmy Soliman	2205203
Alaa Hassan Melook	2205214
Yumna Medhat Anter	2205231
Rana Ashraf Abdelaziz	2205019
Mahmoud Amr Zaghloula	2205055
Abdelrahman Hisham Elmoghazy	2205032

Supervised by:

Dr. Essam Shabaan

Academic Year: 2025-2026

Abstract

Cybersecurity assessments in modern organizations often suffer from fragmented practices, where threat modeling, penetration testing, and continuous monitoring operate in isolation. This disconnection leads to inefficient risk management, redundant vulnerabilities, and delayed responses to evolving cyber threats.

This project proposes an intelligence-driven framework that integrates **Threat Modeling**, **Penetration Testing**, and **Threat Intelligence** to create a unified, risk-based security assessment process. By combining the predictive capabilities of threat modeling with the practical validation of penetration testing, the framework ensures that testing efforts are aligned with real business risks. The incorporation of threat intelligence enables adaptive and context-aware updates, allowing security assessments to evolve with the current threat landscape.

The proposed solution includes automated threat-to-test mapping, dynamic scope definition, centralized documentation, and continuous feedback loops for ongoing improvement. Expected outcomes include enhanced testing efficiency, better risk prioritization, and improved collaboration among security teams. Ultimately, this project aims to bridge the gap between design-time threat analysis and real-world defense validation, providing a modern, intelligence-driven approach to proactive cybersecurity management.

Acknowledgments

We would like to express our sincere gratitude to Dr. Essam Shabaan for his invaluable guidance and support throughout this project. His expertise and encouragement have been instrumental in shaping our research direction and achieving our objectives.

We would also like to extend our heartfelt thanks to Prof. Dr. Tamer Helmy, Dean of the Faculty, for his continuous support and for providing a motivating academic environment.

Our sincere appreciation goes to our academic advisor, Dr. Reem Essameldin Ebrahim, for her guidance, valuable feedback, and encouragement throughout the research process.

We are also grateful to the Faculty of Computers and Data Science at Alexandria National University for providing the necessary resources and environment for conducting this research.

Finally, we thank our families and colleagues for their unwavering support and motivation during the completion of this project.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
ATT&CK	Adversarial Tactics, Techniques & Common Knowledge
CAPEC	Common Attack Pattern Enumeration and Classification
CDN	Content Delivery Network
CI/CD	Continuous Integration/Continuous Deployment
CTI	Cyber Threat Intelligence
CVE	Common Vulnerabilities and Exposures
DDoS	Distributed Denial of Service
DFD	Data Flow Diagram
DREAD	Damage, Reproducibility, Exploitability, Affected users, Discoverability
MISP	Malware Information Sharing Platform
ML	Machine Learning
OWASP	Open Web Application Security Project
SIEM	Security Information and Event Management
STRIDE	Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege
TIBSA	Threat-Intelligence Based Security Assessment
WAF	Web Application Firewall

Chapter 1

Introduction

1.1 Introduction

This project focuses on integrating Threat Modeling with Penetration Testing to develop a more risk-driven and intelligence-oriented security assessment framework. By combining the predictive power of threat modeling with the practical validation of penetration testing, the proposed integration aims to help organizations prioritize real business risks, improve testing efficiency, and enhance their overall security posture.

Furthermore, the project incorporates Threat Intelligence into this integration to make the process more context-aware and adaptive. Threat intelligence provides insights into real-world attack patterns, emerging threats, and adversary tactics—allowing the threat model to evolve dynamically in line with the current threat landscape.

This unified approach ensures that both threat modeling and penetration testing are driven by up-to-date, intelligence-backed data, leading to more relevant, actionable, and continuous security assessments.

1.2 Problem Statement

In many organizations, Threat Modeling, Penetration Testing, and Continuous Monitoring are performed as isolated processes, resulting in fragmented and inefficient security management.

- Threat modeling identifies potential attack vectors but lacks real-world validation.
- Penetration testing confirms vulnerabilities but does not leverage predictive threat insights.
- Security assessments that exclude threat intelligence often fail to capture emerging attack trends and adversary behaviors.

This disconnection leads to delayed detection, misprioritized vulnerabilities, and repeated exposures—leaving systems vulnerable to preventable attacks.

Therefore, a unified framework is needed to integrate Threat Modeling, Penetration Testing, Threat Intelligence, and Continuous Monitoring, creating an adaptive and intelligence-driven security process.

1.3 Motivation

As cyberattacks become increasingly sophisticated and dynamic, traditional one-time assessments are insufficient to ensure ongoing protection.

Integrating threat intelligence with threat modeling and penetration testing improves both accuracy and timeliness in threat detection and validation.

This integration allows security teams to:

- Focus on high-impact business risks.
- Simulate real-world adversary tactics.
- Build smarter and adaptive defense strategies.

For cybersecurity students and professionals, this project bridges the gap between predictive analysis and practical validation, providing valuable hands-on experience aligned with modern organizational security practices.

1.4 Project Objectives

1. Integrate Threat Modeling with Penetration Testing to create a risk-driven, efficient, and intelligence-informed security assessment process.
2. Focus testing on high-impact assets and critical attack vectors based on real business risks.
3. Translate threat scenarios into actionable penetration test cases for targeted validation.
4. Establish a continuous feedback loop where pentest results refine the threat model.
5. Strengthen the organization's security posture through collaboration and risk-based decision-making.

1.5 Expected Solutions

1.5.1 Threat-Driven Testing Framework

Penetration testing activities will be derived directly from the threat model, ensuring focused testing on high-risk and business-critical areas.

1.5.2 Automated Threat Mapping

Each identified threat will be automatically mapped to corresponding test cases, attack vectors, and mitigation strategies, allowing faster and more structured analysis.

1.5.3 Dynamic Scope Definition

The testing scope will dynamically adjust based on system updates and the evolving threat model—preventing scope creep while maintaining complete coverage of critical assets.

1.5.4 Centralized Documentation

All threat models, pentest results, vulnerabilities, and remediation actions will be stored in a unified repository to ensure traceability, auditability, and collaboration.

1.5.5 Risk Prioritization and Visualization

A visual risk scoring tool (e.g., heatmaps, risk matrices) will be incorporated to help stakeholders prioritize vulnerabilities based on impact, likelihood, and exploitability.

1.5.6 Continuous Feedback and Improvement Loop

Findings from penetration testing will feed back into the threat model, maintaining a continuous improvement cycle that adapts to new threats.

1.5.7 Collaboration and Role Integration

Encourages communication between developers, architects, and security testers through shared threat models, diagrams, and testing objectives—bridging the gap between design and defense.

1.5.8 Comprehensive Reporting

Generates integrated reports that link each vulnerability to:

- Its originating threat scenario
- The associated risk level
- The recommended mitigation strategy

1.6 System Scope

The project aims to deliver a framework and process model (not a commercial platform) that demonstrates how Threat Modeling, Penetration Testing, and Threat Intelligence can be seamlessly integrated.

It focuses on:

Designing the integration workflow between these components.

Implementing sample use cases to validate the concept.

Providing risk visualization and documentation tools to support security decision-making.

1.7 Out of Scope

The following features and tools are not part of the main project scope, but may serve as potential extensions or future enhancements:

1.7.1 Development of a Website or Web-Based Service

A web-based service that provides analytical tools such as:

Malware Analyzer

URL Analysis

IP Analysis

Port Scanner

Hash Analyzer

Password Strength Checker

1.7.2 Dashboards that Visualize

Dashboards that visualize:

- Attack trends over time (e.g., via graphs)
- Statistical data such as:

  - Most common malware types
  - Ratio of malicious vs. clean files
  - Total number of files analyzed

These functionalities represent a separate web application layer, which is outside the project's core research and implementation scope.

1.8 Expected Outcomes

- A proof-of-concept framework that demonstrates how integrating threat modeling, penetration testing, and threat intelligence enhances organizational security.
- Documentation and reports showing the relationship between modeled threats and verified vulnerabilities.
- A risk-driven testing process adaptable to evolving cyber threats.
- A collaborative workflow improving communication among security and development teams.

1.9 Conclusion

This project proposes a unified, intelligence-driven security assessment framework that bridges predictive analysis and practical validation. By integrating threat modeling, penetration testing, and threat intelligence, the approach empowers organizations to prioritize real risks, respond faster, and continuously strengthen their defenses in an ever-evolving threat landscape.

Chapter 2

Literature Review

2.1 Introduction

This chapter reviews existing research and methodologies related to threat modeling, penetration testing, and predictive security approaches. The literature is organized into three key areas: (1) traditional threat modeling frameworks, (2) AI- and automation-based methodologies, and (3) integration of threat modeling with penetration testing. Each work is analyzed by its problem focus, proposed method, results, limitations, and relevance to this project.

2.2 Foundational Threat Modeling Approaches

The study by *Dong Seong Xu et al.* introduced an automated security test generation approach based on formal threat models, which aimed to transform theoretical threat modeling into practical automated testing to strengthen system reliability and security validation. Their work addressed the challenge of bridging the gap between high-level threat modeling and automated, executable security testing. Traditional security testing approaches were often manual, error-prone, and limited in scalability, especially when applied to complex software systems. To overcome these limitations, Xu et al. proposed a formal, model-based framework that automatically generated and executed security test cases derived from Predicate/Transition (PrT) nets, an advanced form of Petri nets representing both control flow and data constraints. The framework followed a structured process of threat modeling, reachability analysis, Model–Implementation Mapping (MIM), and automated test generation. Implemented in the ISTA tool, the approach was evaluated on Magento and FileZilla Server, where about 95% of the generated attack paths were executable and achieved a 90% mutant kill rate, demonstrating high test effectiveness. Although it required expert modeling effort and maintenance of PrT nets and MIM mappings, the study provided a rigorous and automated approach to model-

based security testing, advancing the field toward autonomous and verifiable security assurance. [1]

In contrast, Marback et al. proposed a threat model-based approach to security testing that addressed the challenge of generating effective security test cases directly from threat models rather than relying on traditional, ad hoc testing methods. They introduced a structured framework that integrated Data Flow Diagrams (DFDs) and the STRIDE methodology to identify potential threats within software systems. These threats were then expanded into Threat Trees, representing hierarchical attack paths that could be systematically transformed into executable test cases. The approach was implemented and evaluated on real web applications, revealing security flaws that conventional functional testing had missed. The results demonstrated improved test coverage and the ability to detect multi-step attacks by linking threat models to executable tests. However, the study noted that combinatorial explosions might occur as system complexity increased, making large-scale automation challenging. Overall, the research provided a systematic connection between threat modeling and security testing, enhancing the precision and coverage of vulnerability detection, and offering a threat model-based testing technique that bridged threat identification and test case creation, allowing developers to systematically detect and address vulnerabilities early in the software development lifecycle. [2]

Expanding this concept, Palanivel and Selvadurai presented a risk-driven testing approach that integrates risk assessment with threat modeling focused on optimizing security testing by prioritizing test cases based on quantitative risk assessment. They proposed a risk-driven framework that integrates Extended Finite State Machines (EFSMs) with STRIDE-based threat modeling to identify and evaluate potential security risks. Each threat is assigned a Risk Value = Possibility $\times$ Impact, allowing the system to rank and select test cases according to their severity and likelihood. The methodology was validated on multiple system models, such as ATM and LMS, showing that prioritizing high-risk transitions reduced test cases by around 20% without compromising coverage. This demonstrates improved efficiency and resource utilization in testing. However, the study acknowledged that the risk estimation process can be subjective and that the evaluation was primarily analytical rather than experimental. Overall, the paper presents a practical and cost-effective approach to risk-based security testing, aligning testing priorities with system vulnerabilities. [3]

Additionally, Rahim et al. conducted a comprehensive study applying threat modeling to critical Water Grid Systems (WGS), emphasizing the value of structured risk analysis in anticipating and mitigating cyberattacks targeting essential infrastructure. In their paper titled "Risk Analysis of Water Grid Systems Using Threat Modeling" (Journal of Physics: Conference Series, 2022), Fiza Abdul Rahim, Norziana Jamil, Zaihisma Che Cob, Lariyah Mohd Sidek, and Nur Izz Insyirah Sharizan investigated cybersecurity risks

within WGS—recognized as a vital national infrastructure—through a systematic threat modeling and risk assessment framework. Their methodology involved asset identification, access point analysis, threat categorization using the STRIDE model, risk scoring via the DREAD framework, and the development of mitigation strategies. Utilizing the Microsoft Threat Modeling Tool, the researchers modeled the WGS architecture, including its SCADA, PLC, and IoT-based layers, and identified 154 potential threats distributed across STRIDE categories: 30 Spoofing, 15 Tampering, 22 Repudiation, 46 Denial of Service, and 39 Elevation of Privilege. Through DREAD-based risk evaluation, the study found that the most severe threats were Tampering (score 14), Denial of Service (score 13), and Repudiation (score 12). These were prioritized for mitigation through stronger authentication, encryption, redundancy, application hardening, and enhanced logging mechanisms. The research concluded that threat modeling effectively anticipated cyber risks in WGS architectures, providing actionable insights for prioritizing cybersecurity controls. The authors recommended further exploration of integrated STRIDE–DREAD approaches with other assessment tools to enhance the resilience of cyber-physical water systems against emerging threats. [4]

In the healthcare sector, Alozie investigated the use of threat modeling to improve cybersecurity in medical environments. The study highlighted that a formalized modeling process could identify vulnerabilities threatening patient data and healthcare operations. Alozie (2024) examined the critical role of threat modeling in securing digital healthcare ecosystems and argued that healthcare's unique sensitivity to data breaches and ransomware required systematic, design-stage security analysis. The paper surveyed and compared three established threat modeling frameworks—PASTA, STRIDE, and Attack Trees—and positioned threat modeling as a foundation for maintaining confidentiality, integrity, and availability (CIA), ensuring compliance (e.g., GDPR, HIPAA, PCI DSS), and improving operational resilience. Methodologically, Alozie described the components of effective threat modeling for healthcare, including the identification of assets and entry points, enumeration of threats, vulnerability assessment, and risk scoring. The study emphasized the integration of internal (logs, incident history, scans) and external threat intelligence feeds, and advocated automation for continuous model updates, real-time monitoring, and dynamic policy adjustment. For detection and prediction, the paper highlighted the value of ML/AI techniques—such as anomaly detection, behavioral analysis, and predictive analytics—to detect deviations and anticipate attacks, especially in IoT and cloud-enabled medical devices, while also recommending simple, checklist-style approaches (e.g., Privacy Impact Assessments) for non-security experts.

A structured comparative analysis was presented, where STRIDE was recommended for healthcare due to its systematic, component-level coverage (S, T, R, I, D, E) and ease of adoption. PASTA was recognized for its comprehensive, risk-centric modeling but was noted as complex to implement, whereas Attack Trees were useful when sys-

tem boundaries were clearly defined but less effective for unknown or evolving systems. The paper also discussed practical concerns, such as user authentication, third-party integrations, biometric and behavioral authentication, and common risks like password guessing, session disclosure, and DoS attacks. A risk-rating table was included to prioritize mitigation strategies. Limitations and implementation challenges were acknowledged, including the need for continuous staff training, high-quality threat intelligence, performance overheads, and the absence of standardized regulatory frameworks for connected medical devices. The author concluded that adopting STRIDE-based, regularly updated threat models—supported by automation and ML when feasible—substantially strengthened healthcare cybersecurity posture and informed practical mitigation and compliance strategies, providing valuable insights for integrating systematic threat modeling into this project's threat-driven penetration testing and risk assessment framework. [5]

2.3 Automated and AI-enhanced Threat Modeling

Moreover, Granata and Rak conducted a comparative study of open-source automated threat modeling tools. Their systematic analysis assessed multiple frameworks and provided recommendations for standardizing the evaluation of these tools. Following the Kitchenham et al. (2009) SLR methodology—planning, conducting, and reporting—they examined 466 papers and refined them to 55 key studies that addressed two research questions: the methodologies that enabled automation and the open-source tools that supported them. The findings were grouped into four categories: modeling techniques, threat classification, threat selection, and tools. Data Flow Diagrams (DFDs) and graph-based models were the most common due to their suitability for automation, while STRIDE and LINDDUN dominated classification for their comprehensive coverage of security and privacy threats. Most automated approaches combined DFDs with STRIDE-based classification using label- or relationship-based methods to identify risks. The authors compared four major open-source tools—Microsoft Threat Modeling Tool, OWASP Threat Dragon, SLA-Generator, and PyTM—by applying them to a WordPress e-commerce site. Microsoft's tool demonstrated the highest automation (88 threats with mitigations), OWASP Dragon identified 31 threats, SLA-Generator mapped threats to NIST SP 800-53 controls, and PyTM detected 91 detailed threats using CAPEC and MITRE data. The study concluded that no single tool was universally superior; rather, effectiveness depended on user goals, technical expertise, and system complexity. [6]

On the other hand, Shin, Elkins, Larson, Perez, and Cameron proposed the Actionable Intelligence-Oriented Cyber Threat Modeling Framework, which integrated cyber threat intelligence (CTI) with practical, data-driven decision-making. In their 2022 study, Shin, Elkins, Larson, Perez, and Cameron developed this framework to tackle the problem of information overload in CTI systems by aligning automated intelligence

processing with operational cybersecurity needs. Their approach combined conceptual modeling with empirical validation through the creation of a prototype system called the Threat Intelligence Modeling Environment (TIME). TIME was designed to automate the correlation between an organization's assets, vulnerabilities, and external threat data using standardized NIST Security Content Automation Protocol (SCAP) formats such as CVE, CVSS, CWE, CAPEC, and CPE. The system continuously collected CTI data from multiple external feeds—like AT&T's AlienVault Open Threat Exchange and IBM's X-Force Exchange—and mapped it to internal asset inventories obtained from network endpoints to produce real-time, actionable intelligence. A proof-of-concept experiment was conducted in a controlled virtual environment containing Windows 10 endpoints and a Linux-based backend developed with PostgreSQL, Python, and Grafana. The evaluation confirmed that the framework successfully achieved automated data gathering, correlation, and threat alert generation without human intervention. The findings showed that TIME substantially reduced analyst workload, accelerated decision-making, and strengthened proactive cyber defense by integrating automation, intelligence correlation, and standardization within a scalable SOAR (Security Orchestration, Automation, and Response) structure. Ultimately, the study demonstrated that this framework effectively bridged the gap between reactive threat modeling and modern anticipatory defense strategies, marking a significant advancement in operational cybersecurity intelligence. [7]

Dekker and Alevizos developed a threat-intelligence-driven methodology that incorporated uncertainty into cyber risk analysis to enhance decision-making accuracy by realistically modeling uncertain threat scenarios. In their work, they proposed TIBSA (Threat-Intelligence Based Security Assessment), a novel methodology aimed at improving cybersecurity by addressing uncertainties in threat assessment through a comprehensive methodological framework. TIBSA leveraged Cyber Threat Intelligence (CTI) to systematically analyze attacker tactics, techniques, and procedures (TTPs), moving beyond traditional hierarchical models like Attack Trees by employing flexible Causal Graphs to represent threats from an attacker's perspective. The methodology consisted of six main steps: (1) understanding the cyber threat landscape using CTI, (2) identifying assets vulnerable to possible, probable, or plausible attacks, (3) determining relevant TTPs using frameworks such as MITRE ATT&CK or CAPEC, (4) applying a scoring model to assess risks while minimizing bias through multi-evaluator inputs, (5) identifying existing security controls including prevention, detection, mitigation, and recovery mechanisms, and (6) evaluating their effectiveness using metrics like prevent, detect, constrain, and recover, followed by a benefit-cost analysis. This approach ensured interoperability across organizational security functions and supported informed decision-making by providing actionable, data-driven recommendations. Moreover, TIBSA addressed both known unknowns (threats with partial information) and unknown unknowns (unforeseen threats), aligning with ISO 27005:2022 standards. It also introduced key concepts such as the Risk

Paradox—where mitigating one risk could increase another—and the Ellsberg Paradox, which reflected a preference for known risks. Due to its structured yet adaptable design, the methodology could be applied in either full-scale or rapid assessments, making it suitable for organizations with varying resource capacities. Future research directions identified by the authors included integrating artificial intelligence to accelerate risk analysis and combining TIBSA with frameworks like NIST CSF to enhance its applicability across diverse cybersecurity contexts. [8]

In another study, Smith et al. proposed a ransomware detection system using Predictive Behavioral Mapping (PBM). Their method employed behavioral analytics to autonomously identify and classify ransomware activity with improved precision. The authors addressed the persistent challenge of detecting ransomware attacks that evaded traditional signature- and behavior-based defense mechanisms. They observed that conventional detection systems were reactive and often failed to recognize polymorphic, obfuscated, and zero-day ransomware variants because they relied on predefined signatures or detected threats only after execution. To overcome these limitations, the authors introduced a novel framework called Predictive Behavioral Mapping (PBM), which integrated machine learning with temporal behavioral analysis to predict malicious intent at an early stage. The PBM methodology consisted of several structured phases: data collection of system- and process-level events such as file access, registry edits, and API calls; feature extraction and encoding to transform these activities into numerical representations; behavioral mapping to analyze inter-event dependencies and identify temporal relationships; and predictive modeling using supervised algorithms like Decision Trees and Random Forests to classify benign and malicious behavior patterns. The framework also enabled autonomous threat identification, allowing real-time ransomware detection without human intervention and marking a shift from reactive to proactive cybersecurity mechanisms. The PBM framework was implemented and evaluated in a controlled Windows virtual environment using both benign and malicious datasets that included real ransomware samples. Experimental results showed that PBM achieved 98.6% detection accuracy and a 1.8% false positive rate, significantly outperforming conventional detection approaches. The study also presented a detailed pseudocode (Algorithm 1) illustrating how event data were processed, mapped, and classified within the PBM pipeline. Moreover, PBM demonstrated strong generalization by detecting previously unseen ransomware variants, confirming its adaptability and predictive efficiency. However, the authors acknowledged several limitations, including the model's dependency on large, diverse datasets and its current focus on ransomware alone. They suggested extending PBM to broader environments such as cloud infrastructures, IoT networks, and mobile ecosystems, and incorporating adaptive learning for better scalability. Overall, this research provided a practical and experimentally validated approach to proactive cyber defense, bridging the gap between traditional threat modeling and predictive detection.

By combining behavioral analytics and machine learning, PBM exemplified how predictive intelligence could enhance autonomous threat modeling and directly supported this project's objective of integrating automated, intelligent detection within the penetration testing and threat analysis lifecycle. [9]

Likewise, Bin Sarhan and Altwaijry presented a comprehensive methodological framework for detecting insider threats through the use of advanced machine learning techniques. The study employed the CERT r4.2 insider threat dataset, which contained over 32 million events from 1,000 users, to simulate realistic organizational environments. Their methodology began with extensive data preprocessing and feature engineering using the Deep Feature Synthesis (DFS) algorithm, which automatically generated more than 69,000 behavioral features for each user. To manage the high dimensionality of the data, they applied Principal Component Analysis (PCA) to reduce the feature space to the most influential variables. In addition, SMOTE was utilized to address class imbalance and improve model generalization. Two categories of models were then tested: anomaly detection models (One-Class SVM and Isolation Forest) and classification models (SVM, Random Forest, Neural Network, and AdaBoost). The results showed that the SVM classification model achieved the best performance, with 100% accuracy, precision, recall, and F1-score, outperforming all other approaches. This methodology demonstrated the effectiveness of combining automated feature synthesis, dimensionality reduction, and data balancing techniques to enhance the accuracy and reliability of insider threat detection systems. [10]

2.4 Integration of Threat Modeling with Penetration Testing

Furthermore, Alharbi et al. conducted an applied methodological study titled "Ethical Hacking: Threat Modeling and Penetration Testing a Remote Terminal Unit" (KTH, 2020), which evaluated the security of a Siemens SICAM CMIC Remote Terminal Unit (RTU) by integrating formal threat modeling—based on Shostack's four-step approach and the STRIDE framework—with practical black-box penetration testing. The study followed a structured workflow encompassing information gathering and network enumeration using Nmap, automated vulnerability scanning of the web interface through OWASP ZAP, Nikto, Nexpose, and SQLMap, and targeted manual testing such as password-cracking attempts, Denial-of-Service (DoS) experiments (e.g., XML Bomb, Billion Laughs, and slow-loris-style tests), and SD-card content analysis and tampering. Threat modeling guided the selection and prioritization of attack vectors, while penetration testing verified their practical exploitability. Findings revealed no exploitable SQL injection vulnerabilities but exposed significant configuration and protocol weaknesses, in-

cluding reliance on TLS 1.0, weak cipher suites (3DES, with exposures to SWEET32 and BEAST), and the absence of essential HTTP security headers (HSTS, X-Frame-Options, X-XSS-Protection, X-Content-Type-Options). Additionally, the system exhibited a reproducible DoS vulnerability requiring manual restart and potential for SD-card tampering or code injection. Hamra concluded that although the RTU demonstrated reasonable robustness against remote cyberattacks, it remained highly susceptible to compromise with physical access, recommending mitigations such as upgrading TLS configurations, enforcing modern security headers, implementing proper certificate management or PKI, and securing SD-card integrity to enhance the device's overall resilience. [11]

Moving toward automation, Chu discussed the automation of penetration testing in his Ph.D. dissertation. The study highlighted how automated tools and AI-based methods improved efficiency, accuracy, and scalability in penetration testing. It focused on integrating intelligent decision-making models, specifically the Belief-Desire-Intention (BDI) architecture, with ontology-based reasoning to create adaptive and efficient security testing systems. Chu emphasized that traditional penetration testing—while effective in identifying vulnerabilities—was often manual, time-consuming, and inconsistent, motivating the need for automation to enhance scalability and repeatability. Following the Penetration Testing Execution Standard (PTES), the research incorporated tools such as Nmap, Metasploit, Nessus, Hydra, and sqlmap to automate processes from reconnaissance to exploitation. The proposed BDI-based model represented the attacker as an intelligent agent capable of dynamic reasoning, where beliefs stored system knowledge, desires defined attack goals, and intentions guided executable actions. This model was enhanced by OntoPT, an ontology formalizing relationships among attackers, vulnerabilities, targets, and techniques, supported by Semantic Web Rule Language (SWRL) for automated inference. Experimental evaluation on Metasploitable2 showed that the automated approach reduced execution time from 179 seconds to 52 seconds, demonstrating that the integration of BDI and ontology-based reasoning produced a more intelligent, adaptive, and real-time penetration testing framework suitable for complex environments such as IoT systems. [12]

Continuing this direction, Huang and Zhu proposed PenHeal, a two-stage Large Language Model (LLM) framework designed for automated penetration testing and optimal vulnerability remediation. The study utilized a comprehensive experimental and analytical methodology to design and evaluate the system. It followed a structured approach that began with the development of two integrated modules: the Pentest Module for automated vulnerability detection and the Remediation Module for generating and prioritizing cost-efficient mitigation strategies. The methodology included the implementation of Counterfactual Prompting, Retrieval-Augmented Generation (RAG), and a Group Knapsack Algorithm to optimize remediation processes under limited resources. The researchers tested the system in a controlled environment using Metasploitable2 as

the target machine and Kali Linux as the attacker host, equipped with tools such as Metasploit and Nmap. To ensure reliability, both GPT-3.5 and GPT-4 models were employed strategically—GPT-4 handled reasoning-intensive tasks (Planner, Executor, Advisor, Evaluator), while GPT-3.5 was used for summarization and data extraction. The evaluation phase employed three key performance metrics: Detection Coverage, Remediation Effectiveness, and Remediation Efficiency, all normalized on a 0–10 scale for quantitative comparison. Experimental results showed that PenHeal improved vulnerability detection coverage by 31%, remediation effectiveness by 32%, and reduced remediation costs by 46% compared to baseline models such as PentestGPT and GPT-4. This methodological framework provided a reproducible and data-driven foundation for assessing the automation potential of LLMs in penetration testing and cybersecurity remediation behaviors. [13]

Similarly, Chauhan analyzed the impact of penetration testing on mitigating insider threats. The study employed a Systematic Literature Review (SLR) methodology to comprehensively examine the role of penetration testing in addressing insider threats. A structured and rigorous process was followed, beginning with an extensive search across multiple academic databases, including IEEE Xplore, ScienceDirect, SpringerLink, ACM Digital Library, and Google Scholar, using predefined keywords such as "Insider Threats," "Penetration Testing," "Ethical Hacking," and "Cybersecurity." From an initial pool of 745 studies, 432 were screened based on title and abstract, 189 underwent full-text review, and finally 64 high-quality papers were included according to strict inclusion and exclusion criteria. The selected studies were assessed for methodological rigor, reliability, ethical compliance, and relevance to the research objectives. Data were extracted to capture study characteristics, research methods, and key findings related to penetration testing's effectiveness in detecting and mitigating insider threats. Both qualitative and quantitative analyses were conducted, including thematic synthesis and meta-analysis, to identify recurring themes such as threat detection, threat mitigation, behavioral modeling, and the integration of penetration testing within broader cybersecurity frameworks. This methodological approach ensured a comprehensive, unbiased, and evidence-based understanding of how penetration testing enhanced cybersecurity resilience against insider threats. [14]

In addition, Sanagana integrated threat modeling with Next-Generation Firewall (NGFW) architectures. The proposed system linked threat intelligence with network defense policies, improving the ability to mitigate network-based attacks dynamically. In this study, Durga Prasada Rao Sanagana (2023) introduced a proactive cybersecurity framework titled "Mitigating Network Threats: Integrating Threat Modeling in Next-Generation Firewall Architecture", which aimed to enhance network defense by embedding threat modeling directly into NGFW systems. The study emphasized that traditional reactive firewalls and static defense mechanisms were inadequate in addressing

modern, dynamic, and multi-vector cyber attacks. To overcome these challenges, the author proposed a structured integration between threat modeling methodologies and NGFW capabilities to anticipate, identify, and mitigate threats before exploitation. The framework combined advanced firewall features—such as Deep Packet Inspection (DPI), Intrusion Prevention Systems (IPS), and Application Awareness—with systematic threat identification and prioritization. It incorporated threat intelligence feeds, risk-driven decision-making, and real-time behavioral analysis to strengthen adaptive and proactive security. The methodology was organized into five major components: (1) developing detailed threat models to map system assets and vulnerabilities, (2) integrating both internal and external threat intelligence sources, (3) automating continuous monitoring and dynamic rule generation, (4) applying AI and machine learning for anomaly detection and predictive analytics, and (5) ensuring compliance with international standards such as GDPR, HIPAA, and PCI DSS through consistent risk assessments. The findings demonstrated that this integration transformed NGFWs from reactive, rule-based systems into intelligent, adaptive defense mechanisms capable of real-time risk prediction and prioritization. Through theoretical validation and experimental analysis, the proposed framework showed improved network resilience and reduced response latency by automating policy adaptation and enabling proactive threat blocking. The integration of AI and continuous automation allowed the system to learn evolving attack behaviors and maintain up-to-date defense strategies. Nonetheless, the study acknowledged several limitations, including the complexity of developing comprehensive threat models, reliance on accurate threat intelligence, and the potential performance overhead caused by continuous data analysis. Despite these constraints, the proposed model represented a significant advancement in network protection, bridging the gap between threat modeling, artificial intelligence, and NGFW architectures. It established a foundation for predictive and self-adaptive cybersecurity systems, directly aligning with this project's objective of integrating automated threat modeling and intelligent penetration testing for proactive and compliant network defense. [15]

Maniraj et al. showcased the practical use of OWASP ZAP for comprehensive web application security testing, demonstrating that open-source automated tools remained essential for effectively identifying and addressing application-level vulnerabilities. In their work, they presented a methodological review of OWASP ZAP (Zed Attack Proxy), an open-source tool designed to enhance web application security by detecting vulnerabilities through systematic and comprehensive testing. OWASP ZAP operated as a proxy server that intercepted and analyzed HTTP requests and responses to identify security flaws such as Cross-Site Scripting (XSS), SQL Injection, and weak session management. Their methodology involved multiple testing techniques, including passive scanning to examine configurations like cookies and SSL/TLS without altering requests, active scanning to simulate attacks, spidering to map all application pages and links, and fuzzing to test

input validation using random data. The proposed process included initializing the ZAP client with an API key and target URL, conducting spidering and active scans, analyzing detected vulnerabilities, and generating detailed reports outlining severity levels and remediation steps. The study also highlighted OWASP ZAP's integration with DevOps pipelines such as Jenkins and GitLab, which enabled security testing to be embedded early in the development process. Experimental results showed coverage rates between 88–94%, with the tool identifying 120–150 vulnerabilities, including 20–35 critical ones, in an average of 25–35 minutes. Although OWASP ZAP proved effective in detecting a wide range of vulnerabilities and supporting modern APIs like RESTful, SOAP, and GraphQL, the authors noted limitations such as false positives and difficulty in detecting business logic errors. To overcome these challenges, they suggested enhancements like AI-driven vulnerability detection, integration with threat intelligence for zero-day threat identification, and support for emerging technologies such as Progressive Web Apps (PWAs) and IoT systems. Overall, the proposed methodology ensured proactive risk reduction, compliance with OWASP standards, and provided practical guidance for developers and security teams to strengthen web application defenses effectively. [16]

2.5 Comparative Summary and Research Gaps

The following table provides a comparative overview of selected research works.

Table 2.1: Comparative Summary of Related Research Works

Author	Description	Techniques	Result
[1] Dong Seong Xu, Barbara S. Chaparro, Xinming Ou, and Prasad Rao	Proposed a formal, model-based framework using Predicate/Transition (PrT) nets to automatically generate and execute security test cases from threat models.	- Formal Threat Modeling - Predicate/Transition (PrT) Nets - Model–Implementation Mapping (MIM) - Reachability Analysis - Automated Test Generation	Achieved 95% executable attack paths and 90% mutant kill rate, showing high test effectiveness.
[2] Shamim Ripon, Farid Ahmed, and Hafiz Md. Hasan Babu	Proposed a structured framework that integrates Data Flow Diagrams (DFDs) and the STRIDE methodology to identify threats, expand them into Threat Trees, and transform them into executable security test cases.	- Data Flow Diagrams (DFDs) - STRIDE Methodology - Threat Trees - Threat-to-Test Transformation	Improved test coverage and detection of multi-step attacks on real web applications, though scalability is limited by combinatorial explosion in complex systems.
[3] Dwaipayan Roy, Pradeep K. Das, and Rajib Mall	Introduced a risk-driven security testing framework combining Extended Finite State Machines (EFSMs) with STRIDE-based threat modeling to prioritize test cases based on quantitative risk assessment (Risk Value = Possibility × Impact).	- Risk-Driven Testing - Extended Finite State Machines (EFSMs) - STRIDE-Based Threat Modeling - Quantitative Risk Assessment - Risk-Based Test Prioritization	Reduced test cases by 20% without loss of coverage, improving testing efficiency and resource utilization, though risk estimation remained somewhat subjective.

Continued on next page

Table 2.1 – continued from previous page

Author	Description	Techniques	Result
[4] Fiza Abdul Rahim, Norziana Jamil, Zaihisma Che Cob, Lariyah Mohd Sidek, and Nur Izz Insyirah Sharizan	Conducted a methodological study applying STRIDE–DREAD-based threat modeling and risk analysis to Water Grid Systems (WGS) using the Microsoft Threat Modeling Tool to identify, rate, and mitigate cybersecurity threats across SCADA, PLC, and IoT layers.	- STRIDE (for threat identification & classification) - DREAD (for risk assessment & prioritization) - Data Flow Diagram (DFD) (for system modeling and data flow analysis) - Countermeasure Mapping (for linking threats to security controls)	Identified 154 threats with highest risks in Tampering (score 14), DoS (score 13), and Repudiation (score 12); proposed targeted mitigations improving cybersecurity resilience in water infrastructures.
[5] C. E. Alozie	Examines the application of STRIDE, PASTA, and Attack Tree threat modeling approaches to strengthen healthcare cybersecurity and regulatory compliance.	- Comparative use of STRIDE, PASTA, and Attack Trees frameworks - Risk assessment and prioritization of vulnerabilities - Integration of threat intelligence and automation for model updates - Application of AI/ML techniques (anomaly detection, predictive analysis) in IoT and cloud-based healthcare systems	STRIDE-based, regularly updated models with automation and ML support offer the best balance of coverage and usability, significantly improving healthcare system resilience and data protection.
[6] Daniele Granata and Massimiliano Rak	The paper reviews various automated threat modeling methods and tools to evaluate how security threats can be efficiently identified and analyzed without manual intervention, aiming to enhance accuracy, scalability, and consistency in the threat identification process.	- Systematic Literature Review (SLR) methodology based on Kitchenham et al. - Data Flow Diagram (DFD)-based threat modeling - Graph-based modeling approaches - STRIDE and LINDDUN frameworks for threat classification - Comparative evaluation of open-source tools (Microsoft TMT, OWASP Threat Dragon, SLA-Generator, PyTM)	PyTM and Microsoft's Threat Modeling Tool demonstrated the highest levels of automation, successfully identifying up to 91 and 88 potential threats respectively, highlighting their effectiveness in minimizing manual analysis and improving threat detection coverage.
[7] Bongsik Shin, Dennis Elkins, Erik Larson, Rafael Perez, and Glen Cameron	Developed the Actionable Intelligence-Oriented Cyber Threat Modeling Framework and implemented the TIME prototype to automate the correlation of organizational assets, vulnerabilities, and external CTI data using NIST SCAP standards (CVE, CVSS, CWE, CAPEC, CPE).	- Threat Intelligence Integration - Threat Modeling - CVE–CPE Mapping - Proactive Defense Approach - Automation of Intelligence Gathering - AI/ML-based Analysis - Data Prioritization and Correlation - SOAR - SIEM Integration - Real-time Monitoring - Vulnerability Analysis - Decision-Making Optimization	Successfully automated CTI data collection and correlation, reducing analyst workload and improving decision-making speed, thus enhancing proactive cyber defense within a scalable SOAR architecture.

Continued on next page

Table 2.1 – continued from previous page

Author	Description	Techniques	Result
[8] Martijn Dekker and Lampis Alevizos	Proposed TIBSA (Threat-Intelligence Based Security Assessment), a six-step methodology integrating Cyber Threat Intelligence (CTI) and causal graph modeling to assess and prioritize cyber risks while minimizing uncertainty and bias, aligned with ISO 27005:2022 standards.	- Cyber Threat Intelligence (CTI) - Causal Graphs - MITRE ATT&CK Framework - CAPEC Framework - Scoring Model - Benefit-Cost Analysis - Metrics-based Evaluation - ISO 27005:2022 Alignment - Risk Paradox - Ellsberg Paradox	Provided a flexible, data-driven framework enhancing organizational ability to anticipate and mitigate both known and unknown threats, improving decision-making and risk management effectiveness.
[9] J. Smith, R. Johnson, and L. Williams	Introduces the Predictive Behavioral Mapping (PBM) framework to detect ransomware before payload execution using machine learning and behavioral analytics.	- Predictive Behavioral Mapping (PBM) integrating Machine Learning (Decision Trees, Random Forests) - Behavioral and temporal analysis of process and system-level events - Feature extraction and encoding for event classification - Real-time detection and autonomous classification in a virtual environment	PBM achieved 98.6% accuracy and 1.8% FPR, proving superior to traditional detection; it enables early, autonomous ransomware identification and suggests future adaptation for cloud and IoT systems.
[10] B. Bin Sarhan and N. Altwaijry	Proposed a machine learning-based insider threat detection framework using Deep Feature Synthesis (DFS), PCA for dimensionality reduction, and SMOTE for dataset balancing on the CERT r4.2 dataset.	- Black-box testing - White-box testing - Gray-box testing - Social engineering - Vulnerability scanning - Network penetration testing - Web application testing - Reporting and remediation techniques	The SVM classification model achieved 100% accuracy, precision, recall, and F1-score in detecting malicious insider behaviours.
[11] Sam Hamra	Conducted an applied methodological study combining Shostack's threat modeling and STRIDE with black-box penetration testing to assess the security of a Siemens SICAM CMIC RTU, using tools like Nmap, OWASP ZAP, Nikto, Nexpose, and SQLMap.	- Threat Modeling, STRIDE Model (for threat classification) - DREAD Model (for risk rating) - Microsoft Threat Modeling Tool (for implementation) - Risk Assessment & Ranking - Countermeasure Development / Mitigation Controls	The report identified major issues like weak TLS 1.0 configuration, missing HTTP security headers, DoS vulnerability, and SD-card tampering risks, recommended upgrading encryption, enforcing headers, and securing physical access.
[12] Guangliang Chu	The paper automates penetration testing using a Belief-Desire-Intention (BDI)-based intelligent agent with ontology reasoning to enable adaptive and efficient security assessments. The agent models attacker behavior, updates its knowledge using OntoPT ontology and SWRL rules, and automates penetration testing across PTES phases.	- Belief-Desire-Intention (BDI) agent-based architecture - Ontology modeling (OntoPT) for knowledge representation - Semantic Web Rule Language (SWRL) for automated reasoning - Integration of BDI model with ontology for Adaptive decision-making - Automation of the Penetration Testing Execution Standard (PTES) phases - Experimental evaluation using the Metasploitable2 environment	The approach reduced testing time from 179s to 52s, demonstrating a significant improvement in efficiency and intelligence through adaptive decision-making and automated reasoning, leading to faster and more accurate penetration testing results.

Continued on next page

Table 2.1 – continued from previous page

Author	Description	Techniques	Result
[13] J. Huang and Q. Zhu	The paper "PenHeal: A Two-Stage LLM Framework for Automated Pentesting and Optimal Remediation" presents a two-phase AI model using GPT-based LLMs with RAG, Counterfactual Prompting, and a Knapsack Algorithm to automate penetration testing and optimize vulnerability remediation.	- Retrieval-Augmented Generation (RAG) (via LangChain) - Multi-agent LLM architecture (Planner / Executor / Instructor / Summarizer / Extractor / Estimator / Advisor / Evaluator) - Role-play prompting - CVE lookup & CVSS estimation/integration - Group Knapsack Algorithm (for remediation selection) - Hybrid LLM deployment (GPT-4 for reasoning-heavy, GPT-3.5 for utilities)	The framework achieved 31% higher detection coverage, 32% better remediation effectiveness, and 46% cost reduction compared to baseline systems like PentestGPT and GPT-4-only models, proving that LLMs can automate and optimize pentesting efficiently.
[14] K. Chauhan	The paper "Insider Threats Mitigation: Role of Penetration Testing" provides a Systematic Literature Review on how penetration testing methods—black-box, white-box, and gray-box—help mitigate insider threats and integrate with other cybersecurity controls.	- Deep Feature Synthesis (DFS) - Principal Component Analysis (PCA) - SMOTE (Synthetic Minority Oversampling Technique) - One-Class SVM (OCSVM) - Isolation Forest (iForest) - Support Vector Machine (SVM) - Random Forest (RF) - Neural Network (NN) - AdaBoost - Grid Search Optimization - k-Fold Cross Validation	The review found that regular and well-structured penetration testing significantly reduces insider threat incidents by proactively detecting vulnerabilities. It emphasizes combining penetration testing with behavior analytics, access control, and employee training for stronger insider threat mitigation.
[15] Durga Prasada Rao Sanagana	Proposes embedding threat modeling directly into Next-Generation Firewalls (NGFWs) to enhance proactive and adaptive network defense.	- Integration of Threat Modeling with Next-Generation Firewall (NGFW) components (DPI, IPS, Application Awareness) - Use of AI-based anomaly detection and predictive analytics - Threat intelligence fusion from internal and external sources - Automation and risk-driven analysis for adaptive defense policies	The integration transforms NGFWs into intelligent, self-learning systems, improving resilience, adaptability, and compliance, though complexity and data-dependence remain key limitations.
[16] Chitra Sabapathy Ranganathan Satheeshkumar Sekar S. P. Maniraj	Presents a methodological review of OWASP ZAP, detailing its systematic approach to web application security testing through passive/active scanning, spidering, and fuzzing, with integration into DevOps pipelines for continuous security assurance.	- Passive Scanning - Active Scanning - Spidering - Fuzzing - HTTP Request/Response Interception - Vulnerability Detection (XSS, SQL Injection, Weak Session Management) - DevOps Integration (Jenkins, GitLab)	Achieved 88–94% coverage, detected 120–150 vulnerabilities (including 20–35 critical) within 25–35 minutes, demonstrating strong efficiency though limited by false positives and difficulty detecting business logic flaws.

2.5.1 Comparative Features Matrix

The following table presents a systematic comparison of key features across available solutions.

Solution	TM	Auto	PT	CM	AI/ML	SDLC	Risk	TI
Microsoft TMT	✓	Med	×	×	×	Ltd	Basic	×
Cigent	✓	High	Ltd	✓	Ltd	✓	✓	✓
ThreatModeler	✓	High	Ltd	✓	×	✓	✓	✓
IriusRisk	✓	High	Via API	✓	Ltd	✓	✓	✓
Threat Dragon	✓	Low	×	×	×	Ltd	Basic	×
PyTM	✓	High	×	Ltd	×	✓	✓	✓
Threagile	✓	High	×	Ltd	×	✓	✓	Ltd
OWASP ZAP	×	Med	✓	✓	Ltd	✓	Basic	Ltd
Metasploit	×	Med	✓	×	×	Ltd	×	Ltd
Nessus	Ltd	High	✓	✓	×	Ltd	✓	✓
Burp Suite	×	Med	✓	Ltd	Ltd	Ltd	Basic	×

Table 2.2: Comparative Features Matrix of Available Solutions

Legend: TM=Threat Modeling, Auto=Automation Level, PT=Penetration Testing, CM=Continuous Monitoring, TI=Threat Intelligence, Ltd=Limited, Med=Medium, ✓=Yes, ×=No

2.5.2 Research Gap Summary

Overall, the reviewed literature demonstrates significant advancements in integrating threat modeling with automated security testing, penetration testing, and AI-driven defense mechanisms. Collectively, the studies highlight a clear shift from manual, static testing toward intelligent, adaptive, and risk-prioritized cybersecurity frameworks.

While approaches such as PrT-net-based automation, STRIDE-DFD threat modeling, and risk-driven EFSM testing improved accuracy and efficiency, they often required high modeling expertise and faced scalability challenges. Similarly, AI-enhanced and CTI-integrated frameworks like TIBSA, TIME, and PenHeal achieved automation and predictive intelligence but remained limited by data quality, computational overhead, and domain specificity. Tools like OWASP ZAP and NGFW integrations showed strong practical applicability but suffered from false positives and limited coverage of complex attack vectors.

These findings reveal the need for a unified, intelligent framework that bridges automated threat modeling, AI-based prediction, and dynamic penetration testing. The next chapter will propose such a hybrid model to enhance proactive defense, optimize resource use, and ensure adaptability to evolving cyber threats.

2.6 Currently Available Solutions and Features Matrix

This section examines existing commercial and open-source solutions that integrate threat modeling with penetration testing capabilities. The analysis focuses on their core features, automation levels, integration capabilities, and limitations to identify gaps that this project aims to address.

2.6.1 Commercial Solutions

Microsoft Threat Modeling Tool

The Microsoft Threat Modeling Tool is a mature, widely-adopted solution that automates threat identification using Data Flow Diagrams (DFDs) and the STRIDE framework. It provides an intuitive graphical interface for system modeling and automatically generates threat reports with suggested mitigations. However, it operates primarily as a standalone design-time tool with limited integration capabilities for continuous monitoring or automated penetration testing. The tool excels in threat identification but lacks real-time validation mechanisms to verify whether identified threats are actually exploitable in production environments.

Cigent Threat Modeling Platform

Cigent offers an enterprise-grade threat modeling platform that combines automated threat discovery with risk quantification. The platform integrates with existing security tools and provides continuous threat model updates based on system changes. It features collaboration capabilities for security teams and compliance reporting aligned with industry frameworks. Despite its strengths in automation and integration, Cigent does not provide native penetration testing capabilities, requiring external tools to validate identified threats practically.

ThreatModeler

ThreatModeler is a comprehensive commercial platform that automates threat modeling across the Software Development Lifecycle (SDLC). It supports multiple modeling methodologies including STRIDE, PASTA, and OCTAVE, and integrates with CI/CD pipelines for continuous threat assessment. The platform offers advanced features such as attack path analysis, compliance mapping, and collaborative modeling environments. However, its penetration testing integration remains limited to basic vulnerability scanning rather than active exploitation and validation.

IriusRisk

IriusRisk provides an automated threat modeling platform with a focus on DevSecOps integration. It features an extensive library of threat patterns, automated security requirements generation, and integration with ticketing systems for remediation tracking. The platform supports API-based integrations with security testing tools but does not natively execute penetration tests or validate exploitability of identified threats in real-time.

2.6.2 Open-Source Solutions

OWASP Threat Dragon

OWASP Threat Dragon is a free, open-source threat modeling tool that supports STRIDE-based threat identification through visual DFD modeling. It offers both desktop and web-based versions with a user-friendly interface suitable for teams of varying technical expertise. The tool provides basic threat libraries and mitigation suggestions but lacks automated testing capabilities, requiring manual validation of identified threats through separate penetration testing processes.

PyTM (Python Threat Modeling)

PyTM is a code-based threat modeling framework that allows security engineers to define system architectures programmatically and automatically generate threat reports. It integrates well with CI/CD pipelines and supports customization through Python scripting. PyTM leverages CAPEC and MITRE ATT&CK databases for comprehensive threat coverage. However, it remains primarily a modeling tool without built-in penetration testing or continuous validation mechanisms.

Threagile

Threagile is an open-source, risk-centric threat modeling tool that uses YAML-based architecture definitions to generate threat models and risk assessments automatically. It provides quantitative risk scoring and supports automated report generation with detailed mitigation strategies. Threagile excels in automation and scalability but does not include active testing capabilities to verify the practical exploitability of identified risks.

OWASP ZAP (Zed Attack Proxy)

While primarily a penetration testing tool rather than a threat modeling platform, OWASP ZAP is extensively used for web application security testing. It performs automated vulnerability scanning, active attacks, and fuzzing to identify exploitable weak-

nesses. ZAP integrates well with CI/CD pipelines and provides extensive API support. However, it lacks threat modeling capabilities and operates reactively rather than proactively identifying potential threats during the design phase.

Metasploit Framework

Metasploit is a comprehensive penetration testing framework that provides a vast collection of exploits, payloads, and auxiliary modules. It enables security professionals to simulate real-world attacks and validate system vulnerabilities. Metasploit excels in exploitation and post-exploitation activities but does not incorporate threat modeling or risk prioritization mechanisms, requiring separate processes to identify and prioritize targets.

2.6.3 Gap Analysis

The comparative analysis reveals several critical gaps in existing solutions:

Integration Gap: Most threat modeling tools operate independently from penetration testing frameworks, creating a disconnect between threat identification and practical validation. Organizations must manually bridge this gap, leading to inefficiencies and potential oversights.

Automation Gap: While many tools offer automation in either threat modeling or penetration testing, few provide end-to-end automation that continuously validates threat models through active testing. This limitation prevents organizations from maintaining real-time security posture awareness.

Intelligence Gap: Current solutions lack sophisticated AI/ML integration for predictive threat analysis and autonomous decision-making. Most tools rely on predefined rules and patterns, failing to adapt to novel or evolving attack vectors.

Continuous Monitoring Gap: The majority of threat modeling solutions operate as point-in-time assessment tools rather than continuous monitoring systems. This approach fails to address the dynamic nature of modern cyber threats and rapidly evolving system architectures.

Unified Risk Context Gap: Existing tools often present threat modeling results and penetration testing findings separately, requiring manual correlation and prioritization. This fragmentation hinders effective risk management and resource allocation.

Feedback Loop Gap: There is limited capability for penetration testing results to automatically update and refine threat models, preventing organizations from learning from actual attack outcomes and improving their security posture iteratively.

These gaps underscore the need for an integrated framework that combines automated threat modeling, intelligent penetration testing, and continuous monitoring within a unified platform—the core objective of this research project.

2.7 Tools Background

This section provides detailed technical background on the primary tools and frameworks that will be leveraged or referenced in this research project. Understanding these tools' architectures, capabilities, and limitations is essential for designing an effective integrated solution.

2.7.1 Threat Modeling Frameworks and Standards

STRIDE Framework

STRIDE, developed by Microsoft, is a mnemonic-based threat classification system representing six categories of security threats: Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, and Elevation of privilege. Each category addresses a fundamental security property—authentication, integrity, non-repudiation, confidentiality, availability, and authorization respectively. STRIDE's systematic approach enables security teams to methodically identify potential threats by examining each component of a system against these six categories. The framework's strength lies in its comprehensive coverage and ease of use, making it particularly effective for structured threat modeling during the design phase. However, STRIDE does not inherently prioritize threats or assess their likelihood, requiring supplementary risk assessment methodologies.

DREAD Framework

DREAD is a risk assessment model that complements threat identification frameworks like STRIDE by quantifying threat severity. The acronym stands for Damage potential, Reproducibility, Exploitability, Affected users, and Discoverability. Each threat is scored on a scale (typically 1-10) across these five dimensions, and the average score represents the overall risk rating. DREAD provides an objective, quantitative mechanism for prioritizing threats based on their potential impact and likelihood of exploitation. Despite its utility, DREAD has been criticized for subjectivity in scoring and the potential for inconsistent assessments across different evaluators. Some organizations have discontinued its use in favor of more standardized frameworks, yet it remains valuable for risk-driven security testing.

PASTA (Process for Attack Simulation and Threat Analysis)

PASTA is a comprehensive, risk-centric threat modeling methodology consisting of seven stages: Definition of Objectives, Definition of Technical Scope, Application Decomposition, Threat Analysis, Vulnerability and Weakness Analysis, Attack Modeling, and Risk

and Impact Analysis. Unlike STRIDE's focus on threat categorization, PASTA emphasizes business risk alignment and attack simulation. The framework integrates business objectives with technical security analysis, ensuring that threat modeling efforts directly support organizational risk management goals. PASTA's thoroughness makes it suitable for complex, enterprise-level applications, though its comprehensive nature requires significant time and expertise to implement effectively.

MITRE ATT&CK Framework

The MITRE ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) framework is a globally-accessible knowledge base of adversary behaviors based on real-world observations. It organizes attack techniques into tactical categories such as Initial Access, Execution, Persistence, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement, Collection, Command and Control, Exfiltration, and Impact. Each technique is documented with detailed descriptions, detection methods, and mitigation strategies. ATT&CK provides a common language for describing cyber adversary behavior and has become the de facto standard for threat intelligence sharing and defensive planning. Its integration with threat modeling enhances the realism and accuracy of threat scenarios by grounding them in observed attacker tactics.

CAPEC (Common Attack Pattern Enumeration and Classification)

CAPEC is a comprehensive dictionary of known attack patterns used by adversaries to exploit vulnerabilities in cyber-enabled capabilities. Maintained by MITRE, CAPEC describes attack patterns from the attacker's perspective, detailing prerequisites, execution flows, and potential mitigations. Each pattern is categorized by mechanism (such as injection, resource manipulation, or deceptive interactions) and linked to relevant Common Weakness Enumeration (CWE) entries. CAPEC complements threat modeling by providing concrete examples of how theoretical threats might be executed in practice, facilitating more realistic and actionable threat models.

2.7.2 Penetration Testing Tools

Metasploit Framework

Metasploit is the world's most widely used penetration testing framework, providing a comprehensive platform for developing, testing, and executing exploit code against remote targets. It contains over 2,000 exploits, 500 payloads, and numerous auxiliary modules for reconnaissance, scanning, and post-exploitation. Metasploit's modular architecture allows security professionals to combine exploits with different payloads and encoding techniques to bypass security controls. The framework supports both manual

exploitation through its console interface (msfconsole) and automated scanning through integration with vulnerability assessment tools. Metasploit's powerful capabilities make it indispensable for validating vulnerabilities identified during threat modeling, though its complexity requires significant expertise to use effectively.

Nmap (Network Mapper)

Nmap is the industry-standard tool for network discovery and security auditing. It uses raw IP packets to determine available hosts on a network, their open ports, running services, operating systems, and firewall configurations. Nmap supports various scanning techniques including TCP SYN scanning, TCP connect scanning, UDP scanning, and more specialized methods for bypassing firewalls and intrusion detection systems. Its scripting engine (NSE) extends functionality with over 600 scripts for vulnerability detection, service enumeration, and network reconnaissance. Nmap's speed, flexibility, and accuracy make it essential for the reconnaissance phase of penetration testing and for continuous network monitoring.

OWASP ZAP (Zed Attack Proxy)

OWASP ZAP is an integrated penetration testing tool specifically designed for web application security testing. It acts as a man-in-the-middle proxy, intercepting and analyzing HTTP/HTTPS traffic between a browser and web application. ZAP provides automated scanners for common vulnerabilities (XSS, SQL injection, CSRF, etc.), an intercepting proxy for manual testing, spidering capabilities for application mapping, and fuzzing tools for input validation testing. Its extensible architecture supports custom scripts and plugins, enabling adaptation to specific testing requirements. ZAP's API facilitates integration with CI/CD pipelines for continuous security testing.

Burp Suite

Burp Suite is a comprehensive web application security testing platform combining manual and automated testing capabilities. Its core components include an intercepting proxy, application-aware spider, advanced web application scanner, intruder tool for customized attacks, repeater for manipulating and resending requests, sequencer for analyzing session token randomness, and decoder/encoder utilities. Burp Suite Professional's scanner uses sophisticated techniques to identify vulnerabilities with minimal false positives. The tool's extensibility through the BApp Store and custom extensions makes it adaptable to diverse testing scenarios.

Nikto

Nikto is an open-source web server scanner that performs comprehensive tests against web servers to identify potential vulnerabilities, misconfigurations, and security issues. It checks for over 6,700 potentially dangerous files and programs, outdated server versions, version-specific problems, and configuration issues such as multiple index files and HTTP server options. Nikto can also identify installed web servers and software through fingerprinting techniques.

SQLMap

SQLMap is an automated SQL injection detection and exploitation tool that supports a wide range of database management systems including MySQL, Oracle, PostgreSQL, Microsoft SQL Server, SQLite, and others. It automates the process of detecting and exploiting SQL injection flaws, providing capabilities for database fingerprinting, data enumeration, accessing underlying file systems, and executing commands on the operating system. SQLMap supports various injection techniques including boolean-based blind, time-based blind, error-based, UNION query-based, stacked queries, and out-of-band.

Hydra

Hydra is a fast and flexible login cracker that supports numerous protocols including AFP, Cisco AAA, Cisco auth, Cisco enable, CVS, FTP, HTTP(S), IMAP, IRC, LDAP, MS-SQL, MySQL, NCP, NNTP, Oracle, POP3, PostgreSQL, RDP, SMB, SMTP, SNMP, SSH, Telnet, and VNC. It performs parallelized dictionary and brute-force attacks against authentication mechanisms, making it effective for testing password strength and identifying weak credentials.

2.7.3 Vulnerability Assessment Tools

Nessus

Nessus, developed by Tenable, is one of the most widely deployed vulnerability scanners in the industry. It performs comprehensive vulnerability assessments across networks, systems, applications, and cloud infrastructure. Nessus maintains an extensive database of vulnerability checks (over 165,000 plugins), covering CVEs, configuration issues, missing patches, malware, and compliance violations. Its policy-based scanning allows customization for specific regulatory requirements (PCI DSS, HIPAA, CIS benchmarks, etc.). Nessus provides detailed vulnerability reports with risk ratings, remediation guidance, and exploit availability information.

OpenVAS

OpenVAS (Open Vulnerability Assessment System) is a comprehensive open-source vulnerability scanning and management framework. As a fork of the original Nessus before its commercialization, OpenVAS maintains community-driven development and offers capabilities comparable to commercial scanners. It includes a continuously updated feed of Network Vulnerability Tests (NVTs), support for various scan configurations and policies, authenticated and unauthenticated scanning modes, and detailed reporting with remediation recommendations.

Nexpose

Rapid7's Nexpose (now part of InsightVM) is an enterprise-grade vulnerability management solution that combines comprehensive scanning with risk-based prioritization and remediation workflows. It provides real-time vulnerability assessment, adaptive security with live monitoring, integration with Metasploit for exploit validation, policy compliance scanning, and asset discovery and classification.

2.7.4 Automation and Integration Frameworks

PTES (Penetration Testing Execution Standard)

PTES is a comprehensive framework defining the standard methodology for penetration testing engagements. It encompasses seven phases: Pre-engagement Interactions (scope definition, rules of engagement), Intelligence Gathering (information collection about the target), Threat Modeling (identification of potential threats), Vulnerability Analysis (discovery and validation of vulnerabilities), Exploitation (attempted compromise of systems), Post Exploitation (assessment of compromised systems' value and potential for further access), and Reporting (documentation of findings and recommendations). PTES provides a common language and structured approach for penetration testing, ensuring consistency, completeness, and professionalism across engagements.

OWASP Testing Guide

The OWASP Testing Guide is a comprehensive resource for web application security testing, maintained by the Open Web Application Security Project community. It provides detailed methodologies, test cases, and tools for assessing web application security across multiple categories including information gathering, configuration management, authentication, authorization, session management, input validation, error handling, cryptography, business logic, and client-side testing.

Jenkins and CI/CD Integration

Jenkins is an open-source automation server widely used for continuous integration and continuous deployment (CI/CD) pipelines. In the context of security testing, Jenkins enables automated execution of security scans, penetration tests, and compliance checks throughout the software development lifecycle. Security tools such as OWASP ZAP, Nessus, and static analysis tools can be integrated into Jenkins pipelines through plugins or API calls, automatically triggering security assessments when code is committed or deployed.

2.7.5 Artificial Intelligence and Machine Learning Tools

TensorFlow and PyTorch

TensorFlow (developed by Google) and PyTorch (developed by Facebook/Meta) are the leading open-source frameworks for developing and deploying machine learning models. In cybersecurity contexts, these frameworks enable the development of models for anomaly detection, behavioral analysis, threat prediction, malware classification, and automated decision-making. Their extensive libraries, community support, and production-ready deployment capabilities make them suitable for integrating AI-driven intelligence into security platforms.

Scikit-learn

Scikit-learn is a Python library providing simple and efficient tools for data mining and machine learning. It includes classification algorithms (SVM, Random Forests, Neural Networks), regression techniques, clustering methods, dimensionality reduction, and model selection utilities. In cybersecurity, scikit-learn is commonly used for feature engineering, model training and evaluation, and implementing traditional machine learning approaches for tasks such as intrusion detection, risk scoring, and threat classification.

Natural Language Processing (NLP) Tools

NLP tools such as spaCy, NLTK, and transformers (Hugging Face) enable automated analysis of textual security data including vulnerability descriptions, threat reports, security advisories, and penetration testing findings. In threat modeling contexts, NLP can extract relevant entities (CVE identifiers, attack techniques, affected systems), classify threat severity, generate automated documentation, and correlate information across multiple sources.

2.7.6 Threat Intelligence Platforms

MISP (Malware Information Sharing Platform)

MISP is an open-source threat intelligence platform designed for sharing, storing, and correlating Indicators of Compromise (IoCs) and threat information. It supports structured data formats (STIX, MISP objects), automated correlation of attributes, integration with security tools through APIs, and collaborative sharing among trusted communities. MISP enables organizations to consume external threat intelligence, enrich it with internal observations, and share findings with partners—creating a collective defense capability.

STIX/TAXII

Structured Threat Information Expression (STIX) and Trusted Automated Exchange of Indicator Information (TAXII) are OASIS standards for representing and exchanging cyber threat intelligence. STIX defines a language for describing cyber threats, including observables, indicators, incidents, tactics/techniques, threat actors, and campaigns. TAXII defines protocols for transmitting STIX content between organizations and systems. Together, they enable standardized, machine-readable threat intelligence sharing that can be automatically consumed and acted upon by security tools.

AlienVault OTX (Open Threat Exchange)

AlienVault OTX is a community-driven threat intelligence platform where security researchers and practitioners share IoCs, attack patterns, and threat analysis. It provides access to millions of threat indicators across various categories including malware samples, malicious domains/IPs, file hashes, and YARA rules. OTX offers RESTful APIs for automated threat intelligence consumption, enabling integration with security tools and SIEM platforms.

2.7.7 Containerization and Virtualization Technologies

Docker

Docker is a containerization platform that packages applications and their dependencies into lightweight, portable containers. In penetration testing and security research contexts, Docker enables creation of consistent, reproducible testing environments, rapid deployment of vulnerable applications for testing (e.g., DVWA, Metasploitable), isolation of potentially dangerous tools and exploits, and automation of test environment provisioning and teardown. Docker's efficiency and portability make it ideal for building scalable, automated penetration testing platforms.

Kubernetes

Kubernetes is an open-source container orchestration platform that automates deployment, scaling, and management of containerized applications. In security testing contexts, Kubernetes enables orchestration of large-scale penetration testing operations, dynamic scaling of testing resources based on demand, management of distributed testing agents and scanners, and integration with CI/CD pipelines for continuous security assessment.

VirtualBox and VMware

VirtualBox (open-source) and VMware (commercial) are virtualization platforms that enable running multiple operating systems simultaneously on a single physical machine. In penetration testing, virtualization provides isolated testing environments, the ability to snapshot and restore system states for repeatable testing, and simulation of complex network topologies. These platforms support building realistic testing scenarios without risking production systems, making them fundamental to safe and effective security research and validation.

2.8 Summary

The tools and frameworks reviewed represent current state-of-the-art in threat modeling, penetration testing, and security automation. Each tool addresses specific aspects yet typically operates independently. Understanding their capabilities and limitations provides foundation for designing integrated framework leveraging their strengths while addressing identified gaps.

Chapter 3

Requirements Analysis

3.1 Chapter Overview

This chapter presents a comprehensive requirements analysis for the proposed system, integrating and refining the business, user, and system requirements in alignment with the project proposal and the research gaps identified in Chapter 2. The chapter follows an IEEE-style structure and adopts an Agile–Scrum development methodology to support continuous adaptation to evolving cybersecurity threats.

3.2 Business Requirements Identification

The business requirements for this graduation project were identified through an iterative and collaborative process involving academic supervisors, cybersecurity domain knowledge, and continuous team discussions. These requirements are directly derived from the problem statement, project objectives, expected solutions, and the research gaps highlighted in Chapter 2. Current cybersecurity practices often suffer from fragmentation, where threat modeling remains largely theoretical, penetration testing is reactive and unguided by predictive risk analysis, and continuous monitoring lacks adaptability to emerging threats. This project addresses these limitations by proposing a unified, intelligence-driven framework that integrates threat modeling, penetration testing, threat intelligence, and continuous monitoring.

The key business requirements are defined as follows. First, the system shall support seamless integration of security processes. Threat modeling, penetration testing, threat intelligence ingestion, and continuous monitoring must operate within a single coordinated workflow. Threats identified during modeling activities shall directly influence penetration testing scope and priorities, while testing results shall feed back into the threat model. This requirement directly addresses the integration gap identified in the reviewed literature.

Second, the system shall enforce risk-driven prioritization of security activities. Testing efforts shall be guided by quantified risk metrics that reflect asset criticality, threat likelihood, and potential business impact. By prioritizing high-risk attack paths instead of exhaustive but unfocused testing, the framework improves assessment efficiency while maintaining meaningful security coverage.

Third, the system shall enable collaboration and traceability across security roles. Developers, architects, penetration testers, and analysts must share a common security context supported by centralized documentation and traceable artifacts. Each vulnerability, test result, and mitigation shall be linked to its originating threat scenario, ensuring transparency and accountability throughout the security lifecycle.

Fourth, the system shall support continuous adaptation and improvement. Threat models shall not remain static documents; instead, they shall evolve dynamically based on penetration testing outcomes and updates from threat intelligence sources. This requirement directly responds to the lack of effective feedback loops highlighted in Chapter 2.

Fifth, the system shall emphasize automation and scalability. Automated threat-to-test mapping, dynamic scope definition, and report generation are required to reduce manual effort and enable the framework to scale to larger or more complex environments without excessive overhead.

Finally, the system shall provide comprehensive reporting and compliance support. Generated reports shall clearly link threats, vulnerabilities, risks, and mitigations, enabling informed decision-making and supporting alignment with recognized security standards and regulatory expectations.

These business requirements ensure that the proposed framework bridges the gap between predictive security analysis and practical validation while remaining feasible within the constraints of an academic graduation project.

3.2.1 Software Development Methodology

The Agile software development methodology, implemented through the Scrum framework, was adopted for this project due to the dynamic and exploratory nature of cybersecurity research. Agile emphasizes flexibility, iterative development, and continuous feedback, making it particularly suitable for cybersecurity projects where requirements evolve as threats and insights emerge. Rather than defining all system requirements upfront, the Agile approach allows the framework to be refined incrementally as threat models mature, testing workflows are validated, and integration challenges are identified.

3.2.2 Justification for Scrum Selection

Scrum was selected because it aligns closely with the exploratory and evolving nature of cybersecurity system development. Threat intelligence updates, penetration testing findings, and architectural refinements can introduce new requirements at any stage of the project. Scrum's short, time-boxed Sprints enable the team to adapt quickly to these changes without destabilizing the overall project plan. Additionally, Scrum supports incremental delivery of functional components, allowing early validation of core ideas such as threat-to-test mapping and feedback loops. Regular Sprint Reviews and Retrospectives facilitate continuous evaluation of design decisions and risk assumptions. From an academic perspective, Scrum's lightweight structure also supports effective progress tracking and documentation within limited time and resource constraints.

3.2.3 Scrum-Based Development Plan

The project was implemented over multiple Sprints spanning approximately 15 weeks, as summarized below:

Sprint	Duration	Deliverables
Sprint 1	Weeks 1–3	Requirements elicitation, high-level system architecture definition, initial STRIDE-based threat modeling, and backlog prioritization.
Sprint 2	Weeks 4–6	Expansion of threat models and automated mapping of threats to known attack patterns and adversary techniques.
Sprint 3	Weeks 7–9	Integration of penetration testing workflows and automation of selected testing activities.
Sprint 4	Weeks 10–12	Implementation of continuous feedback mechanisms to update threat models and risk scores based on testing results.
Sprint 5	Weeks 13–15	Development of risk visualization components, reporting mechanisms, and overall system refinement.

Table 3.2: Scrum-Based Development Plan

This iterative plan ensured alignment with project objectives and supported continuous validation against the identified research gaps.

3.3 User Functional Requirements

The proposed framework supports multiple user roles, each interacting with the system from a different security perspective. Defining these roles ensures that functional requirements are aligned with real-world security workflows and responsibilities.

3.3.1 User Roles

Role	Description / Responsibilities
Security Analyst	Develops and maintains threat models, evaluates risk scores, and monitors threat intelligence updates.
Penetration Tester	Executes targeted penetration tests, validates modeled threats, and feeds findings back into the system.
Developer	Reviews identified vulnerabilities and implements mitigation measures during system development or maintenance.
System Architect	Assesses architectural risks, validates threat scenarios, and ensures alignment between system design and security requirements.
Compliance Stakeholder	Reviews reports to ensure alignment with organizational policies and regulatory requirements.

Table 3.4: User Roles

3.3.2 User Functional Requirements

User functional requirements (UFRs) define the interactions between end users and the system. Each requirement is aligned with one or more of the roles defined above. These requirements were specified following IEEE Std 830 principles and refined through Scrum backlog iterations.

UFR ID	Requirement	Primary User Role(s)
UFR-1	The system shall allow users to create, edit, and version-control threat models using STRIDE and DFD techniques.	Security Analyst, System Architect
UFR-2	The system shall automatically map identified threats to corresponding penetration testing techniques and tools.	Security Analyst, Penetration Tester
UFR-3	The system shall dynamically adjust testing scope based on system changes and emerging threats.	Penetration Tester, Security Analyst
UFR-4	The system shall provide a centralized repository for storing threat models, testing results, and mitigation actions.	All Roles
UFR-5	The system shall present interactive risk visualizations (e.g., heatmaps, matrices) to support prioritization.	Security Analyst, System Architect, Compliance Stakeholder
UFR-6	The system shall automatically update threat models based on penetration testing outcomes.	Security Analyst, Penetration Tester
UFR-7	The system shall support collaborative features such as shared access, notifications, and role-based permissions.	All Roles
UFR-8	The system shall generate comprehensive security assessment reports for technical and non-technical stakeholders.	Compliance Stakeholder, Security Analyst
UFR-9	The system shall ingest threat intelligence from standardized feeds (e.g., STIX/TAXII) to enrich analysis.	Security Analyst
UFR-10	The system shall support safe simulation environments for controlled penetration testing.	Penetration Tester, Security Analyst

Table 3.6: User Functional Requirements

This structured approach ensures clarity, traceability, and alignment of responsibilities with system features, improving usability, collaboration, and adaptability for all stakeholders.

3.4 System Functional and Non-Functional Requirements

This section defines the functional and non-functional requirements of the system, ensuring alignment with business and user needs. Functional requirements describe the system behavior and services, while non-functional requirements specify constraints and quality attributes.

3.4.1 System Functional Requirements

System functional requirements define the system's core security and monitoring capabilities.

SFR ID	Requirement	Primary User Role
SFR-1	The system shall provide a threat modeling engine supporting STRIDE-based classification and asset mapping.	Security Analyst, System Architect
SFR-2	The system shall automate penetration testing execution using integrated tools (e.g., OWASP ZAP, Nmap, Metasploit).	Penetration Tester
SFR-3	The system shall integrate external threat intelligence and correlate it with internal assets.	Security Analyst, System Architect
SFR-4	The system shall automate threat-to-test mapping and dynamic scoping.	Security Analyst, Penetration Tester
SFR-5	The system shall implement a continuous feedback loop to refine threat models and risk scores.	Security Analyst, Penetration Tester
SFR-6	The system shall generate visual dashboards and structured security reports for decision-making.	Security Analyst, System Architect, Compliance Stakeholder
SFR-7	The system shall maintain secure, traceable storage of all security artifacts.	All Roles
SFR-8	The system shall support virtualized or containerized testing environments.	Penetration Tester, Security Analyst

Table 3.8: System Functional Requirements

3.4.2 Non-Functional Requirements

Non-functional requirements specify quality attributes and operational constraints of the system:

Attribute	Requirement
Performance	The system shall process threat analysis and testing results within acceptable response times suitable for continuous monitoring.
Scalability	The system shall support growth in assets, threats, and testing tools without requiring major architectural changes.
Reliability	The system shall ensure consistent operation, accurate data correlation, and availability of security artifacts.
Security	The system shall protect data confidentiality, integrity, and availability using encryption, authentication, and access controls.
Usability	The system shall provide intuitive interfaces, clear visualizations, and user-friendly navigation.
Maintainability	The system shall support modular updates, integration of new intelligence sources, and efficient debugging or enhancements.
Portability	The system shall be deployable across multiple platforms and environments, including local, virtualized, and cloud setups.

Table 3.10: Non-Functional Requirements

These non-functional requirements ensure the system's robustness, adaptability, and user-centered design, complementing the functional capabilities described above.

3.5 System Evaluation and Validation Criteria

Evaluation criteria were derived directly from the defined requirements and serve as a basis for validating the proposed framework. Functional validation focuses on the accuracy of threat-to-test mapping, completeness of threat coverage, and traceability across artifacts. Performance criteria assess efficiency, responsiveness, and scalability. Risk effectiveness criteria evaluate prioritization accuracy and adaptive risk management, while usability criteria assess ease of use, collaboration support, and report clarity.

3.6 Chapter Summary

This chapter presented an integrated requirements analysis aligned with the project proposal and research gaps. It justified the adoption of the Agile–Scrum methodology and defined the business, user, and system requirements that guide the system design and implementation discussed in the subsequent chapters.

Chapter 4

System Design

4.1 Overview

This chapter presents the comprehensive system design for the TIBSA (Threat Intelligence and Behavioral Security Analysis) platform. The design encompasses a multi-layered architecture that integrates modern security tools, cloud-native services, and machine learning capabilities to deliver a robust threat intelligence and security analysis solution. The chapter begins with a detailed exploration of the platform's eight-layer architecture, covering everything from the user interface layer through to the core TIBSA suite. Each layer is examined in terms of its components, data flows, and integration points with other system elements. The chapter also includes complete data flow maps that trace the path of information through the system for key operations such as authentication, file analysis, and threat modeling. Finally, user interface scenarios are presented through wireframes that demonstrate the platform's key screens and user interactions.

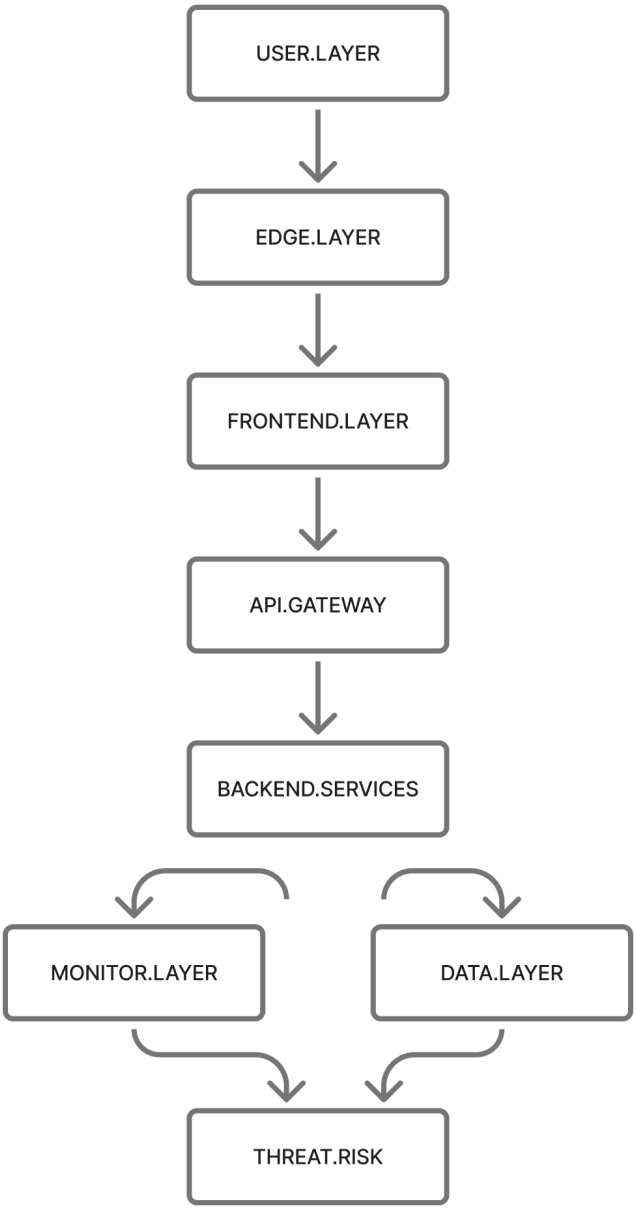

Figure 4.1: TIBSA Platform - Complete Architecture Overview

4.2 Layered Architecture Design

This section presents the detailed architecture of the TIBSA platform, organized into eight distinct layers. Each layer represents a logical grouping of components with specific responsibilities, from the user-facing interfaces at the top to the core TIBSA suite at the bottom. The layered approach ensures separation of concerns, modularity, and maintainability, while enabling seamless data flow and integration between components.

4.2.1 LAYER 1 - User Layer

Figure 4.2: Layer 1- User Layer Architecture

1.1 Security Analyst Console

The Security Analyst Console is the primary interface used by SOC analysts, threat hunters, and penetration testers. It provides access to dashboards, incident views, analytics tools, and pentest management interfaces. This layer uses no external tools, as it is a pure UI layer that serves as the entry point for security professionals.

The console provides several key services including a threat hunting interface, alert investigation dashboard, pentest workflow triggering capabilities, and a results review console. Analysts interact with the system through various actions such as queries, pentest runs, and incident views, all of which are authenticated using JWT tokens and tracked through UI interactions.

The data flow begins when the console receives analyst actions and JWT tokens as input, which are then processed into authenticated API requests, analysis queries, and pentest triggers as output. These requests are first sent to Frontend Security Controls in Layer 3 for input sanitization, then forwarded to the API Gateway in Layer 4 for authentication and routing. The console receives data back from Backend Services in Layer 5 via the API Gateway for display to the analyst.

1.2 Client Logic Module

The Client Logic Module serves as the browser-side brain of the platform, managing the complete lifecycle of user sessions and authentication. It operates entirely as client-side JavaScript logic without relying on external tools, handling critical functions that ensure smooth and secure user interactions.

This module provides comprehensive services including JWT token management, token refresh automation, session timeout handling, UI state management based on RBAC (Role-Based Access Control), API error handling, and file chunking preparation. It continuously monitors token expiration and automatically refreshes them to maintain uninterrupted user sessions.

The module accepts user interactions, JWT tokens, UI events, form data, and files as input, transforming these into structured requests, refreshed tokens, chunked uploads, and properly handled errors as output. It maintains session state and authentication tokens while communicating with the Auth Interceptor in Layer 3 to attach tokens to outgoing requests. All backend communication flows through the API Gateway in Layer 4, and the module receives new tokens from the Authentication Service in Layer 5 during refresh cycles.

1.3 Customer Browser Interfaces

The Customer Browser Interfaces provide all customer-facing web pages, offering a complete set of UI components for user interaction without requiring external tools. These interfaces encompass every aspect of the customer experience from initial registration through daily platform usage.

The services provided include login and registration pages, MFA (Multi-Factor Authentication) enrollment and verification UI, comprehensive dashboard views, an intuitive file upload interface, and scanning tool initiation forms. Each interface is designed with

user experience and security in mind, ensuring that all customer actions are properly validated before reaching backend systems.

Customer actions such as login attempts, registration submissions, file uploads, and scan initiations serve as input to these interfaces, along with MFA inputs when required. The interfaces generate clean UI-validated requests, handle session initialization, and manage navigation events as output. All requests are first sent to Client-Side Security Controls in Layer 3 for validation, then forwarded to the API Gateway in Layer 4 via authenticated requests. Dynamic content is received from Backend Services in Layer 5 to populate dashboards and display results.

1.4 Client-Side Security Controls

The Client-Side Security Controls act as the first major security checkpoint in the system, implementing multiple layers of protection using browser-native security features and custom validators. These controls serve as the gatekeeper between user input and the backend infrastructure, ensuring that only clean, validated data proceeds downstream.

The security services provided include XSS (Cross-Site Scripting) prevention through comprehensive input sanitization, Content Security Policy (CSP) enforcement, filename validation to prevent directory traversal attacks, soft rate limiting to prevent abuse, and MFA flow protection to secure authentication processes. Each control operates transparently to users while maintaining strict security standards.

The controls receive outgoing requests, form fields, filenames, and script execution attempts from all User Layer components (Security Analyst Console, Client Logic Module, and Customer Browser Interfaces) as input. They produce sanitized inputs, enforce CSP rules, validate safe file uploads, and apply rate-limited requests as output. After validation, all data is sent to the Frontend Layer (Layer 3), while malicious payloads are blocked before they can reach the backend systems.

4.2.2 LAYER 2 - Edge Layer

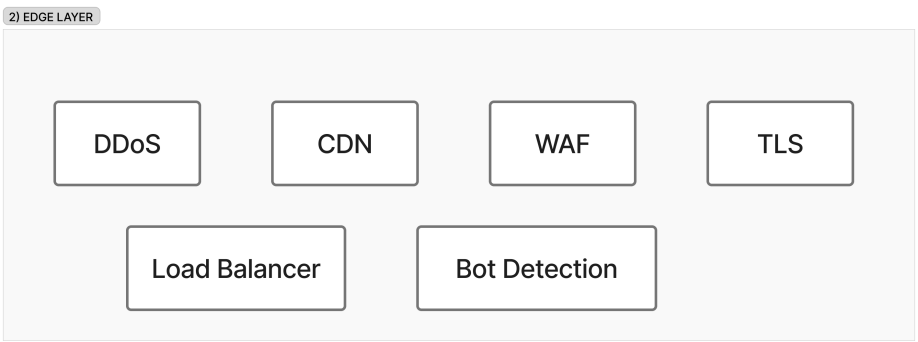

Figure 4.3: Layer 2- Edge Layer Architecture

2.1 CDN (Content Delivery Network)

The CDN serves as the global distributed entry point for all traffic entering the platform, utilizing Cloudflare's Free Plan as Service #1. It functions as the first point of contact between users worldwide and the TIBSA platform, providing critical performance and availability improvements through intelligent caching and distribution.

The CDN provides several essential services including global content caching, static asset delivery for JavaScript, CSS, and images, nearest-node routing for low latency, basic request validation, and traffic absorption with load distribution. By caching content at edge locations around the world, the CDN dramatically reduces latency for users regardless of their geographic location.

Raw HTTP(S) requests from user browsers worldwide arrive at the CDN as input. When the CDN has cached content (a cache hit), it returns static responses directly to users, bypassing downstream systems entirely. For cache misses, the CDN forwards clean requests to the WAF in Layer 2.2 for application-layer inspection, then caches the response for future requests.

2.2 WAF (Web Application Firewall)

The WAF performs deep inspection of incoming requests by analyzing patterns associated with common web attacks, operating as part of Cloudflare's Free Plan (Service #1). It examines every request that passes through the CDN, looking for malicious patterns and known attack signatures before allowing traffic to proceed further into the infrastructure.

The WAF provides comprehensive protection including SQL Injection detection and blocking, Cross-Site Scripting (XSS) prevention, CSRF token validation, path traversal detection, malicious payload filtering, and CAPTCHA challenges for suspicious requests. Each of these protections operates in real-time, examining request headers, bodies, and parameters for signs of malicious intent.

After receiving CDN-filtered requests as input, the WAF produces three possible outputs: allowed requests are sent to DDoS Protection in Layer 2.3, blocked requests receive an error page, and suspicious requests are challenged with CAPTCHA or JavaScript verification. All blocked attempts are logged and sent to the SIEM analysis system in Layer 7 for security monitoring and threat intelligence purposes.

2.3 DDoS Protection

The DDoS Protection system identifies and mitigates abnormal traffic spikes and distributed denial-of-service attacks using Cloudflare's Free Plan (Service #1). It continuously monitors traffic patterns in real-time, detecting anomalies that could indicate an ongoing attack and taking immediate action to protect the platform's availability.

The system provides multi-layered protection including Layer 3, 4, and 7 DDoS attack mitigation, continuous traffic rate monitoring, anomaly detection using behavioral analysis, packet dropping for malicious traffic, and traffic normalization to ensure clean requests reach application servers. It can distinguish between legitimate traffic spikes (such as during peak usage) and malicious attack patterns.

WAF-validated requests arrive as input, and the system produces two types of output: normalized safe traffic is forwarded to TLS Termination in Layer 2.4, while abnormal or malicious packets are dropped immediately. The system protects all downstream services from volumetric attacks and continuously monitors traffic patterns, sending detailed metrics to the Monitoring Layer in Layer 7 for analysis and alerting.

2.4 TLS Termination

TLS Termination is the point where encrypted HTTPS traffic is securely decrypted, utilizing Cloudflare's TLS/SSL capabilities (Service #1). This critical security function validates certificates, ensures proper use of cryptographic standards, and offloads the computationally expensive encryption and decryption tasks from backend servers, allowing them to focus on application logic.

The system provides essential services including HTTPS traffic decryption, certificate validation, TLS 1.3 support for modern security standards, SSL/TLS offloading to improve backend performance, and secure session handling. It ensures that all traffic uses strong encryption in transit while making content accessible to downstream security inspection tools.

Encrypted HTTPS requests from DDoS Protection arrive as input, and the system validates SSL certificates and encryption standards before producing decrypted HTTP requests as output. These decrypted requests are then sent to the Load Balancer in Layer 2.5, enabling downstream systems to inspect and process the actual request content while maintaining end-to-end security.

2.5 Load Balancer

The Load Balancer distributes incoming requests across backend servers based on intelligent routing policies, using Cloudflare's Load Balancing feature (Service #1). It ensures optimal resource utilization and high availability by directing traffic to healthy servers and avoiding overloaded or failed instances.

The balancer provides comprehensive distribution services including request distribution across multiple servers, health check monitoring to detect server issues, round-robin routing for equal distribution, least connections algorithm to favor less-busy servers, weighted load distribution for capacity-based routing, and automatic failover handling when servers become unavailable. Each routing decision is made in real-time based on

current system conditions.

Decrypted HTTP requests from TLS Termination serve as input, and the balancer monitors server health continuously to adjust routing accordingly. Output consists of properly routed requests sent to correct backend service instances based on health checks and load conditions. Before reaching backend services, requests pass through the Bot Detection Engine in Layer 2.6 for final edge validation.

2.6 Bot Detection Engine

The Bot Detection Engine analyzes traffic behavior, device fingerprints, user interaction patterns, request frequency, and known malicious IP ranges to distinguish human users from automated bots, utilizing Cloudflare's Bot Management feature (Service #1). This sophisticated analysis protects the platform from automated attacks, scraping, and abuse while allowing legitimate automation when appropriate.

The engine provides multiple security services including bot versus human traffic distinction, device fingerprinting to identify returning visitors and potential threats, behavioral analysis to detect automation patterns, request pattern detection to identify suspicious activity, malicious IP blocking, automated scraping prevention, and credential stuffing protection. These capabilities work together to create a comprehensive defense against bot-based threats.

Routed requests from the Load Balancer arrive as input along with metadata including IP addresses, user agents, and behavioral patterns. The engine produces two types of output: human-verified requests are forwarded to the Frontend Layer in Layer 3, while blocked or challenged bots receive either a challenge page requiring human verification or are blocked outright. As the final security check at the edge before traffic reaches the application layer, this engine provides critical protection against automated attacks and scraping attempts.

4.2.3 LAYER 3 - Frontend Layer

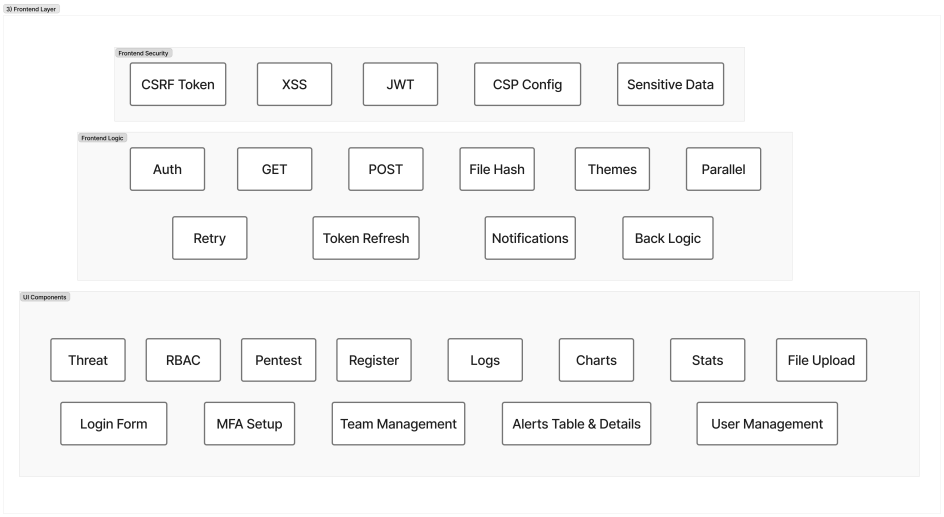

Figure 4.4: Layer 3- Frontend Layer Architecture

3.1 Frontend Framework & UI Components

The Frontend Framework provides all user interfaces, dashboards, and interactive components for the entire platform using Next.js 15, Tailwind CSS, and shadcn/ui as Service #2. This modern technology stack enables server-side rendering, static site generation, and dynamic client interactions while maintaining excellent performance and developer experience.

The framework provides comprehensive services including server-side rendering (SSR) for improved initial load times, static site generation (SSG) for cacheable pages, API routes for backend communication, complete dashboard interfaces for all user types, an intuitive file upload UI, threat modeling interfaces, authentication pages covering login, registration, and MFA, alerts and threat visualization tools, user management interfaces, a guided pentest wizard, and a comprehensive reports viewer.

User interactions, API responses, and routing requests serve as input to the framework, which produces rendered HTML and React components along with API calls to backend services as output. The framework consists of several specialized components: Login Form, Register Component, and MFA Setup UI handle user authentication by accepting email, password, and MFA codes as input and producing JWT access tokens and MFA confirmation as output, sending requests through the Auth Interceptor to the API Gateway and ultimately to Authentik in Layer 4.

Stats Widgets provide KPI visualization, scan statistics, and vulnerability metrics by receiving analytics API responses from backend services and displaying them as charts, counters, and time-series graphs. The Threat Map component offers geographic and topological threat visualization by connecting alert metadata and geo-location data to

create an interactive threat map, integrating with the Alerts Table and MISP threat intelligence from Layer 5.

The Alerts Table and Alert Details Modal provide structured alert listing and detailed incident views, receiving data from the SIEM in Layer 7 and Notification Service in Layer 5 to display paginated alerts and drill-down details with evidence. The File Upload Component manages file submission with validation, using the File Hash Calculator and Chunk Uploader to initiate chunked uploads with progress tracking, sending data to the File Upload Handler in Layer 3.2.

The Pentest Wizard guides users through penetration test configuration, accepting scope, targets, scan depth, and schedule as input and producing validated job configurations that are sent to the Pentest Orchestrator in Layer 5. The Logs Console enables real-time log viewing and searching by receiving streamed logs from Loki in Layer 7 and rendering them with search capabilities and downloadable exports.

User Management UI, Team Management UI, and RBAC Matrix components handle user creation, role assignment, and permission management, accepting user data and role definitions as input and sending updates to the User Service in Layer 5 via the API Gateway. The framework receives all traffic from the Edge Layer after bot detection, communicates with the API Gateway in Layer 4 for all backend operations, renders dynamic content based on user roles and permissions, and sends large file operations to the File Upload Handler.

3.2 Frontend Security Controls

Frontend Security Controls implement browser-side security mechanisms that protect UI and client-side data using built-in browser security features combined with custom validators. These controls operate transparently to provide multiple layers of protection without impacting user experience.

The XSS Sanitizer provides input sanitization and script tag removal, accepting raw user text and HTML input from forms and producing sanitized, safe HTML strings that are sent to UI Components and the GET/POST Helpers. The JWT Validator performs client-side token validation by accepting JWTs from storage and producing validation status along with extracted claims including user ID, role, and expiry information, which are then used by RBAC Logic and the Auth Interceptor.

The CSRF Token Handler protects against cross-site request forgery by accepting CSRF tokens from the backend and attaching them as headers to all outgoing POST, PUT, and DELETE requests before sending them through the Auth Interceptor to the API Gateway. The CSP Config Loader enforces Content Security Policy by accepting CSP configuration from the server and applying CSP rules that are enforced by the browser, protecting against XSS and malicious script injection.

Sensitive Data Masking provides PII and sensitive data redaction by accepting raw scan results, alert details, and sensitive fields and producing masked values (such as "******1234") that are safely displayed in UI Components. These security controls protect all outgoing requests from the Frontend Layer, work closely with the Frontend Logic Layer to ensure secure communication, and send validated data to the API Gateway in Layer 4.

3.3 Frontend Logic Layer

The Frontend Logic Layer handles all client-side business logic, request preparation, and state management using custom JavaScript and TypeScript logic. This layer orchestrates the complex interactions between user interface components, security controls, and backend services.

The GET/POST Helpers provide standardized HTTP request handling, accepting API endpoints, URL parameters, body payloads, and headers as input and producing parsed JSON responses or standardized errors as output, which are then sent to UI Components and the Notifications Queue. The Auth Interceptor performs automatic authentication header attachment by accepting requests from GET/POST helpers along with access tokens and CSRF tokens, producing fully authenticated requests with Authorization and CSRF headers that are sent to the API Gateway in Layer 4, while also triggering token refresh on 401 or 403 errors.

Token Refresh Logic handles automatic token renewal by accepting expiring or expired tokens along with refresh tokens as input and producing new access tokens that are stored in browser storage as output. This component communicates with Authentik in Layer 4 for token refresh and updates the Auth Interceptor with new tokens to maintain uninterrupted sessions.

Retry Logic provides network failure recovery with exponential backoff, accepting failed requests and error details such as timeouts or 5xx errors as input and producing either successful responses or final errors that are sent to the Notifications Queue. This is particularly critical for large file uploads and long-running pentest calls where transient network issues should not cause complete failure.

RBAC Logic implements client-side role-based access control by accepting JWT claims and the RBAC matrix as input and producing visibility and permission flags for UI elements as output, controlling which UI components are shown or hidden per user role. It's important to note that while this provides immediate UI feedback, server-side validation is still enforced in backend services for security.

The File Hash Calculator computes SHA-256 hashes for integrity by accepting binary or text files as input and producing unique file hashes as output, which are sent to the backend as metadata and to the Chunk Uploader for integrity checks. This prevents

duplicate scans and ensures forensic consistency across the platform.

Parallel Upload and Chunk Uploader manage large file chunking and parallel upload by accepting file objects, chunk size, upload URL, and session ID as input and producing chunk upload requests, progress updates, and final file IDs as output. This component works with Retry Logic for failed chunks and sends data to the Tusd Server in Layer 3.4, which stores files in Cloudflare R2 in Layer 6.

The Notifications Queue manages real-time notification handling by accepting backend events through polling or WebSockets along with frontend actions as input and producing UI notifications, alerts to the Alerts Table, and log events to the Logs Console as output. It displays scan completions, vulnerabilities, report readiness, and system warnings to keep users informed of important events.

Theme Manager controls visual theme preferences (light and dark mode) by accepting user preferences, system settings, and CSS definitions as input and producing applied CSS variables and dynamic styling as output, improving readability for dashboards and threat visualizations.

The Frontend Logic Layer receives data from Frontend Security Controls in Layer 3.2 after validation, sends requests to the API Gateway in Layer 4 via the Auth Interceptor, and manages all state and logic before backend communication occurs.

3.4 File Upload Handler

The File Upload Handler manages resumable, chunked uploads of large files up to 10 GB with integrity checking and resume support, utilizing Uppy and Tusd (open source tus.io server) as Service #8. This sophisticated upload system ensures that even very large files can be uploaded reliably over unreliable networks.

The handler provides essential services including resumable file uploads that can be paused and continued, chunked upload that splits large files into manageable pieces, upload progress tracking for user feedback, automatic retry on failure without losing progress, session persistence to resume interrupted uploads across browser sessions, support for files up to 10 GB in size, and integrity verification to ensure files are not corrupted during transmission.

File objects from the File Upload Component in Layer 3.1.5 serve as input along with chunk size, upload URL, and session ID. The handler produces several outputs including chunked file segments that are uploaded to the backend, progress updates to the UI to keep users informed, a final file ID upon completion, and upload confirmation.

The handler receives file hashes from the File Upload Component, uses the Chunk Uploader in Layer 3.3.7 for parallel upload management, and sends data to the Tusd Server backend component which stores files in Cloudflare R2 in Layer 6. Once upload is complete, the file is passed to the Scan Service in Layer 5 for processing, and the

Notifications Queue in Layer 3.3.8 is notified of completion or failure.

Integration points include working with the File Hash Calculator in Layer 3.3.6 for deduplication to avoid re-scanning identical files, coordinating with Retry Logic in Layer 3.3.4 for failed chunk recovery, storing final files in Cloudflare R2 in Layer 6 via the Tusd server, and triggering the backend processing pipeline that flows through Scan Service, Sandbox, and ML Engine.

4.2.4 LAYER 4 - API Engine

Figure 4.5: Layer 4 - API Engine Architecture

4.1 API Gateway

The API Gateway serves as the central entry point for all API requests using KrakenD Community Edition as Service #4, providing request validation, authentication, rate limiting, and intelligent routing to backend microservices. This critical component ensures that only valid, authenticated, and properly formatted requests reach backend services.

The gateway provides comprehensive services including serving as the central API entry point for all client requests, request routing to appropriate microservices, rate limiting enforcement to prevent abuse, request and response transformation to adapt between frontend and backend formats, API composition for aggregating multiple backend calls, circuit breaker pattern implementation to prevent cascade failures, and thorough request validation.

The gateway consists of several specialized components: The Auth Filter performs JWT token validation and permission checking by accepting HTTP requests with Authorization headers as input and producing authenticated request context or 401/403 errors as output. It validates tokens with Authentik in Layer 4.2 and sends valid requests to the Rate Limiter component.

The Rate Limiter controls request frequency and prevents DoS attacks by accepting request metadata including IP addresses, user IDs, and endpoint information as input. It uses Upstash Redis in Layer 6 for rate limit storage and produces either allowed requests that proceed to the Schema Validator or 429 Too Many Requests responses when limits are exceeded.

The Schema Validator ensures request body validity against JSON schemas by accepting JSON and FormData payloads as input and producing validated, sanitized data

or 400 Bad Request errors as output. This ensures that backend services receive only correct, safe input and sends validated requests to the Routing Engine.

The Routing Engine performs intelligent request routing to appropriate microservices by accepting validated, authenticated requests with URL and HTTP method information as input and producing routed requests to target backend services as output. It can route to Authentication Service in Layer 4.2, User Service, Scan Service, Threat Intelligence Service, Billing Service, Notification Service, ML Engine, Sandbox, and Threat Modeling Engine, all in Layer 5.

The gateway receives requests from the Frontend Layer in Layer 3.3 via the Auth Interceptor, first checks the Auth Filter for token validation, then checks the Rate Limiter using Redis cache, validates request schema and payload, routes to appropriate backend services in Layer 5, and returns aggregated responses to the frontend.

4.2 Authentication & Authorization Service

The Authentication & Authorization Service manages user authentication flows including login, registration, token refresh, and multi-factor verification using Authentik, an open-source identity platform, as Service #3. This service is the cornerstone of platform security, ensuring that only authorized users can access protected resources.

The service provides extensive capabilities including user login and registration, Multi-Factor Authentication with TOTP and WebAuthn, comprehensive Role-Based Access Control (RBAC), user directory management, Single Sign-On (SSO) support, OAuth2 and OIDC provider functionality, LDAP and Active Directory integration, session management, token issuance and validation, and password reset flows.

The login function accepts email and password as input and produces JWT access tokens and refresh tokens as output. The process validates credentials, checks MFA status, and issues tokens that are sent to the frontend via the API Gateway. The register function accepts email, password, and user details as input and produces user account creation confirmation as output, validating input, creating users in the directory, and triggering verification emails via the Notification Service in Layer 5.

The refreshToken function accepts valid refresh tokens as input and produces new access tokens as output, validating refresh tokens and issuing new access tokens. This function is called by Token Refresh Logic in Layer 3.3.3 when access tokens expire. The verifyMFA function accepts user sessions and MFA codes (TOTP or WebAuthn) as input and produces MFA verification results and session upgrades as output, validating the second factor and upgrading session privileges for administrative actions and sensitive operations.

The service integrates with the API Gateway Auth Filter in Layer 4.1.1 for token validation, stores user credentials and sessions in Neon Postgres in Layer 6, uses Redis

in Layer 6 for session caching, sends notifications via the Notification Service in Layer 5, enforces RBAC policies for all backend services, and provides tokens to all authenticated frontend requests.

RBAC integration includes defining roles such as Admin, Analyst, User, and Guest, managing permissions for scanning, viewing reports, managing users, and configuring the system, enforcing permission checks in backend services, and integrating with the User Management UI in Layer 3.1.8 for role assignment.

4.2.5 LAYER 5 - Backend Services

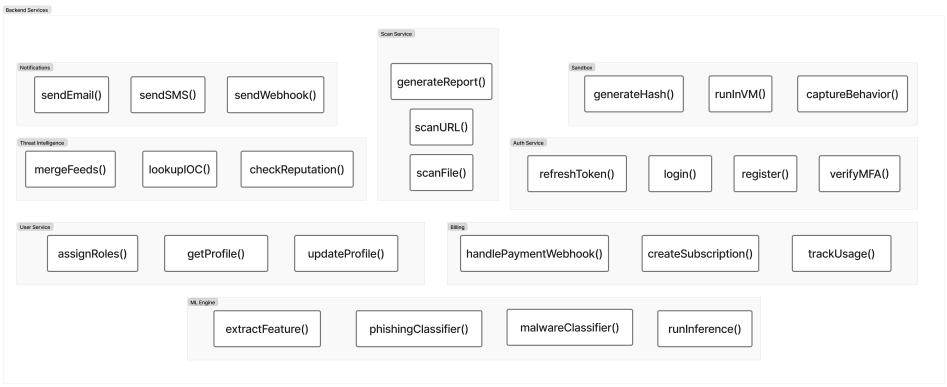

Figure 4.6: Layer 5 - Backend Services Architecture

5.1 User Service

The User Service manages account-level functionality, user profiles, and role assignments using a custom Node.js or Python microservice. This service is responsible for all user-related operations beyond authentication, providing the foundation for personalized user experiences and access control.

The service provides comprehensive functionality including user profile management, account settings updates, role assignment and updates, user directory operations, and permission mapping. Each of these capabilities ensures that users can manage their accounts while administrators maintain proper access control.

The getProfile function accepts user ID from JWT as input and produces user profile data including name, email, role, and preferences as output, reading this information from Neon Postgres in Layer 6. The updateProfile function accepts user ID and updated profile data as input and produces update confirmation as output, writing changes to Neon Postgres and triggering audit log entries.

The assignRoles function accepts user ID and role list as input and produces role assignment confirmation as output. This function updates Authentik RBAC in Layer 4 and Postgres in Layer 6, but requires admin privileges to execute, ensuring that only authorized personnel can modify user permissions.

The service receives requests from the API Gateway in Layer 4, is authenticated by Authentik in Layer 4, stores data in Neon Postgres in Layer 6, logs all actions in Audit Logs Storage in Layer 6 for compliance and security monitoring, and returns data to the User Management UI in Layer 3.

5.2 Scan Service

The Scan Service coordinates security scanning operations for URLs and files using a custom orchestration service, managing the complex workflow of analyzing potentially malicious content through multiple analysis engines. This service is central to the platform's security analysis capabilities.

The service provides critical functionality including URL scanning for phishing and malware, file malware detection, report generation, and scan result aggregation. It orchestrates multiple specialized tools and services to provide comprehensive security analysis.

The scanURL function accepts URLs to scan as input and produces scan results including malicious indicators, reputation scores, and threat levels as output. The process checks URLs against Threat Intelligence in Layer 5.3, queries external feeds such as VirusTotal and URLhaus, runs ML classification through the ML Engine in Layer 5.6, and aggregates all results before storing them in the Scan Results Database in Layer 6.

The scanFile function accepts file IDs from R2 storage as input and produces comprehensive file analysis reports as output. The process retrieves files from Cloudflare R2 in Layer 6, sends them to CAPE Sandbox in Layer 5.7 for dynamic analysis, extracts features for the ML Engine in Layer 5.6, queries Threat Intelligence in Layer 5.3, aggregates all results, and stores them in the Scan Results Database in Layer 6.

The generateReport function accepts scan IDs as input and produces formatted security reports in JSON or PDF format as output. The process aggregates all scan data, formats the report according to templates, stores the report in R2, and returns the report URL to users.

The service receives scan requests from the API Gateway in Layer 4, retrieves files from Cloudflare R2 in Layer 6, sends files to CAPE Sandbox in Layer 5.7 for analysis, sends URLs and features to the ML Engine in Layer 5.6 for classification, queries the Threat Intelligence Service in Layer 5.3 for IOC lookups, stores results in Postgres Scan Results DB in Layer 6, stores reports in Cloudflare R2 in Layer 6, queues long-running jobs in BullMQ in Layer 5.9, and notifies users via the Notification Service in Layer 5.5.

5.3 Threat Intelligence Service

The Threat Intelligence Service provides threat enrichment, IOC lookups, and reputation intelligence from multiple sources using MISP (Malware Information Sharing Platform) as Service #11 and External Threat Feeds including VirusTotal, HybridAnalysis,

AbuseIPDB, and URLhaus as Service #12. This service transforms raw indicators into actionable intelligence.

The service provides essential capabilities including IOC (Indicators of Compromise) lookup, domain, IP, and URL reputation checking, threat feed ingestion and correlation, threat intelligence sharing, and historical threat data storage. These capabilities enable the platform to leverage global threat intelligence for improved detection.

The lookupIOC function accepts IOCs such as IP addresses, domains, file hashes, and URLs as input and produces threat intelligence reports including associated campaigns, malware families, and threat actors as output. The function queries the MISP internal database, checks external APIs, and aggregates results from multiple sources to provide comprehensive intelligence.

The checkReputation function accepts IP addresses, domains, or URLs as input and produces reputation scores (clean, suspicious, or malicious) along with detailed context as output. It queries multiple sources including AbuseIPDB, URLhaus, VirusTotal, and HybridAnalysis in parallel, then aggregates scores to produce an overall assessment.

The mergeFeeds function accepts multiple threat feed sources as input and produces enriched, deduplicated threat intelligence as output. The process ingests feeds from various sources, deduplicates indicators, correlates related information, and stores the results in MISP. This function runs automatically via scheduled jobs managed by BullMQ.

The MISP configuration includes an internal MISP instance for threat storage and correlation, automated feed ingestion from public and private sources, event tagging with MITRE ATT&CK TTPs for standardized threat categorization, sharing groups for collaborative threat intelligence, and API integration with external services.

External threat feeds provide specialized intelligence: VirusTotal API offers file and URL scanning with hash lookup capabilities, HybridAnalysis provides automated malware analysis, AbuseIPDB delivers IP reputation and abuse reports, URLhaus maintains a malicious URL database, and AlienVault OTX contributes open threat intelligence.

The service receives IOC lookup requests from the Scan Service in Layer 5.2, queries the MISP local database and external APIs, stores intelligence in the MISP database backed by Postgres in Layer 6, updates continuously via automated feed ingestion, returns results to the Scan Service in Layer 5.2 for report inclusion, enriches alert data for the SIEM in Layer 7, and feeds into the Threat Map visualization in Layer 3.

5.4 Billing Service

The Billing Service manages subscriptions, usage tracking, and payment processing using Lemon Squeezy as Service #14. This service ensures proper monetization of the platform while providing transparent usage tracking and billing for customers.

The service provides comprehensive billing functionality including subscription plan

management for Freemium, Pro, and Enterprise tiers, payment processing, invoice generation, usage tracking and metering, webhook handling for payment events, and subscription lifecycle management.

The createSubscription function accepts user ID and plan details as input and produces subscription ID and payment URL as output. The process creates a subscription in Lemon Squeezy, returns a checkout URL to the user, and stores subscription details in Postgres Billing Records in Layer 6.

The trackUsage function accepts user ID and resource usage including scans, storage, and API calls as input and produces updated usage metrics as output. The process increments usage counters and checks against plan limits, storing all data in Postgres in Layer 6 to ensure accurate billing and limit enforcement.

The handlePaymentWebhook function accepts webhook events from Lemon Squeezy as input and produces updated subscription status as output. The process validates webhooks, updates subscription information, and sends notifications to users. Events handled include payment success, payment failure, subscription renewal, and subscription cancellation.

The service receives subscription requests from the API Gateway in Layer 4, integrates with the Lemon Squeezy external API for payment processing, receives webhooks from Lemon Squeezy for payment events, stores data in Postgres Billing Records in Layer 6, checks usage against plan limits before allowing scans, notifies users via the Notification Service in Layer 5.5, and blocks actions if usage exceeds plan limits.

5.5 Notification Service

The Notification Service handles all outbound messaging and alert delivery across multiple channels using Resend for email and built-in Webhook Support as Service #13. This service ensures that users and administrators are promptly informed of important events and security findings.

The service provides extensive notification capabilities including email notifications for scan results, alerts, password resets, webhook delivery to external systems such as Slack, Teams, and SIEM, optional SMS notifications, real-time alert distribution, and notification templates with customization options.

The sendEmail function accepts recipient, subject, body, and template information as input and produces email delivery confirmation as output. Use cases include scan completion notifications, vulnerability alerts, password reset instructions, and registration confirmations. The function integrates with the Resend API for reliable email delivery.

The sendWebhook function accepts webhook URL and event payload as input and produces webhook delivery status as output. Use cases include integration with Slack, Teams, Jira, ServiceNow, and external SIEM systems. The format uses JSON payloads

with structured event data for easy integration with third-party systems.

The sendSMS function (optional feature) accepts phone number and message as input and produces SMS delivery confirmation as output. Use cases include critical security alerts and MFA codes for enhanced security.

The service is triggered by all backend services when scan completion, alerts, or auth events occur. It sends notifications via the Resend API for emails, delivers to user inboxes, webhook endpoints, and external systems, logs delivery in Audit Logs in Layer 6, queues messages in BullMQ in Layer 5.9 for reliable delivery, and integrates with Notification Dispatcher in Layer 7 for SIEM alerts.

5.6 ML Engine

The ML Engine powers AI-driven threat detection using machine learning models for phishing and malware classification, utilizing Ollama with Llama 3.2 3B/8B or Mistral-Nemo as Service #10. This service provides intelligent threat detection that adapts and improves over time.

The engine provides sophisticated services including phishing URL detection and classification, malware file classification, feature extraction from files and URLs, real-time inference for threat scoring, and model training and updates.

The phishingClassifier function accepts URLs, page content, and domain features as input and produces phishing probability scores from 0 to 1 along with classification (phishing or legitimate) as output. The process extracts features, runs inference using the model, and returns predictions. The model is a fine-tuned Llama 3.2 trained on comprehensive phishing datasets.

The malwareClassifier function accepts file features from static analysis including PE headers, imports, and strings as input and produces malware probability scores and malware family classification as output. The process performs feature extraction, runs model inference, and produces classification results. The model uses Llama 3.2 8B or Mistral-Nemo trained on extensive malware samples.

The extractFeatures function accepts raw files or URLs as input and produces feature vectors for ML input as output. For files, features extracted include PE structure, imports, entropy, strings, and API calls. For URLs, features extracted include domain age, lexical features, WHOIS data, and SSL information.

The runInference function accepts feature vectors as input and produces model predictions with confidence scores as output. The process loads the appropriate model, runs inference on the features, and returns predictions with confidence levels.

The Ollama configuration includes local model hosting for privacy and performance, GPU acceleration for faster inference, model versioning and A/B testing capabilities, and continuous learning from new threat samples to improve accuracy over time.

The engine receives files and URLs for classification from the Scan Service in Layer 5.2, receives features from CAPE Sandbox in Layer 5.7 based on dynamic analysis data, stores features in the ML Features Database in Layer 6, stores predictions in the Scan Results Database in Layer 6, queues inference jobs in BullMQ in Layer 5.9 for async processing, returns results to the Scan Service in Layer 5.2 for report inclusion, and improves from labeled threat data provided by Threat Intelligence in Layer 5.3.

5.7 Sandbox (Dynamic Malware Analysis)

The Sandbox executes suspicious files in isolated virtual machines to observe runtime behavior and extract malicious activity using CAPE Sandbox as Service #9. This critical component provides deep insight into malware behavior that static analysis cannot reveal.

The sandbox provides comprehensive analysis services including safe execution of suspicious files in isolated VMs, behavioral analysis of network connections, file modifications, and registry changes, API call tracing, memory dumping and analysis, screenshot capture during execution, network traffic capture in PCAP format, dropper and payload extraction, and process tree visualization.

The runInVM function accepts files from Cloudflare R2 and execution parameters as input and produces execution session IDs as output. The process spins up clean VM snapshots, loads files, executes them in isolation, and monitors behavior. VM types available include Windows 7, 10, and 11, Linux, and Android, all configurable based on analysis needs.

The captureBehavior function accepts execution session IDs as input and produces comprehensive behavior reports as output. The system captures network connections including IPs and domains contacted, file system changes including files created, modified, or deleted, registry modifications in Windows systems, process creation trees, API calls and system calls, memory dumps, and screenshots of the execution environment.

The generateHash function accepts analyzed files and extracted artifacts as input and produces file hashes including MD5, SHA1, SHA256, and SSDEEP fuzzy hashes as output. The purpose is IOC generation for threat intelligence sharing and future correlation.

The CAPE configuration includes multiple VM snapshots for different OS versions, network simulation for realistic malware execution, a comprehensive signature database for automated detection, Yara rules for pattern matching, and integration with VirusTotal for hash checking.

The sandbox receives files to analyze from the Scan Service in Layer 5.2, retrieves files from Cloudflare R2 in Layer 6, executes files in isolated VM environment on separate infrastructure, generates behavior reports, IOCs, and artifacts, stores results in Postgres Scan Results DB in Layer 6, stores artifacts including PCAP files, memory dumps, and

screenshots in Cloudflare R2 in Layer 6, sends features to the ML Engine in Layer 5.6 for classification, sends IOCs to Threat Intelligence in Layer 5.3 for correlation, is queued via BullMQ in Layer 5.9 since sandbox jobs can take 5 to 15 minutes, and notifies the Notification Service in Layer 5.5 upon completion.

5.8 Threat Modeling Engine (TIBSA Core)

The Threat Modeling Engine automates threat modeling using the TIBSA methodology, generating DFDs, STRIDE analysis, MITRE ATT&CK mapping, and risk reports using Threagile as Service #17. This engine represents the core intellectual property of the platform, automating complex security analysis that traditionally requires expert manual effort.

The engine provides comprehensive services including automated Data Flow Diagram (DFD) generation, STRIDE threat identification, MITRE ATT&CK TTP mapping, risk scoring and prioritization, security control effectiveness evaluation, and PDF/Word report generation with visualizations.

The generateDFD function accepts system architecture descriptions in JSON or YAML format as input and produces auto-generated DFDs in PNG, SVG, or JSON format as output. The process parses architecture definitions, identifies processes, data stores, and flows, and generates professional diagrams.

The performSTRIDE function accepts DFDs and system components as input and produces STRIDE threat lists covering Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege as output. The process analyzes each component and maps applicable STRIDE threats systematically.

The mapMITRE function accepts identified threats as input and produces MITRE ATT&CK TTPs mapped to threats as output. The process correlates threats with the ATT&CK framework and links specific techniques and tactics, providing standardized threat intelligence nomenclature.

The scoreRisk function accepts threats, likelihood, impact, and existing controls as input and produces risk scores similar to CVSS along with prioritized threat lists as output. The process calculates likelihood multiplied by impact, adjusts for existing controls, and ranks threats by severity.

The generateThreatReport function accepts all threat modeling data as input and produces comprehensive PDF or Word reports with diagrams, threat analysis, and recommendations as output. The process compiles all data, formats reports according to professional templates, and generates visualizations for easy consumption.

The Threagile configuration includes YAML-based architecture definitions, an extensible threat catalog, custom risk formulas inspired by CVSS plus probability calculations, PDF report templates for technical and executive audiences, and integration with the

MITRE ATT&CK database.

The engine receives threat modeling requests from the API Gateway in Layer 4, accepts system architecture from users uploaded as JSON or YAML, generates DFDs, STRIDE analysis, MITRE mapping, and risk scores, stores models in Postgres in Layer 6 for threat models and risk data, stores reports in Cloudflare R2 in Layer 6 as PDF or Word documents, stores diagrams in Cloudflare R2 as PNG or SVG files, queues jobs in BullMQ in Layer 5.9 since threat modeling can be time-intensive, notifies the Notification Service in Layer 5.5 when reports are ready, and returns results to the Frontend in Layer 3 for display in dashboards.

5.9 Background Jobs & Workflow Queue

The Background Jobs & Workflow Queue manages long-running asynchronous tasks without blocking the UI or API using BullMQ with Upstash Redis as Service #18. This system ensures reliable job execution with retry logic, enabling the platform to handle time-consuming operations gracefully.

The queue provides essential services including job queue management for async tasks, reliable job execution with retry and backoff strategies, job prioritization and scheduling, worker process management, job status tracking and monitoring, and dead letter queue for failed jobs.

Sandbox Analysis Jobs are queued by the Scan Service in Layer 5.2 and execute CAPE Sandbox file analysis in Layer 5.7. These jobs typically take 5 to 15 minutes per file and are assigned high priority for user-submitted files to ensure timely results.

ML Inference Jobs are queued by the Scan Service in Layer 5.2 and execute ML Engine classification in Layer 5.6. These jobs typically take 10 to 60 seconds per inference and are assigned medium priority as they are faster than sandbox analysis.

Threat Modeling Jobs are queued by the Threat Modeling Engine in Layer 5.8 and execute full TIBSA analysis including DFD, STRIDE, MITRE, and Report generation. These jobs typically take 2 to 10 minutes per model and are assigned low priority as batch processing is acceptable for these comprehensive analyses.

Threat Feed Ingestion Jobs are queued by Threat Intelligence in Layer 5.3 and execute automated feed ingestion from external sources. These jobs typically take 5 to 30 minutes and are scheduled hourly or daily with low priority as they are background maintenance tasks.

Report Generation Jobs are queued by the Scan Service in Layer 5.2 and Threat Modeling in Layer 5.8, executing PDF or Word report compilation and rendering. These jobs typically take 30 to 120 seconds and are assigned medium priority for timely delivery to users.

The BullMQ configuration includes Redis-backed queue persistence using Upstash

Redis in Layer 6, multiple worker pools for different job types, exponential backoff for failed jobs, concurrency limits to prevent resource exhaustion, job monitoring via Grafana in Layer 7, and rate limiting for external API calls particularly for Threat Intel feeds.

The queue receives jobs from all backend services in Layer 5.2, 5.3, 5.6, 5.7, and 5.8, stores queue in Upstash Redis in Layer 6, executes via worker processes on backend infrastructure, monitors via Prometheus metrics that feed into Grafana dashboards in Layer 7, notifies on completion via the Notification Service in Layer 5.5, updates job status in Postgres in Layer 6 for user visibility, and logs failures in Audit Logs in Layer 6.

4.2.6 LAYER 6 - Data Storage Layer

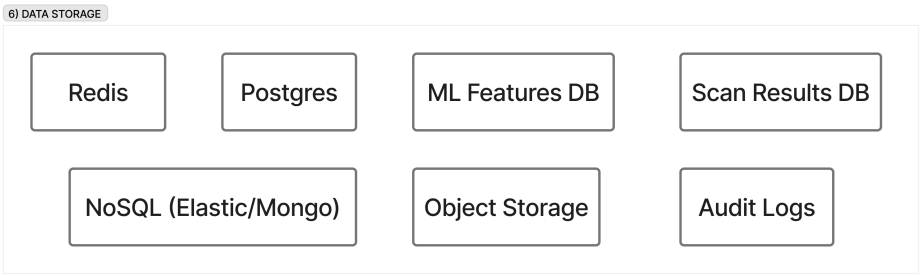

Figure 4.7: Layer 6 - Data Storage Layer Architecture

6.1 Relational Database (Postgres)

The Relational Database serves as the primary storage for all structured data requiring ACID transactions and complex queries using Neon Serverless Postgres as Service #5. This database is the foundation of data persistence for the entire platform, ensuring data integrity and consistency.

The database stores multiple categories of critical data: Users information includes credentials managed via Authentik, profiles, preferences, and roles. Scans data encompasses scan metadata, request parameters, and scan status. Reports information includes report metadata, generation timestamps, and access logs. Threat Models data covers architecture definitions, DFDs, STRIDE analysis, and risk scores. Audit Logs contain all user actions, system events, and administrative changes. Billing Records include subscriptions, invoices, usage metrics, and payment history. System Configuration stores platform settings, feature flags, and RBAC policies.

The schema design includes a users table with id, email, role, created_at, and last_login fields. The scans table contains id, user_id, scan_type, target, status, created_at, and results_json fields. The threat_models table includes id, user_id, architecture_json, dfd_url, stride_results, and risk_score fields. The audit_logs table stores id, user_id,

action, resource, timestamp, and ip_address information. The billing table maintains id, user_id, plan, usage, and subscription_status data.

The database is written by all backend services in Layer 5.1 through 5.9 via ORM, read by all backend services for queries, backed up through automated daily backups provided by Neon, replicated across multiple regions for high availability, and indexed with optimized queries for user_id, scan_id, and timestamps. It connects to Authentik in Layer 4 for user authentication data and is queried by Frontend dashboards in Layer 3 via the API Gateway in Layer 4.

6.2 Cache & Queue (Redis)

The Cache & Queue system provides in-memory caching for performance optimization and queue management for background jobs using Upstash Serverless Redis as Service #6. This high-speed data store dramatically improves platform performance by reducing database load and enabling real-time features.

The system stores multiple types of data: Session Cache maintains active user sessions and JWT refresh tokens. Rate Limiting stores request counters per user and IP for the API Gateway in Layer 4. Query Cache holds frequently accessed data such as user profiles and scan results. Job Queue manages BullMQ job queue in Layer 5.9 for background tasks. Temporary Results stores ML inference results before they are written to Postgres. Real-time Metrics tracks live scan counts, active users, and system load.

The Redis data structures used include Strings for session tokens and cached JSON responses, Hashes for user session data and cached objects, Lists for job queues used by BullMQ, Sets for active users and IP blacklists, and Sorted Sets for rate limit counters with TTL.

The cache is read from and written to by the API Gateway in Layer 4 for rate limiting, used by BullMQ in Layer 5.9 for job queue persistence, utilized by all backend services in Layer 5.1 through 5.8 for performance, serves as session storage for Authentik in Layer 4, manages TTL with auto-expiry for stale cache entries, and falls back to Postgres in Layer 6.1 on cache miss. The system is monitored by Prometheus in Layer 7 for hit and miss rates.

6.3 Object Storage (S3-compatible)

The Object Storage system provides scalable storage for large binary objects, files, and generated artifacts using Cloudflare R2 as Service #7. This cost-effective storage solution handles all large files without egress fees, making it ideal for security artifacts and reports.

The storage holds multiple types of data: Uploaded Files include user-submitted files for scanning such as binaries, documents, and PCAPs. PDF Reports contain generated security reports from the Scan Service in Layer 5.2 and Threat Modeling in Layer 5.8.

Threat Model Diagrams include DFD images in PNG and SVG formats from Threagile in Layer 5.8. Sandbox Artifacts encompass CAPE screenshots, memory dumps, PCAP files, and extracted payloads. Generated Documents include Word reports and executive summaries. User Uploads store profile pictures and team logos if applicable.

The bucket structure organizes files into uploaded-files/ for user-submitted files pending scanning, scan-reports/ for PDF and JSON scan reports, threat-models/ for DFD diagrams and model documents, sandbox-artifacts/ for CAPE analysis outputs including memory dumps, PCAPs, and screenshots, and user-assets/ for profile pictures and logos.

The storage is written to by the File Upload Handler in Layer 3.4 via Tusd, the Scan Service in Layer 5.2 for reports, the Threat Modeling Engine in Layer 5.8 for diagrams and reports, and CAPE Sandbox in Layer 5.7 for artifacts. It is read by the Scan Service in Layer 5.2 to retrieve files for analysis, the Frontend in Layer 3 for report downloads, and CAPE Sandbox in Layer 5.7 to fetch files for execution. Metadata is stored in Postgres in Layer 6.1 including file IDs, URLs, sizes, and timestamps. Access control uses pre-signed URLs for secure, time-limited access, lifecycle policies auto-delete old files after retention periods, and CDN integration with Cloudflare CDN in Layer 2 caches frequently accessed reports.

6.4 Audit Logs Storage

The Audit Logs Storage provides immutable storage of all user actions, system events, and administrative changes for compliance and forensics using a dedicated Postgres table in Neon as Service #5 or a separate append-only log store. This system ensures complete accountability and traceability for all platform activities.

The storage logs multiple categories of events: User Actions include login and logout, file uploads, scan initiations, and report downloads. Administrative Changes cover user creation, role assignments, and configuration updates. System Events track service starts and stops, errors, and security incidents. API Requests log all API calls with timestamps, user IDs, IP addresses, and payloads. Authentication Events record MFA enrollments, password changes, and failed login attempts. Data Access tracks who accessed which reports, threat models, and scan results.

The log structure includes Timestamp for precise event time in ISO 8601 format, User ID for the actor whether user or system, Action for the verb such as created, updated, deleted, or accessed, Resource for the target such as scan ID, user ID, or report URL, IP Address for the source IP, User Agent for browser or client information, Result indicating success or failure, and Metadata providing additional context in JSON format.

The logs are written by all backend services in Layer 5.1 through 5.9 via logging middleware, are immutable as append-only with no updates or deletes, are queried by admin dashboards for compliance reporting, are indexed by timestamp, user_id, and

action for fast searches, are exported to Grafana Loki in Layer 7 for log aggregation, meet compliance requirements for GDPR, SOC 2, and ISO 27001 audit trails, and maintain configurable retention periods such as 7 years for compliance.

6.5 ML Features Database

The ML Features Database provides specialized storage for machine learning training data, feature vectors, and model metadata using dedicated Postgres tables or NoSQL such as MongoDB or Elastic as an extension of Service #5. This specialized storage supports the continuous improvement of the platform's AI capabilities.

The database stores multiple types of ML-specific data: Feature Vectors contain extracted features from files and URLs for ML training. Training Datasets include labeled samples categorized as phishing or benign, malware or clean. Model Metadata tracks model versions, training dates, hyperparameters, and accuracy metrics. Inference Results store ML predictions with confidence scores. Evaluation Metrics maintain precision, recall, F1 scores, and confusion matrices.

The schema design includes a features table with id, file_hash or url, feature_vector in JSON or JSONB, label, and timestamp fields. The models table contains id, model_name, version, trained_at, accuracy, and model_file_url pointing to R2. The predictions table includes id, scan_id, model_id, prediction, confidence, and timestamp fields.

The database is written by the ML Engine in Layer 5.6 for feature extraction and predictions, and by CAPE Sandbox in Layer 5.7 for behavioral features. It is read by the ML Engine in Layer 5.6 for training and inference. The training pipeline performs periodic retraining with new samples, model versioning tracks model improvements over time, A/B testing compares multiple models in production, and the database integrates with Threat Intelligence in Layer 5.3 for labeled threat data.

6.6 Scan Results Database

The Scan Results Database stores comprehensive results from all security scans including URL, file, sandbox, and threat intel using Postgres tables in Neon as Service #5 or NoSQL for flexibility. This database aggregates findings from multiple analysis engines into cohesive scan results.

The database stores multiple categories of scan data: Scan Metadata includes scan ID, user ID, scan type, target, and timestamp. Detection Results contain malicious indicators and threat classifications. Threat Intelligence includes IOC lookups and reputation scores. Sandbox Results contain behavioral analysis summaries. ML Predictions store classification scores from the ML Engine. External Feed Results include VirusTotal and HybridAnalysis responses. Historical Data maintains previous scan results for trending and comparison.

The schema design includes a scans table with id, user_id, scan_type, target, status, and created_at fields. The scan_results table contains id, scan_id, source indicating sandbox, ml, or threat_intel, result_json, and severity fields. The ioc_matches table includes id, scan_id, ioc_type, ioc_value, and threat_intel_source fields.

The database is written by the Scan Service in Layer 5.2 for aggregated scan results, CAPE Sandbox in Layer 5.7 for behavioral analysis, the ML Engine in Layer 5.6 for classification predictions, and Threat Intelligence in Layer 5.3 for IOC matches. It is read by the Scan Service in Layer 5.2 for report generation, Frontend dashboards in Layer 3 for visualization, and the Alerts Engine in Layer 7 for alert triggering. The database is indexed by scan_id, user_id, timestamp, and severity, maintains configurable retention such as keeping the last 90 days and archiving older data, and is exported to the SIEM in Layer 7 for security monitoring.

4.2.7 LAYER 7 - Monitoring & Threat Analysis

Figure 4.8: Layer 7 - Monitoring & Threat Analysis Architecture

7.1 SIEM (Security Information and Event Management)

The SIEM collects, aggregates, and analyzes security events from all platform components to detect attacks on the infrastructure itself using Wazuh Open Source as Service #15. This system provides the security monitoring backbone for the entire platform, protecting the protector.

The SIEM provides comprehensive services including log collection from all services including API Gateway, backend services, and databases, security event correlation, intrusion detection with IDS and IPS capabilities, file integrity monitoring (FIM), vulnerability detection, compliance monitoring for PCI DSS, GDPR, and HIPAA, real-time alerting, and a threat hunting interface.

Data sources include API Gateway logs covering request patterns, authentication failures, and rate limit violations, Backend service logs capturing application errors and suspicious activity, System logs recording OS-level events, process executions, and network connections, Database audit logs tracking unauthorized access attempts and schema changes, and Edge layer logs containing WAF blocks and DDoS events from Cloudflare.

Detection capabilities include identifying brute-force attacks through multiple failed login attempts, detecting SQL injection attempts through malicious query patterns in

logs, monitoring file tampering for unauthorized file modifications, detecting privilege escalation through unexpected role changes, identifying data exfiltration through unusual data transfer patterns, and detecting malware on infrastructure through suspicious process execution.

The SIEM collects from all layers including Edge, Frontend, API, Backend, and Storage, receives logs via Syslog, file monitoring, and API integrations, stores logs in Wazuh Elasticsearch backend, correlates events using a real-time analysis engine, sends alerts to the Alert Engine in Layer 7.2 for prioritization, integrates with Grafana Loki in Layer 7.4 for unified log view, feeds the Threat Dashboard in Layer 7.3 with security metrics, and triggers the Notification Dispatcher in Layer 7.4 for critical alerts.

7.2 Alert Engine

The Alert Engine processes, filters, and prioritizes alerts from SIEM, reducing noise and triggering appropriate responses using custom alert processing logic as part of Wazuh or as a standalone component. This intelligent filtering ensures that security teams focus on genuine threats rather than false positives.

The engine provides critical services including alert deduplication to suppress repeated alerts, severity-based prioritization, context enrichment adding user, asset, and threat intel context, alert correlation to link related events, false positive suppression, automated response triggering, and escalation to human analysts.

The alert processing pipeline follows a structured workflow: First, it receives raw alerts from the SIEM in Layer 7.1. Second, it deduplicates to suppress duplicate alerts within time windows. Third, it enriches alerts by adding context from Postgres for user info and Threat Intel in Layer 5.3. Fourth, it correlates by linking related alerts such as multiple failed logins indicating brute force. Fifth, it prioritizes by scoring based on severity, asset criticality, and threat intel. Sixth, it routes with critical alerts receiving immediate escalation and low priority alerts queued for review. Finally, it responds by triggering automated actions such as blocking IPs, disabling users, or isolating hosts.

The engine receives raw security events from the SIEM in Layer 7.1, enriches with Threat Intelligence from Layer 5.3 for IOC context and Postgres in Layer 6.1 for user and asset data, sends processed alerts to the Threat Dashboard in Layer 7.3 for visualization and the Notification Dispatcher in Layer 7.4 for stakeholder alerts, triggers automated responses such as API calls to block IPs in Cloudflare, and logs actions in Audit Logs in Layer 6.4.

7.3 Threat Dashboard

The Threat Dashboard provides visual interface for security analysts to monitor threats, investigate incidents, and track security posture using Grafana dashboards as part of Ser-

vice #16 combined with custom UI in Next.js from Layer 3. This dashboard transforms raw security data into actionable intelligence.

The dashboard provides multiple visualizations: Active Alerts shows a real-time alert feed with severity indicators. Threat Heatmap displays geographic visualization of attack origins. Attack Timeline provides chronological view of security events. Top Threats highlights most frequent attack types and sources. Asset Risk Scores displays criticality-based asset view. Incident Status tracks open, in-progress, and closed incidents. Alert Trends shows historical alert volume over time. Attack Graphs provides visual representation of multi-step attacks.

Features include drill-down capability to click alerts and view full details and evidence, filtering by severity, time, asset, and alert type, search functionality with full-text search across all security events, export capability for reports for stakeholders and compliance audits, and collaboration features to assign alerts to analysts and add notes.

The dashboard receives data from the Alert Engine in Layer 7.2 for processed, prioritized alerts, the SIEM in Layer 7.1 for raw event data, and Threat Intelligence in Layer 5.3 for IOC context. It displays on the Frontend in Layer 3 specifically in the Security Analyst Console in Layer 1.1, integrates with the Threat Map in Layer 3 for geo-visualization and the Alerts Table in Layer 3 for tabular view, and queries Postgres in Layer 6.1 for historical data.

7.4 Observability Stack (Monitoring, Metrics, Logs)

The Observability Stack provides comprehensive observability for system performance, errors, and operational metrics using Grafana, Prometheus, and Loki as Service #16. This stack ensures that operations teams have complete visibility into platform health and performance.

Prometheus handles metrics collection as a time-series database, providing services including time-series metrics collection, service health monitoring, resource usage tracking for CPU, memory, disk, and network, API response times and error rates, database query performance, queue lengths for BullMQ jobs, and cache hit and miss rates for Redis.

Metrics collected include API Gateway metrics covering requests per second, latency, error rates, and rate limit hits, Backend Services metrics tracking processing times, job queue depths, and error counts, Database metrics monitoring query times, connection pools, and disk usage, ML Engine metrics recording inference times and model accuracy, and Sandbox metrics tracking VM utilization and analysis completion times.

Loki handles log aggregation, providing services including centralized log collection from all services, log querying with LogQL, log correlation with traces, and long-term log storage. Logs aggregated include application logs covering errors, warnings, and info, access logs for API requests, audit logs for user actions, system logs for OS events, and

security logs for authentication and authorization.

Grafana provides visualization and dashboards, offering services including unified dashboard for metrics and logs, custom dashboards per service, alerting based on thresholds, anomaly detection, and performance trending.

Dashboards include Platform Overview showing overall system health, active users, and scan volume, API Gateway displaying request rates, error rates, and latency percentiles, Backend Services showing service-specific metrics and job processing times, Database Performance tracking query times, connection pools, and slow queries, Security Monitoring displaying failed logins, rate limit violations, and WAF blocks, and Cost Monitoring showing resource usage for billing optimization.

The stack scrapes metrics from all backend services in Layer 5.1 through 5.9, the API Gateway in Layer 4.1, and Databases in Layer 6.1 through 6.6. It collects logs from all services via Loki agents, visualizes in Grafana dashboards, sends alerts to the Notification Service in Layer 5.5 on threshold violations, integrates with the SIEM in Layer 7.1 for security event correlation, and is accessed by DevOps team, SREs, and Security analysts.

7.5 Notification Dispatcher

The Notification Dispatcher routes alerts and notifications to appropriate stakeholders via multiple communication channels using Resend as Service #13 for emails and webhook integrations for Slack, Teams, and Jira. This system ensures that critical information reaches the right people through their preferred channels.

The dispatcher provides essential services including email alert delivery for critical security events and system outages, Slack and Teams integration for real-time team notifications, ticketing system integration with Jira and ServiceNow for incident tracking, optional SMS alerts for critical events via Twilio integration, and escalation management to notify senior analysts if alerts go unaddressed.

Notification types include Critical Security Alerts for brute-force attacks, data breaches, and infrastructure compromise, System Health notifications for service outages, database failures, and high error rates, User Notifications for scan completions and report readiness from Layer 5, and Operational Alerts for queue backlogs, resource exhaustion, and failed jobs.

The dispatcher receives from the Alert Engine in Layer 7.2 for security alerts, Prometheus in Layer 7.4 for system health alerts, and the SIEM in Layer 7.1 for critical security events. It delivers via Resend for email, Webhooks for Slack, Teams, and Jira, and optionally SMS. The dispatcher logs delivery in Audit Logs in Layer 6.4 and escalates unacknowledged critical alerts to senior staff.

4.2.8 LAYER 8 - TIBSA Suite (Core)

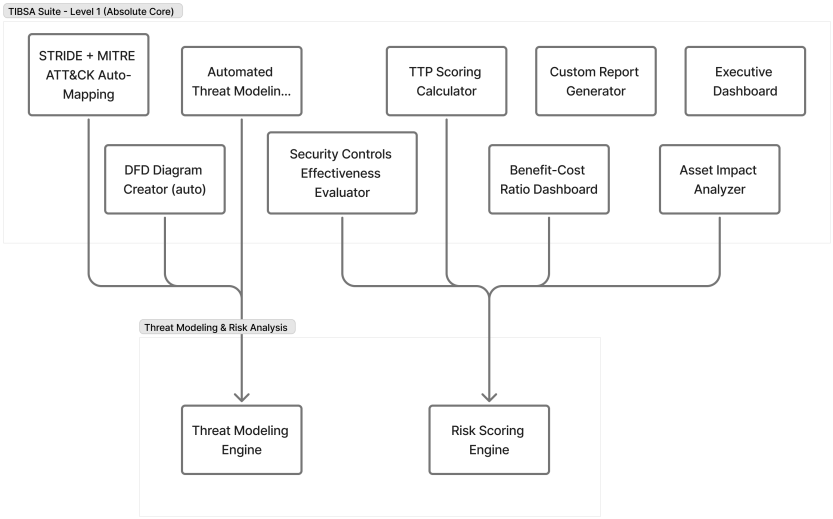

Figure 4.9: Layer 8 - TIBSA Suite Core Architecture

8.1 TIBSA Suite - Level 1 (Absolute Core)

The TIBSA Suite represents the foundational automation engines that execute the complete threat-modeling and risk-analysis workflow, transforming raw system information into structured models, identified threats, calculated risks, and actionable reports using Threagile as Service #17 as the core automation engine.

The Automated Threat Modeling Engine applies the complete TIBSA methodology automatically, accepting system architecture in JSON or YAML format, asset inventory, and data flows as input and producing complete threat models with identified threats as output. The process parses architecture, identifies assets, maps data flows, detects threat patterns, and produces comprehensive models.

The DFD Diagram Creator generates Data Flow Diagrams automatically, accepting parsed system descriptions as input and producing structured DFDs in PNG, SVG, or JSON format as output showing processes, data stores, external entities, and data flows. The process identifies components, maps relationships, generates visual diagrams, and stores them in Cloudflare R2 in Layer 6.

The STRIDE and MITRE ATT&CK Auto-Mapping component provides standardized threat categorization and TTP mapping, accepting system components, DFDs, and identified weak points as input and producing STRIDE threat categories plus correlated MITRE ATT&CK techniques as output. The process maps components to STRIDE categories covering Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege, correlates threats with MITRE ATT&CK TTPs,

and links to real-world attack patterns ensuring industry-aligned, standardized threat identification.

The TTP Scoring Calculator provides risk scoring for each threat and TTP, accepting identified threats, likelihood factors, impact assessment, exploitation difficulty, and environmental context as input and producing risk scores on a 0 to 10 scale along with likelihood and impact ratings as output. The scoring model considers likelihood based on threat actor capability, attack complexity, and existing controls, impact based on asset criticality, data sensitivity, and business disruption, and combines these into a combined score calculated as likelihood multiplied by impact with contextual adjustments using a formula similar to CVSS but enhanced with probability estimation.

The Security Controls Effectiveness Evaluator assesses existing security controls against identified threats, accepting organization's security controls inventory and identified threats as input and producing control effectiveness ratings, coverage gaps, redundancies, and priority controls needed as output. The evaluation determines which threats are adequately mitigated, which threats lack sufficient controls, which controls are redundant or ineffective, and the cost-benefit of proposed controls.

The Benefit-Cost Ratio Dashboard provides cost-effectiveness analysis for control recommendations, accepting recommended controls, implementation costs, and expected risk reduction as input and producing B/C ratios and prioritized control recommendations ranked by cost-effectiveness as output. This helps decision-makers allocate security budget optimally.

The Asset Impact Analyzer provides business and technical impact assessment for critical assets, accepting asset inventory, business criticality, and dependencies as input and producing asset criticality scores, impact categories covering financial, operational, reputational, and compliance aspects, and priority rankings as output. This supports risk-based decision-making and resource allocation.

The Custom Report Generator automates comprehensive report generation, accepting all threat modeling data including DFD, STRIDE, MITRE, risks, and controls as input and producing PDF or Word reports as output. The reports include executive summary, technical threat analysis, risk scores and rankings, recommended controls with priorities, diagrams and visualizations, and compliance mappings. Templates are customizable for different audiences including technical, executive, and compliance stakeholders.

The Executive Dashboard provides high-level leadership dashboard, accepting aggregated threat modeling results as input and producing single-page summary as output. The summary includes top threats and risk levels, critical assets at risk, recommended priority actions, overall security posture score, and trend indicators showing whether security is improving or worsening. The purpose is to enable rapid executive decision-making.

The suite receives threat modeling requests from the API Gateway in Layer 4, accepts

input data from users via the Frontend in Layer 3 in the form of architecture descriptions, executes via the Threagile automation engine, stores models in Postgres in Layer 6 for threat data and risk scores, stores artifacts in Cloudflare R2 in Layer 6 for DFD diagrams and PDF or Word reports, queues jobs in BullMQ in Layer 5.9 for async processing taking 2 to 10 minutes, integrates with the MITRE ATT&CK database for TTP mapping, notifies via the Notification Service in Layer 5.5 when reports are ready, displays in the Frontend Dashboard in Layer 3 specifically in the Executive Dashboard component, and exports to the SIEM in Layer 7 for security monitoring integration.

4.2.9 Key Integration Points Summary

The authentication flow moves from Cloudflare through Next.js to KrakenD to Authentik and finally to Backend Services. The file analysis pipeline flows from Uppy/Tusd to R2 to CAPE Sandbox to ML Engine to MISP and culminates in comprehensive reports. The threat modeling workflow begins with user input, processes through Threagile to produce DFD, STRIDE, and MITRE analysis, and stores reports in R2. Background processing flows from all services through BullMQ to Redis Queue where workers process jobs and store results in appropriate storage. The monitoring flow captures data from all services, sends it to Prometheus and Loki, displays it in Grafana Dashboards, and generates alerts as needed. Security monitoring aggregates all logs in Wazuh SIEM, processes them through the Alert Engine, and distributes notifications via the Notification Dispatcher. Data persistence is handled with all services storing structured data in Postgres, utilizing Redis for cache, and storing large objects in R2.

4.3 Object-Oriented Class Design

This section presents the object-oriented class design of the TIBSA platform, illustrating the structural relationships between the system's core components through a comprehensive Unified Modeling Language (UML) class diagram. The class diagram provides a detailed view of the system's software architecture, showcasing the classes, their attributes, methods, and the associations that bind them together. This design follows established object-oriented principles including encapsulation, inheritance, and composition to ensure modularity, maintainability, and extensibility of the codebase.

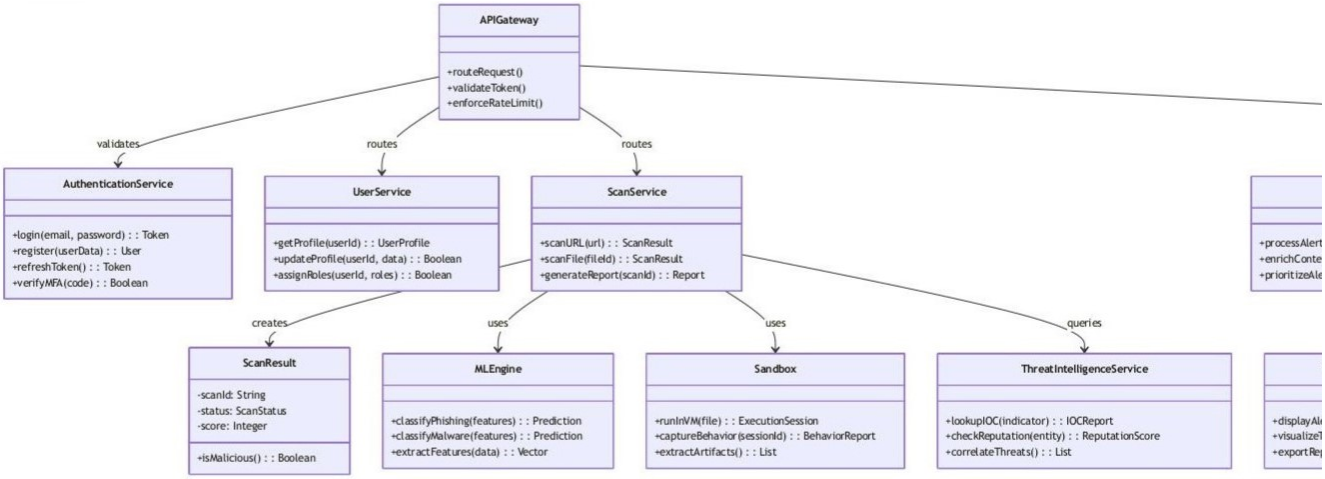

Figure 4.10: TIBSA Platform - Comprehensive Class Diagram (Part 1 of 2)

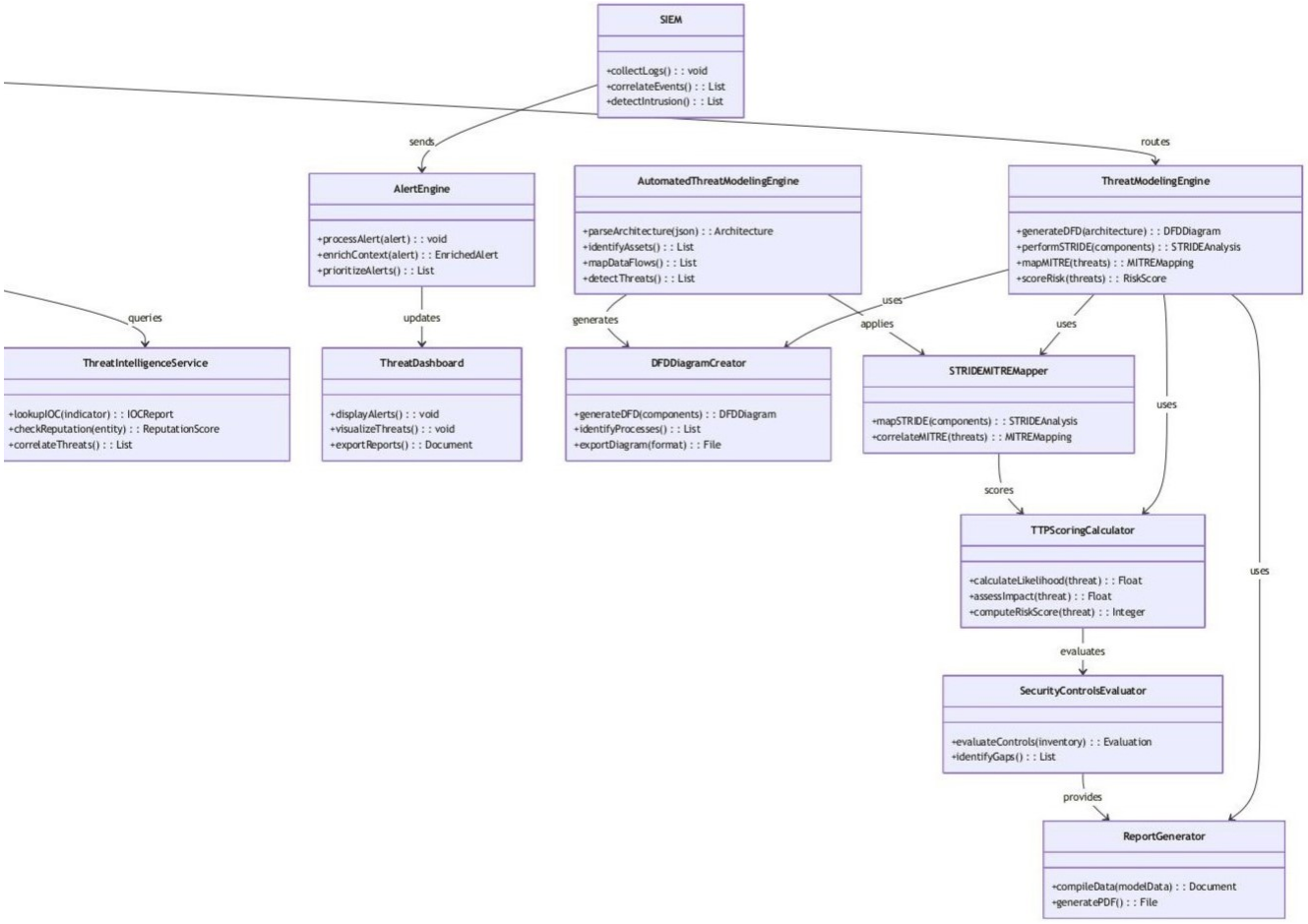

Figure 4.11: TIBSA Platform - Comprehensive Class Diagram (Part 2 of 2)

4.3.1 Core Classes Overview

The TIBSA platform's class structure is organized around several core domain entities that represent the fundamental building blocks of the system. Each class encapsulates specific functionality and data, with well-defined interfaces for interaction with other components. The following subsections provide detailed descriptions of each class, their attributes, methods, and relationships.

4.3.2 URLScanning Class

The URLScanning class serves as the central component responsible for analyzing web addresses for potential security threats such as phishing attempts, malware distribution, and malicious content. This class encapsulates all functionality related to URL-based threat detection and analysis. The class maintains several key attributes including `scanID` which serves as a unique identifier assigned to each URL scanning operation for tracking and reference purposes, `targetURL` representing the web address submitted for security analysis, `scanStatus` indicating the current state of the scanning operation (pending, in-progress, completed, or failed), and `createdAt` which records the timestamp when the scan request was initiated.

The class provides essential methods for URL analysis operations. The `initiateScan()` method initiates the URL scanning process by sending the target URL to the analysis engines. The `getScanResults()` method retrieves the complete analysis results including threat indicators and reputation scores. The `cancelScan()` method terminates an ongoing scan operation and releases associated resources.

4.3.3 AuthenticationHandler Class

The AuthenticationHandler class manages all aspects of user authentication and session management within the TIBSA platform. It serves as the primary interface between the user interface layer and the authentication service, ensuring secure access control throughout the system. The class attributes include `sessionID` which provides a unique token identifying the current user session, `userEmail` containing the email address associated with the authenticated user, `loginAttempts` as a counter tracking failed login attempts for security monitoring, `isAuthenticated` as a flag indicating whether the current session is authenticated, `mfaEnabled` indicating whether multi-factor authentication is enabled for the account, and `lastLogin` recording the timestamp of the user's most recent successful login.

The class implements comprehensive authentication methods. The `registerUser(email, password)` method creates a new user account with the provided credentials. The `authenticateUser(email, password)` method validates

user credentials and issues authentication tokens. The `validateToken(token)` method verifies the validity and expiration status of authentication tokens. The `refreshToken(refreshToken)` method issues new access tokens using valid refresh tokens. The `verifyMFA(code)` method validates multi-factor authentication codes. The `logout()` method terminates the current session and invalidates associated tokens.

4.3.4 UserServices Class

The UserServices class provides comprehensive user account management functionality, handling user profiles, preferences, and account-related operations. This class acts as the service layer for all user-centric operations beyond authentication. The class maintains attributes including `userID` as the unique identifier for the user account, `userName` representing the display name of the user, `userEmail` containing the primary email address associated with the account, `userRole` indicating the role assigned to the user (Admin, Analyst, User, Guest), `profileData` as a container for additional user profile information, and `createdAt` recording the account creation timestamp.

The class provides several methods for user management. The `getProfile(userID)` method retrieves the complete profile data for a specified user. The `updateProfile(userID, data)` method updates user profile information with the provided data. The `deleteAccount(userID)` method permanently removes a user account and associated data. The `getUserRole(userID)` method returns the current role assignment for a user. The `updatePreferences(userID, prefs)` method modifies user preferences and settings.

4.3.5 ScanServices Class

The ScanServices class orchestrates all security scanning operations within the platform, coordinating between file analysis, URL scanning, and threat intelligence services. This class serves as the central hub for initiating, managing, and tracking scan operations across the entire system. The class attributes include `scanID` as the unique identifier for the scan operation, `scanType` indicating the classification of the scan (file, URL, or hash lookup), `targetIdentifier` representing the subject of the scan (file path, URL, or hash value), `priority` indicating the processing priority level for the scan request, `status` showing the current operational status of the scan, and `initiatedBy` containing the user ID of the requesting user.

The class implements comprehensive scanning methods. The `createScan(type, target, userID)` method initializes a new scan operation with specified parameters. The `getScanStatus(scanID)` method returns the current status and progress of a scan. The `getScanResults(scanID)` method retrieves completed scan results and analysis data. The `cancelScan(scanID)` method terminates an in-progress scan operation. The

`queueScan(scan)` method adds a scan to the processing queue for asynchronous execution. The `aggregateResults(scanID)` method combines results from multiple analysis engines into a unified report.

4.3.6 FileAnalysis Class

The FileAnalysis class handles the processing and security analysis of uploaded files, implementing both static and dynamic analysis techniques. This class integrates with sandbox environments, machine learning engines, and threat intelligence services to provide comprehensive file security assessments. The class maintains attributes including `fileID` as the unique identifier assigned to the uploaded file, `fileName` containing the original name of the uploaded file, `fileHash` representing the SHA-256 hash of the file content for integrity verification, `fileSize` indicating the size of the file in bytes, `mimeType` containing the MIME type classification of the file, `uploadDate` recording the timestamp of file upload, and `analysisStatus` indicating the current state of the analysis process.

The class provides essential methods for file analysis. The `uploadFile(file)` method processes and stores an uploaded file for analysis. The `calculateHash(file)` method computes cryptographic hashes (MD5, SHA-1, SHA-256) for the file. The `extractMetadata(fileID)` method extracts file properties and metadata information. The `initiateStaticAnalysis(fileID)` method begins static analysis of file structure and content. The `initiateDynamicAnalysis(fileID)` method submits the file to sandbox environment for behavioral analysis. The `getAnalysisReport(fileID)` method retrieves the comprehensive analysis report for a file.

4.3.7 PentestJobService Class

The PentestJobService class manages penetration testing operations within the TIBSA platform, handling job scheduling, execution coordination, and results collection for automated security assessments. The class attributes include `jobID` as the unique identifier for the penetration testing job, `targetScope` defining the target scope for testing, `testModules` as a collection of testing modules to be executed, `scheduledTime` indicating the scheduled execution time for the job, and `jobStatus` representing the current status of the pentest job.

The class implements methods for penetration testing management. The `createJob(scope, modules)` method creates a new penetration testing job with specified parameters. The `scheduleJob(jobID, time)` method schedules a job for future execution. The `executeJob(jobID)` method initiates the execution of a pentest job. The `getJobResults(jobID)` method retrieves the findings and results from a completed job. The `cancelJob(jobID)` method cancels a scheduled or running job.

4.3.8 ThreatIntelService Class

The ThreatIntelService class provides integration with threat intelligence sources, enabling the platform to enrich security analysis with external threat data, indicators of compromise (IOCs), and reputation information. The class maintains attributes including `queryID` as the unique identifier for threat intelligence queries, `indicatorType` indicating the type of indicator being queried (IP, domain, hash, URL), `indicatorValue` containing the actual indicator value for lookup, `sources` as a list of threat intelligence sources to query, and `lastUpdated` recording the timestamp of the most recent data refresh.

The class provides methods for threat intelligence operations. The `lookupIOC(indicator)` method queries threat intelligence sources for indicator information. The `getReputation(target)` method retrieves reputation scores for domains, IPs, or URLs. The `enrichData(scanResults)` method augments scan results with threat intelligence context. The `syncFeeds()` method synchronizes local threat data with external feed sources. The `queryMISP(indicator)` method queries the MISP platform for related threat events.

4.3.9 MLEngineService Class

The MLEngineService class encapsulates the machine learning capabilities of the TIBSA platform, providing intelligent threat detection through trained models for phishing classification, malware detection, and anomaly identification. The class attributes include `modelID` as the identifier for the machine learning model in use, `modelVersion` indicating the version number of the deployed model, `modelType` representing the classification of the model (phishing, malware, anomaly), `accuracy` containing the current accuracy metric of the model, and `lastTrained` recording the timestamp of the most recent model training.

The class implements machine learning methods. The `classifyURL(url, features)` method classifies a URL as phishing or legitimate. The `classifyFile(fileFeatures)` method determines malware probability for a file. The `extractFeatures(input)` method extracts relevant features for model input. The `runInference(features)` method executes model inference on provided feature vectors. The `getConfidenceScore(prediction)` method returns the confidence level of a prediction.

4.3.10 AccountManager Class

The AccountManager class handles subscription management, usage tracking, and account tier administration, integrating with billing services to manage user entitlements and resource allocation. The class maintains attributes including `accountID` as the

unique identifier for the billing account, `subscriptionPlan` indicating the current subscription tier (Freemium, Pro, Enterprise), `usageQuota` representing the maximum allowed resource usage for the account, `currentUsage` tracking the current resource consumption count, `billingCycle` indicating the billing period (monthly, annual), and `expirationDate` recording the subscription expiration date.

The class provides methods for account management. The `getSubscriptionDetails(accountID)` method retrieves current subscription information. The `updateSubscription(accountID, plan)` method modifies the subscription plan. The `trackUsage(accountID, resource)` method records resource usage for billing purposes. The `checkQuota(accountID)` method verifies if the account has remaining quota. The `generateInvoice(accountID)` method creates billing invoices for the account.

4.3.11 ThreatAnalysisEngine Class

The ThreatAnalysisEngine class represents the core threat modeling functionality of the TIBSA platform, automating the generation of threat models, STRIDE analysis, and risk assessments based on system architecture inputs. The class attributes include `modelID` as the unique identifier for the threat model, `architectureInput` containing the system architecture description in JSON or YAML format, `dfdGenerated` as a flag indicating whether DFD has been generated, `riskScore` representing the calculated overall risk score for the system, and `analysisDate` recording the timestamp of the threat analysis execution.

The class implements comprehensive threat modeling methods. The `generateDFD(architecture)` method creates Data Flow Diagrams from architecture specifications. The `performSTRIDE(dfd)` method executes STRIDE threat analysis on the model. The `mapMITRE(threats)` method maps identified threats to MITRE ATT&CK framework. The `calculateRisk(threats, controls)` method computes risk scores based on likelihood and impact. The `generateReport(modelID)` method produces comprehensive threat modeling reports.

4.3.12 ITScanAPIWrapper Class

The ITScanAPIWrapper class provides a unified interface for interacting with external scanning APIs and antivirus engines, abstracting the complexity of multiple vendor integrations behind a consistent API. The class maintains attributes including `apiEndpoint` containing the base URL for the external API, `apiKey` as the authentication key for API access, `engineName` representing the name of the integrated scanning engine, `isActive` indicating whether the integration is currently active, and `rateLimitRemaining` tracking the remaining API calls within the rate limit period.

The class provides methods for external API integration. The `submitScan(target)` method submits a target to the external scanning service. The `getResults(scanID)` method retrieves scan results from the external service. The `checkStatus(scanID)` method queries the status of an external scan operation. The `validateAPIKey()` method verifies the validity of the configured API key.

4.3.13 FTPThreagileInput Class

The FTPThreagileInput class manages the input processing for the Threagile threat modeling engine, handling file transfers and format conversions required for automated threat model generation. The class attributes include `inputID` as the unique identifier for the input request, `inputFormat` indicating the format of the input file (JSON, YAML), `validationStatus` representing the status of input validation, and `processingQueue` indicating the queue assignment for processing.

The class implements input processing methods. The `validateInput(data)` method validates input data against schema requirements. The `transformFormat(input, targetFormat)` method converts input between supported formats. The `queueForProcessing(inputID)` method adds validated input to the processing queue.

4.3.14 SecurityControlsEvaluator Class

The SecurityControlsEvaluator class assesses the effectiveness of existing security controls against identified threats, providing gap analysis and recommendations for security improvements. The class maintains attributes including `evaluationID` as the unique identifier for the evaluation session, `controlsInventory` as a list of security controls under evaluation, `threatsCovered` representing the threats addressed by current controls, and `gapsIdentified` containing the security gaps requiring attention.

The class provides methods for security controls evaluation. The `evaluateControls(controls, threats)` method assesses control effectiveness against threats. The `identifyGaps(evaluation)` method identifies areas lacking adequate protection. The `recommendControls(gaps)` method suggests controls to address identified gaps. The `calculateCoverage(controls)` method computes the percentage of threats mitigated.

4.3.15 ReportGenerator Class

The ReportGenerator class handles the creation and formatting of all reports within the TIBSA platform, including scan reports, threat model documentation, and executive summaries. The class maintains attributes including `reportID` as the unique identifier for

the generated report, `reportType` indicating the classification of the report (scan, threat model, executive), `outputFormat` representing the desired output format (PDF, Word, JSON), `generatedAt` recording the timestamp of report generation, and `storageURL` containing the location where the report is stored.

The class implements report generation methods. The `compileReportData(sourceID)` method aggregates data required for report generation. The `generatePDF(data)` method creates PDF format reports with visualizations. The `generateWord(data)` method produces Word format documentation. The `exportJSON(data)` method exports report data in JSON format for integration.

4.3.16 Class Relationships and Associations

The classes within the TIBSA platform exhibit various types of relationships that define how objects interact and collaborate to fulfill system functionality. The following describes the key associations illustrated in the class diagram:

Composition Relationships

The ScanServices class maintains a composition relationship with both URLScanning and FileAnalysis classes, indicating that scan operations are composed of specific scanning activities. When a ScanServices instance is destroyed, the associated URLScanning and FileAnalysis instances are also terminated, reflecting the lifecycle dependency between these components.

Association Relationships

The AuthenticationHandler class is associated with the UserServices class, enabling user authentication to be linked with profile management. This association allows authenticated sessions to access and modify user-specific data while maintaining separation of concerns between authentication and user management logic.

The ThreatIntelService class associates with the ScanServices class, providing threat intelligence enrichment capabilities to scan operations. This relationship enables scan results to be augmented with external threat data, IOC matches, and reputation information.

Dependency Relationships

The MLEngineService class depends on the FileAnalysis and URLScanning classes for feature extraction, as machine learning inference requires preprocessed features derived from analyzed files and URLs. This dependency ensures that ML predictions are based on properly extracted and validated input data.

The ThreatAnalysisEngine class depends on the FTPThreagileInput class for receiving properly formatted architecture specifications, and on the SecurityControlsEvaluator class for assessing control effectiveness during threat modeling operations.

Aggregation Relationships

The AccountManager class aggregates UserServices instances, reflecting that a single billing account may encompass multiple user accounts within an organization. This aggregation supports team-based subscriptions and centralized usage tracking across multiple users.

4.3.17 Design Patterns Employed

The class design incorporates several established software design patterns to promote code quality, maintainability, and extensibility:

- **Service Layer Pattern:** Classes such as UserServices, ScanServices, and ThreatIntelService implement the service layer pattern, encapsulating business logic and providing clean interfaces for upper layers.
- **Factory Pattern:** The ScanServices class employs factory methods to create appropriate scanning instances (URLScanning or FileAnalysis) based on the type of target being analyzed.
- **Wrapper Pattern:** The ITScanAPIWrapper class implements the wrapper pattern to provide a unified interface for multiple external scanning APIs, simplifying integration and maintenance.
- **Strategy Pattern:** The MLEngineService class utilizes the strategy pattern to select appropriate classification models based on the type of input being analyzed (URL vs. file).
- **Observer Pattern:** The notification and status update mechanisms employ the observer pattern, allowing UI components to react to changes in scan status and analysis completion.

4.3.18 Class Design Summary

The class diagram presented in this section provides a comprehensive view of the TIBSA platform's object-oriented architecture. The design emphasizes modularity through well-defined class boundaries, maintainability through clear separation of concerns, and extensibility through the use of established design patterns. Each class encapsulates specific domain functionality while maintaining loose coupling with other components through

well-defined interfaces. This architectural approach ensures that the system can evolve to accommodate new features, additional scanning engines, and enhanced threat intelligence capabilities without requiring fundamental structural changes.

4.4 Entity-Relationship Diagram (ERD)

This section presents the entity-relationship design of the cybersecurity platform, illustrating the structural organization of data and the relationships between the system's core entities through a comprehensive Entity-Relationship Diagram (ERD). The ERD provides a detailed view of the platform's database schema, showcasing tables, their attributes, primary and foreign keys, and the associations that connect them. This design follows established database normalization principles to ensure data integrity, consistency, and efficient storage, while also supporting the platform's functional requirements such as scan management, threat modeling, and mitigation tracking. By mapping entities and their relationships in a structured manner, the ERD forms the foundation for both backend implementation and future scalability of the system.

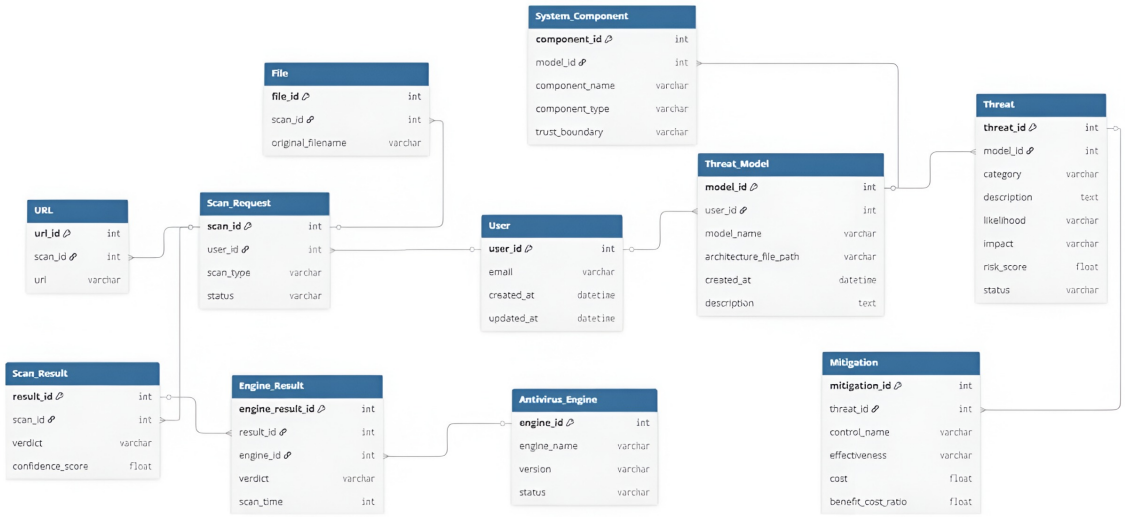

Figure 4.12: TIBSA Platform - Entity-Relationship Diagram (ERD)

4.4.1 User Table

The User table stores information about registered users of the cybersecurity platform, serving as the primary entity for user management. Its attributes include `user_id`, which uniquely identifies each user, email for communication and login purposes, and timestamps `created_at` and `updated_at` to track account creation and modifications. Users are responsible for submitting scan requests and creating threat models, forming the basis of the platform's user-specific operations.

4.4.2 Scan_Request Table

The `Scan_Request` table manages individual scan operations initiated by users. Key attributes include `scan_id` as the unique identifier, `user_id` linking the scan to the

submitting user, `scan_type` specifying the kind of scan (e.g., file or URL), and `status` indicating the current state of the scan, such as pending, completed, or failed. This table ensures that every scan is traceable to a user and forms the foundation for subsequent scan results.

4.4.3 File Table

The File table handles all files submitted for analysis. Its attributes include`file_id` as the unique identifier, `scan_id` referencing the associated scan request, and `original_filename` storing the name of the submitted file. Each file is linked to a specific scan request, allowing the system to track which files are being analyzed in each scanning operation.

4.4.4 URL Table

The URL table stores URLs submitted for scanning. It includes `url_id` as the primary key, `scan_id` linking it to the related scan request, and `url` containing the web address being analyzed. This entity parallels the File table and provides a structured way to manage web-based submissions for threat detection.

4.4.5 Scan_Result Table

The Scan_Result table captures the outcomes of scan requests, linking back to the corresponding Scan_Request via `scan_id`. It contains `result_id` as the unique identifier, `verdict` summarizing the outcome of the scan, and `confidence_score` to indicate the certainty of the results. This table allows for detailed tracking of each scan's outcome and supports subsequent analysis by antivirus engines.

4.4.6 Antivirus_Engine Table

The `Antivirus_Engine` table maintains information about the antivirus engines utilized for scanning. Its attributes include `engine_id` as the unique identifier, `engine_name`, `version`, `status` to indicate operational readiness, and `last_update` for tracking the engine's current version. Each engine can execute multiple scans and contribute to the overall scan results.

4.4.7 Engine_Result Table

The `Engine_Result` table records the detection results provided by individual antivirus engines for specific scan results. Its attributes include `engine_result_id` as the unique

identifier, `result_id` linking to the parent `Scan_Result`, `engine_id` linking to the responsible `Antivirus_Engine`, `verdict` detailing the engine's analysis, and `scan_time` measuring the duration of the scan. This table allows detailed attribution of detections to individual engines.

4.4.8 Threat_Model Table

The `Threat_Model` table represents user-created threat models. Attributes include `model_id` as the unique identifier, `user_id` linking to the creator, `model_name` for the model title, `architecture_file_path` for storing the path to the model file, `description` providing context, and timestamps `created_at` and `updated_at`. Each threat model can include multiple system components and identified threats, forming the core structure for threat analysis.

4.4.9 System_Component Table

The `System_Component` table defines individual components within a threat model. Attributes include `component_id` as the unique identifier, `model_id` referencing the parent threat model, `component_name`, `component_type`, and `trust_boundary` specifying the security zone of the component. This table allows threat models to be structured according to system architecture.

4.4.10 Threat Table

The `Threat` table stores threats identified within a threat model. Key attributes include `threat_id` as the unique identifier, `model_id` linking to the parent threat model, `category` and `description` detailing the nature of the threat, `likelihood` and `impact` for risk assessment, `risk_score` as a calculated metric, and `status` to indicate whether the threat is active or mitigated. Each threat can be associated with multiple `Mitigation` entries.

4.4.11 Mitigation Table

The `Mitigation` table represents the security controls applied to threats. Attributes include `mitigation_id` as the unique identifier, `threat_id` linking to the associated threat, `control_name` describing the mitigation, `effectiveness`, `cost`, and `benefit_cost_ratio` for evaluating resource efficiency. This table provides a structured way to manage and track threat countermeasures within the system.

4.4.12 Relationships and Cardinality

Composition Relationships

The `Scan_Request` table maintains a composition relationship with both the `File` and `URL` tables, indicating that each scan request is composed of a specific file or URL to be analyzed. When a `Scan_Request` instance is deleted, the associated `File` or `URL` instance is also removed, reflecting the lifecycle dependency between a scan operation and the content being scanned. Similarly, the `Scan_Result` table is composed of multiple `Engine_Result` instances, representing the outcomes from different antivirus engines. Deletion of a `Scan_Result` instance cascades to its related `Engine_Result` instances, maintaining the integrity of engine-level scan data.

Association Relationships

The `User` table is associated with the `Scan_Request` table, establishing that users can submit multiple scans while each scan belongs to a single user. The `User` table is also associated with the `Threat_Model` table, allowing users to create and manage multiple threat models, each linked back to the creator for traceability.

The `Threat_Model` table associates with both the `System_Component` and `Threat` tables, reflecting that each threat model contains multiple system components and identifies multiple threats. Furthermore, the `Threat` table is associated with the `Mitigation` table, establishing that each identified threat may have multiple mitigation controls applied to reduce its risk.

The `Antivirus_Engine` table is associated with the `Engine_Result` table, indicating that each antivirus engine can generate multiple results for different scan outcomes. The `Scan_Request` table is also associated with the `Scan_Result` table, enabling the system to link scan submissions with their corresponding results, which can be further enriched by antivirus engines.

Multiplicity Summary

Each `User` can submit one or more `Scan_Request` instances, while each `Scan_Request` is linked to exactly one `User`. Each `Scan_Request` is associated with exactly one `File` or `URL`. Each `Scan_Result` can contain multiple `Engine_Result` instances, while each `Engine_Result` belongs to exactly one `Scan_Result` and is generated by exactly one `Antivirus_Engine`.

Each `User` can create multiple `Threat_Model` instances, with each model belonging to exactly one `User`. Each `Threat_Model` can include multiple `System_Component` instances and identify multiple `Threat` instances. Each `Threat` may have multiple associated `Mitigation` entries, while each `Mitigation` is linked to exactly one `Threat`.

These multiplicity rules ensure data integrity, traceability, and structured workflows, where all scan operations, threat models, and mitigation measures are fully connected and auditable.

4.5 User Interface Wireframes

This section presents the key user interface wireframes that illustrate how users interact with the TIBSA platform. Each scenario demonstrates a specific functionality of the system, from initial authentication through to advanced features such as file scanning, threat analysis, and system administration. The wireframes are designed to prioritize usability, security, and clarity, ensuring that both technical and non-technical users can effectively navigate the platform. The scenarios cover the complete user journey, including authentication flows, dashboard navigation, security scanning operations, and administrative functions.

4.5.1 Login Screen

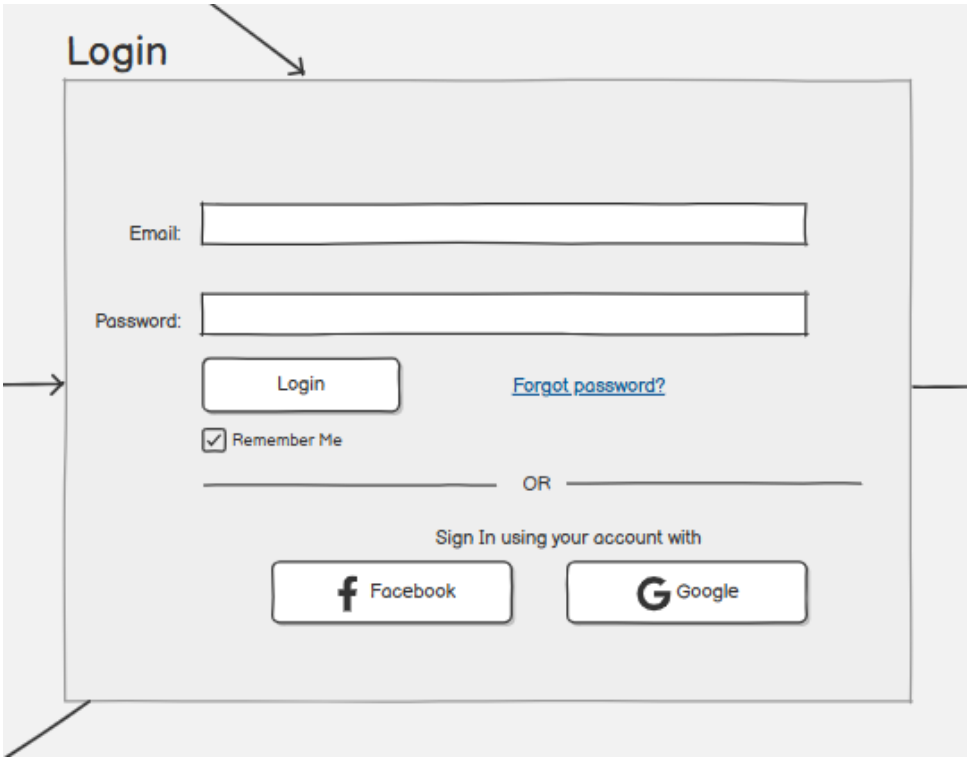

Figure 4.13: Login Screen

The Login screen is the main entry point to the system. It allows registered users to authenticate using their email and password. The screen contains clearly labeled input fields for credentials and a primary login button to initiate authentication. Additional options such as “Remember Me” and “Forgot Password” are included to enhance usability and account recovery. This wireframe focuses on simplicity and security by limiting distractions and guiding the user directly toward authentication.

4.5.2 Registration Screen

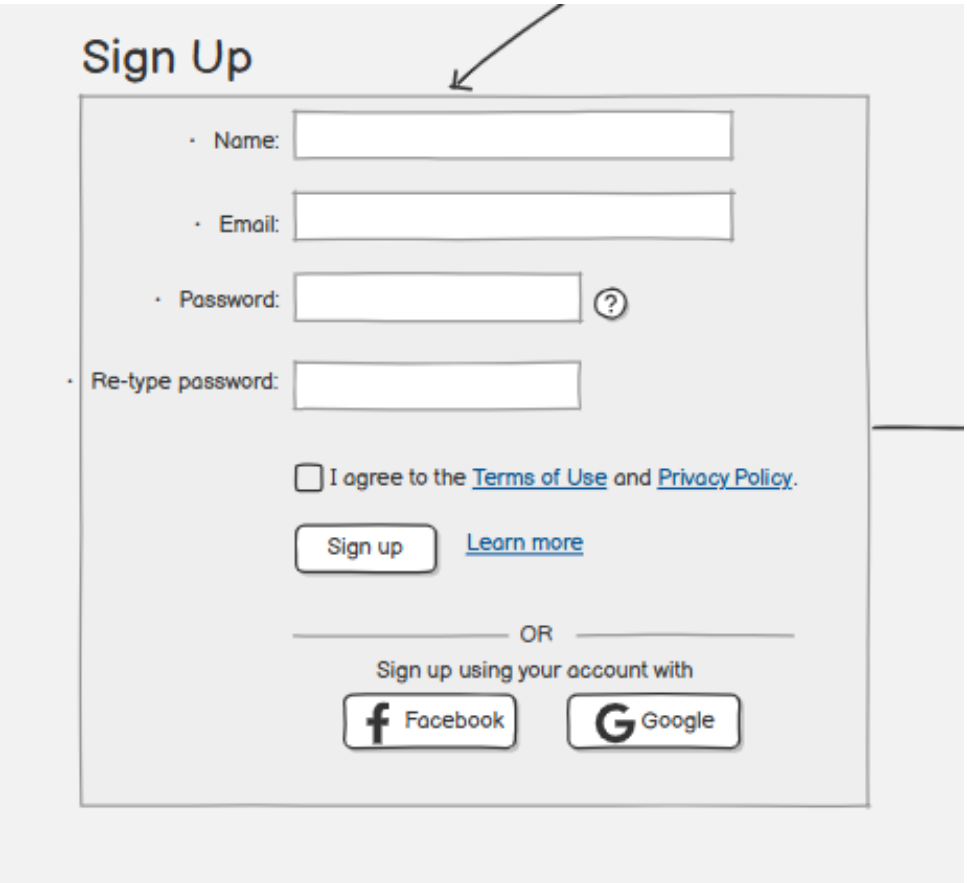

Figure 4.14: Registration Screen

The Registration screen enables new users to create an account on the platform. It includes a structured form collecting essential information such as username, email address, and password. Validation indicators are used to guide the user during data entry. Upon successful registration, the user is redirected to the login process. This screen ensures a smooth onboarding experience while maintaining data consistency and security requirements.

4.5.3 Main Dashboard

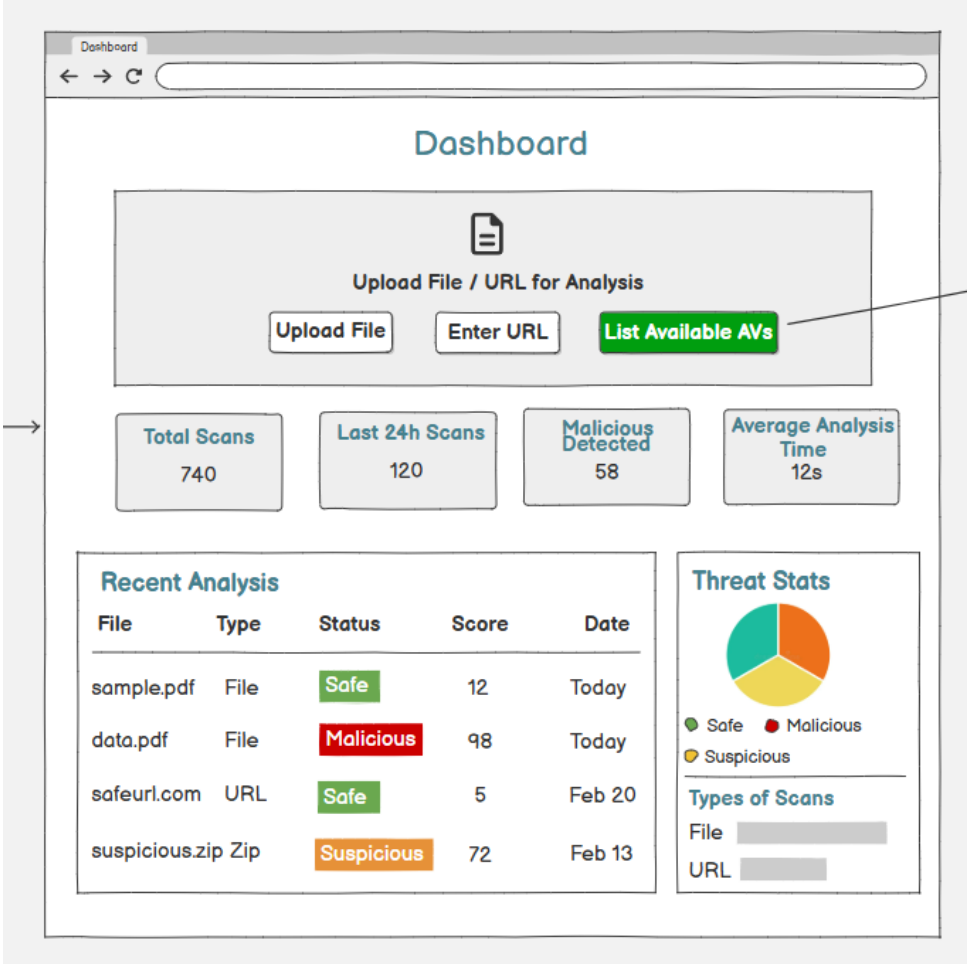

Figure 4.15: Main Dashboard

The Dashboard represents the central control panel of the system after successful login. It provides an overview of system status, recent scan activities, and engine statistics. Key information is displayed using summary cards, charts, and status indicators to allow users to quickly understand the current security state. Navigation elements are placed at the top or side to provide quick access to core system features such as scan management, engine configuration, and reports.

4.5.4 Available Antivirus Engines

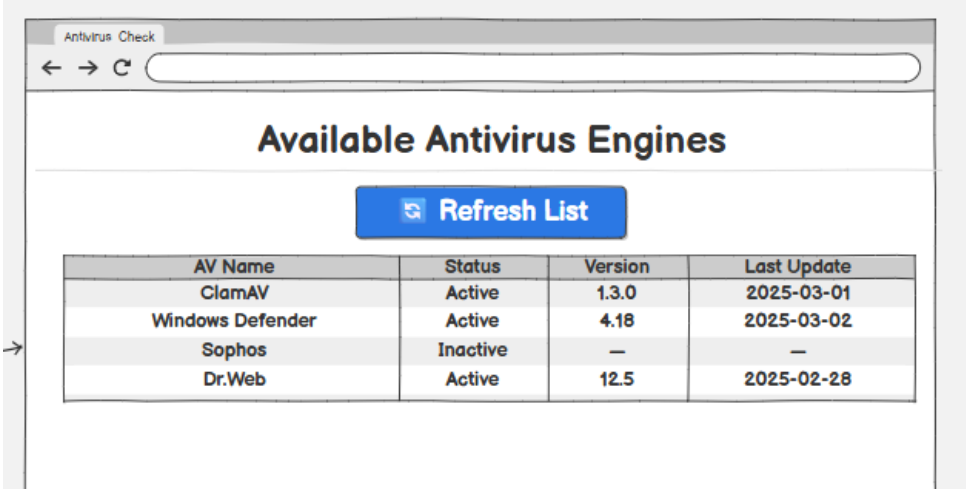

Figure 4.16: Available Antivirus Engines

This screen displays a list of all available antivirus engines supported by the system. Each engine is presented with its current status, version, and last update time. Action buttons allow the user to enable, disable, or configure individual engines. The wireframe highlights modularity by allowing multiple engines to operate independently within the same system.

4.5.5 Update Antivirus Database

This screen provides functionality for updating antivirus signature databases. It displays update progress, current database version, and update history. A progress bar visually communicates update status, ensuring transparency during long-running operations. This screen is essential for maintaining detection accuracy and system reliability.

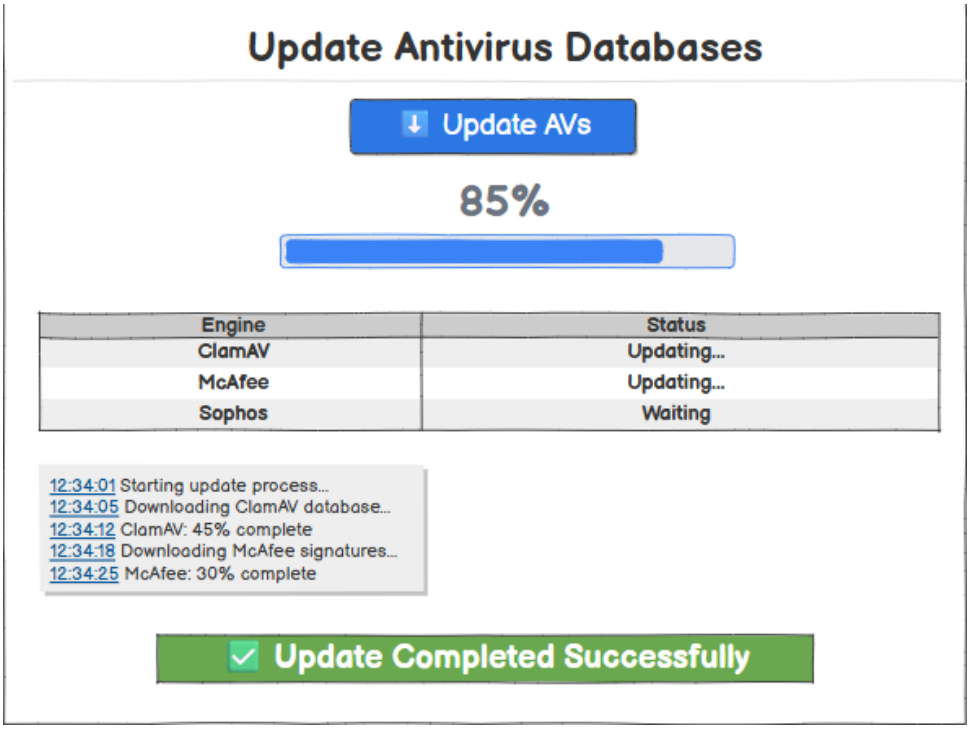

Figure 4.17: Update Antivirus Database

4.5.6 Add & Manage AV Engines

The Admin Dashboard page allows administrators to add and manage antivirus engines. The Add New AV Engine section includes input fields for the engine name, Docker image URL, configuration parameters, and an optional API key, with a prominent "Add AV Engine" button to submit the information. A confirmation message appears upon successful addition. The Manage AV Engines section displays a table of installed antivirus engines with columns for engine name, status, version, last update, and an action button to remove any engine. This interface provides a clear and centralized way to configure and maintain multiple antivirus engines.

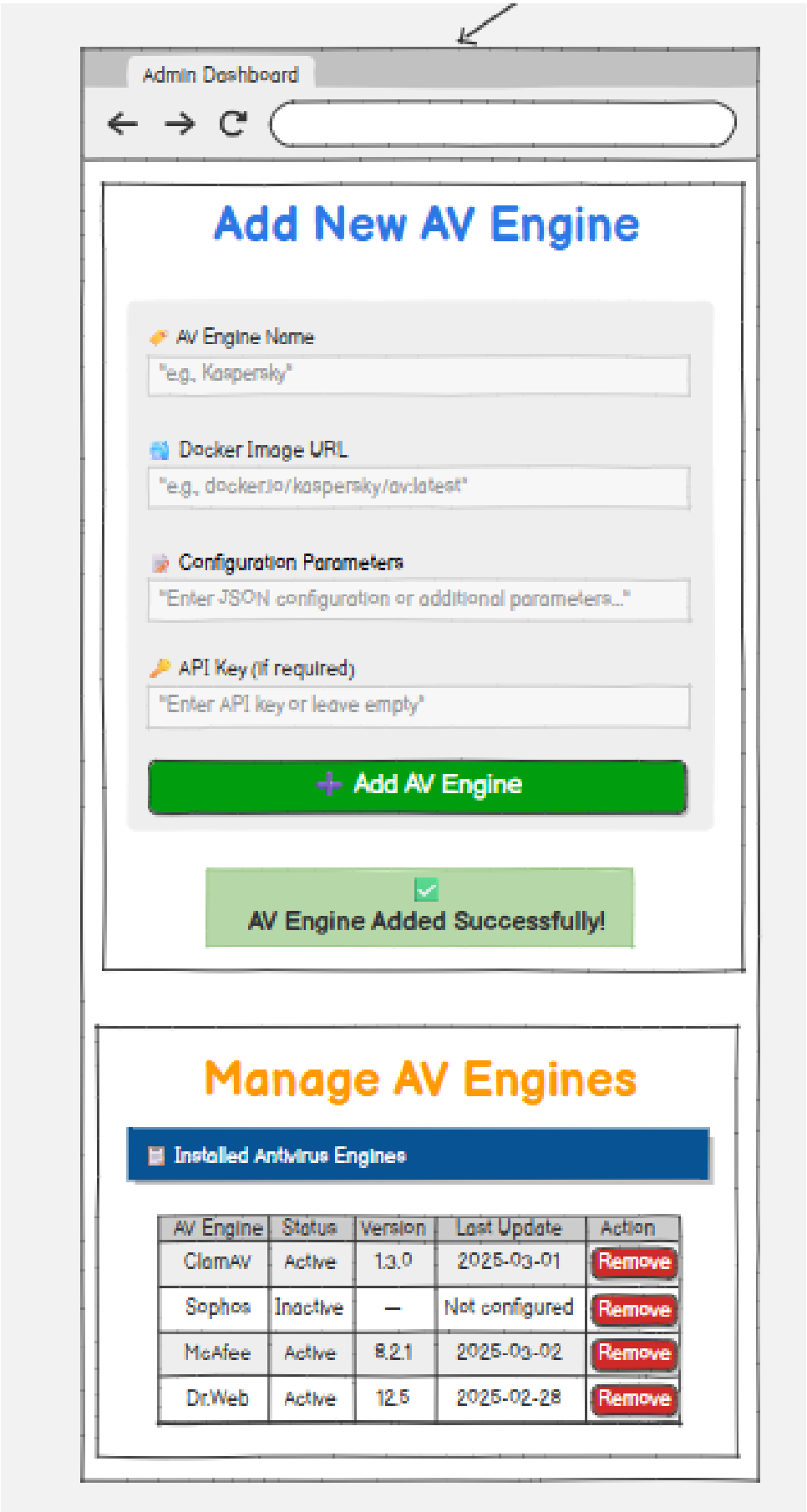

Figure 4.18: Add & Manage AV Engines

4.5.7 File Upload & Scan Request

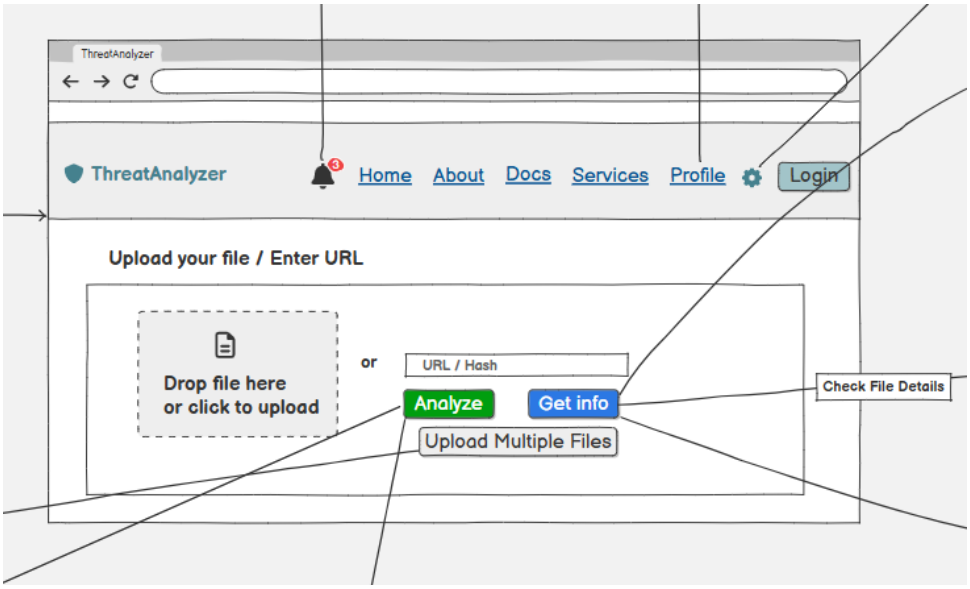

Figure 4.19: File Upload & Scan Request

This screen is the primary interface for file security analysis. It includes a drag-and-drop zone for file uploads, with options to browse files, enter a URL, or submit a file hash. Users can upload multiple files for batch processing. Key buttons include "Analyze" to start scanning, "Get Info" for metadata, and "Check File Details." The top navigation bar provides links to Home, About, Docs, Services, and Profile, along with Login and notifications. The design focuses on flexibility and a clean, user-friendly experience.

4.5.8 Multiple File Scan

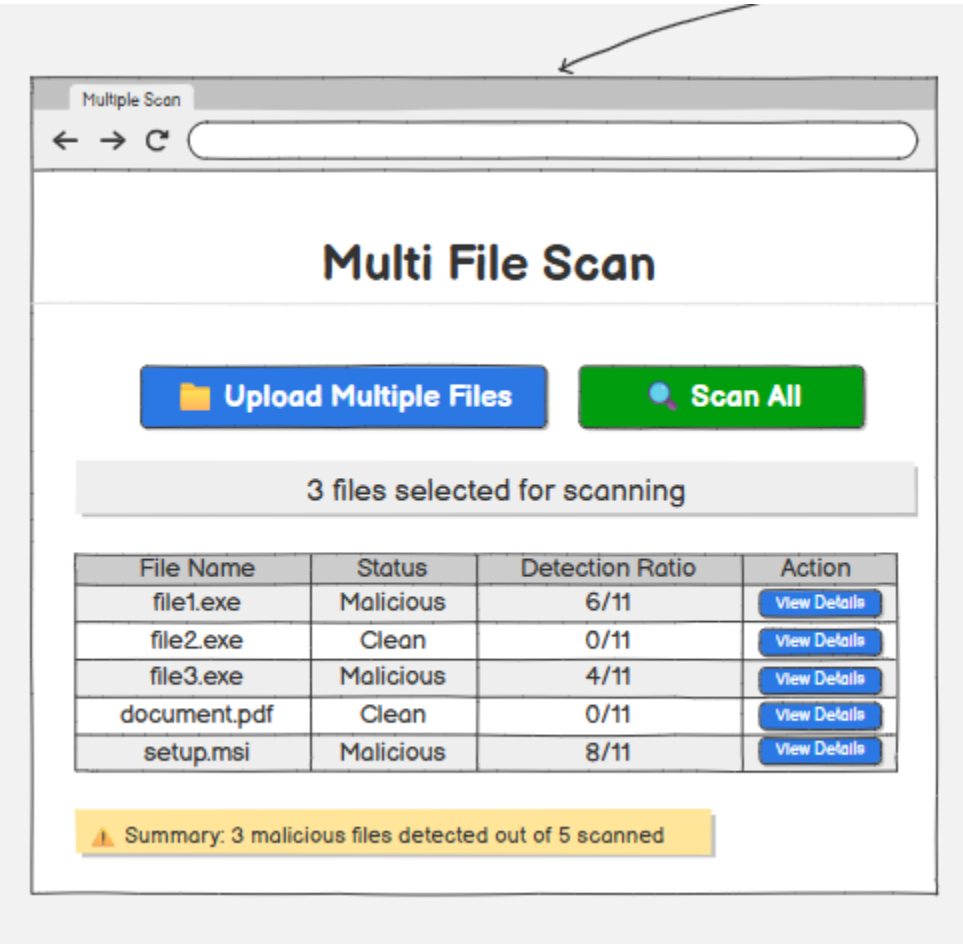

Figure 4.20: Multiple File Scan

This screen displays the results of scanning multiple uploaded files at once. The table lists each file by name, overall status (Malicious or Clean), detection ratio (number of engines that flagged it as a threat), and a "View Details" button for further information. A summary at the bottom indicates the total number of malicious files detected. Buttons at the top allow uploading additional files and starting the scan process. The layout provides a concise tabular overview to facilitate quick assessment of batch scan outcomes.

4.5.9 Scan Results Overview

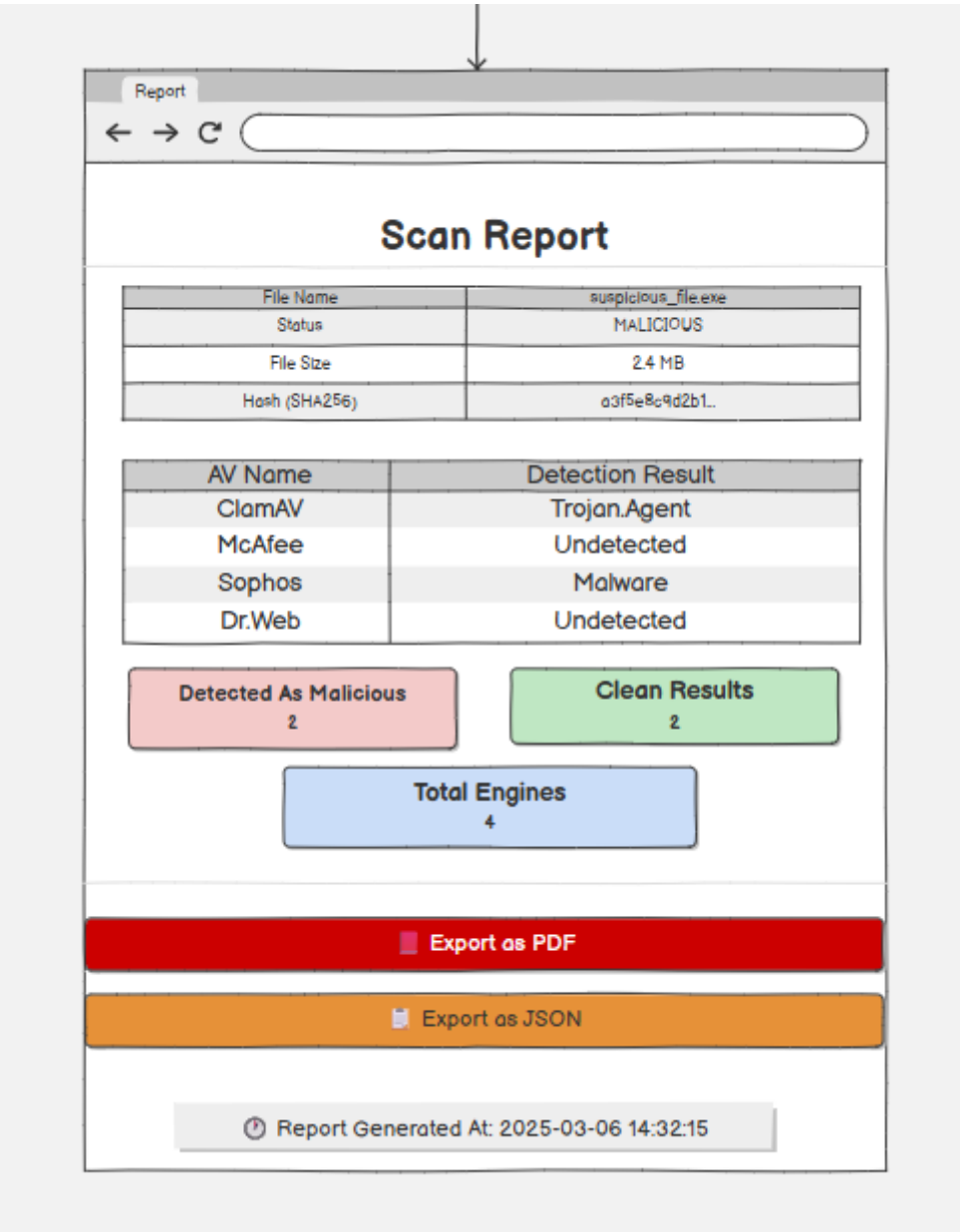

Figure 4.21: Scan Results Overview

This screen shows a comprehensive scan report for the analyzed file. It displays file details including name, malicious status, size, and hash value. A table lists results from each antivirus engine with specific detection outcomes. Summary sections highlight the number of engines detecting the file as malicious and clean, along with the total engines used. Export buttons allow downloading the report as PDF or JSON. A timestamp at the bottom records the report generation date and time. The design offers a structured and clear view for thorough threat evaluation.

4.5.10 Detailed File Information

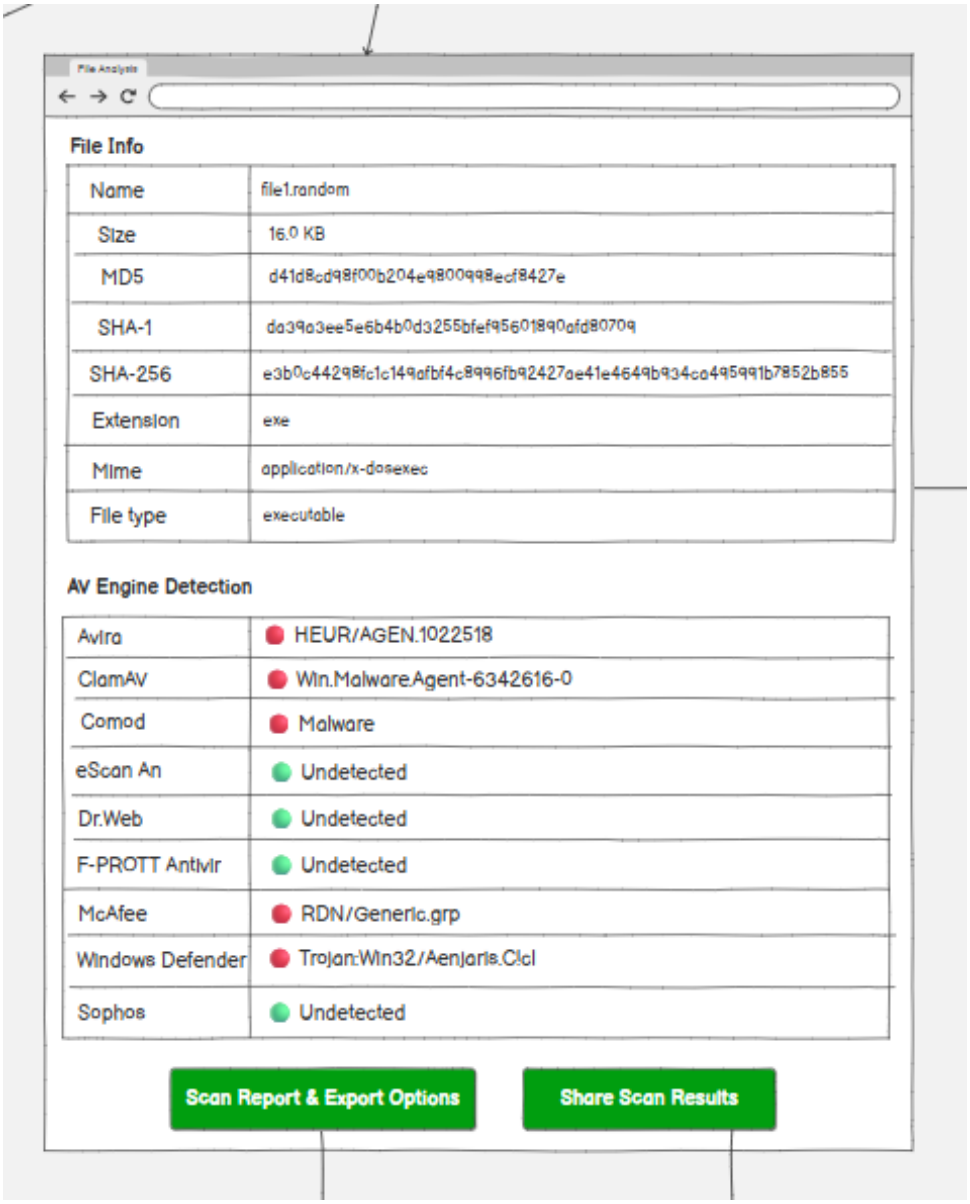

Figure 4.22: Detailed File Information

This screen shows detailed information for the scanned file, including name, size, MD5, SHA-1, SHA-256 hashes, extension, MIME type, and file type. Below, it lists detection results from various antivirus engines with color-coded indicators for threats like trojans, malware, and generic detections, or clean verdicts. Buttons at the bottom provide options for scan report export and sharing results. The layout offers clear metadata and per-engine insights for effective threat analysis.

4.5.11 Scan Report Sharing

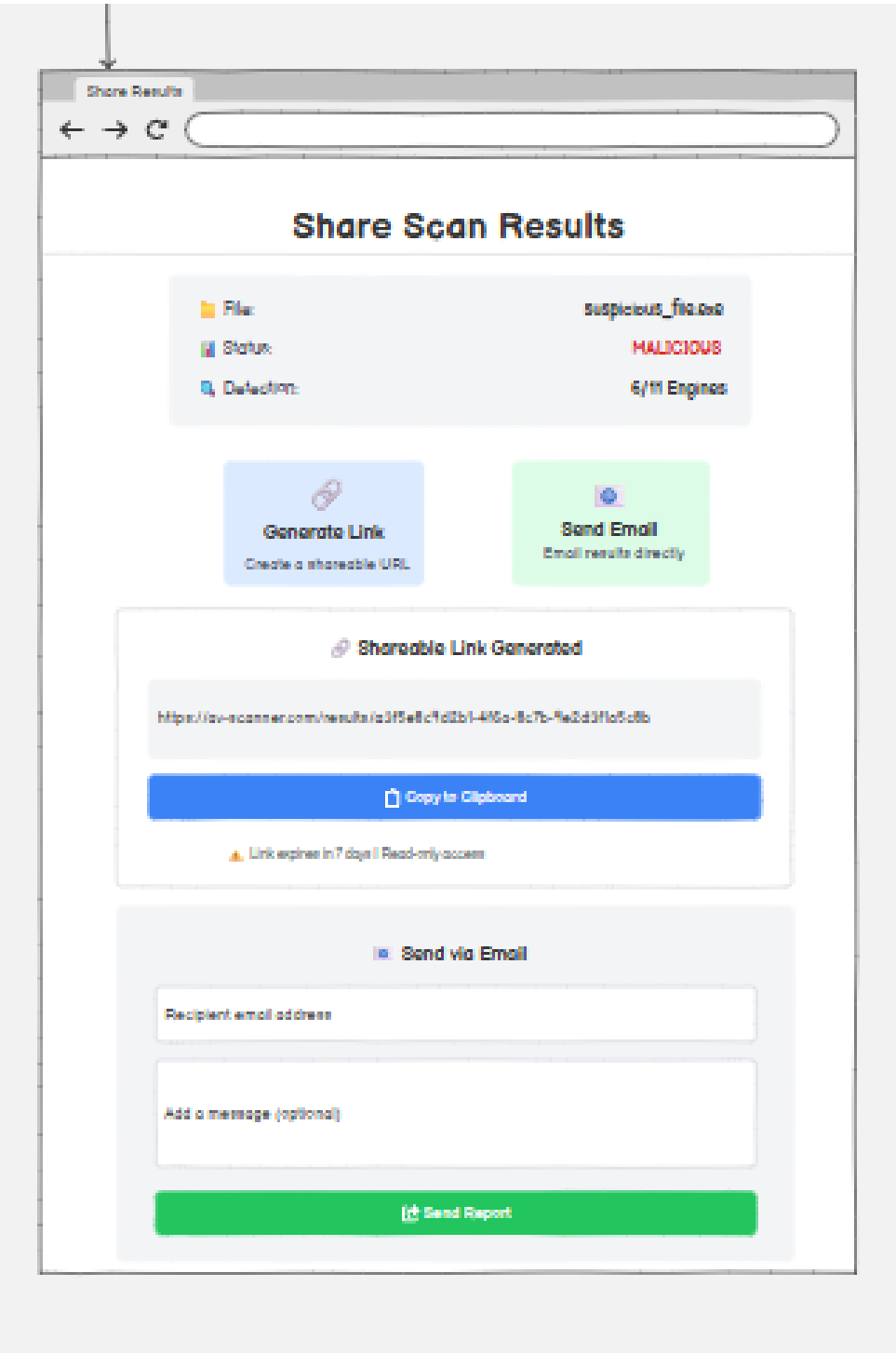

Figure 4.23: Scan Report Sharing

This screen allows users to share the scan results of a malicious file. It displays file details including name, status, and detection ratio. Options include generating a shareable link with a pre-filled URL and copy button, or sending results directly via email with fields for recipient addresses and an optional message. The layout provides secure and convenient

sharing methods for collaboration or reporting.

4.5.12 Scan URL

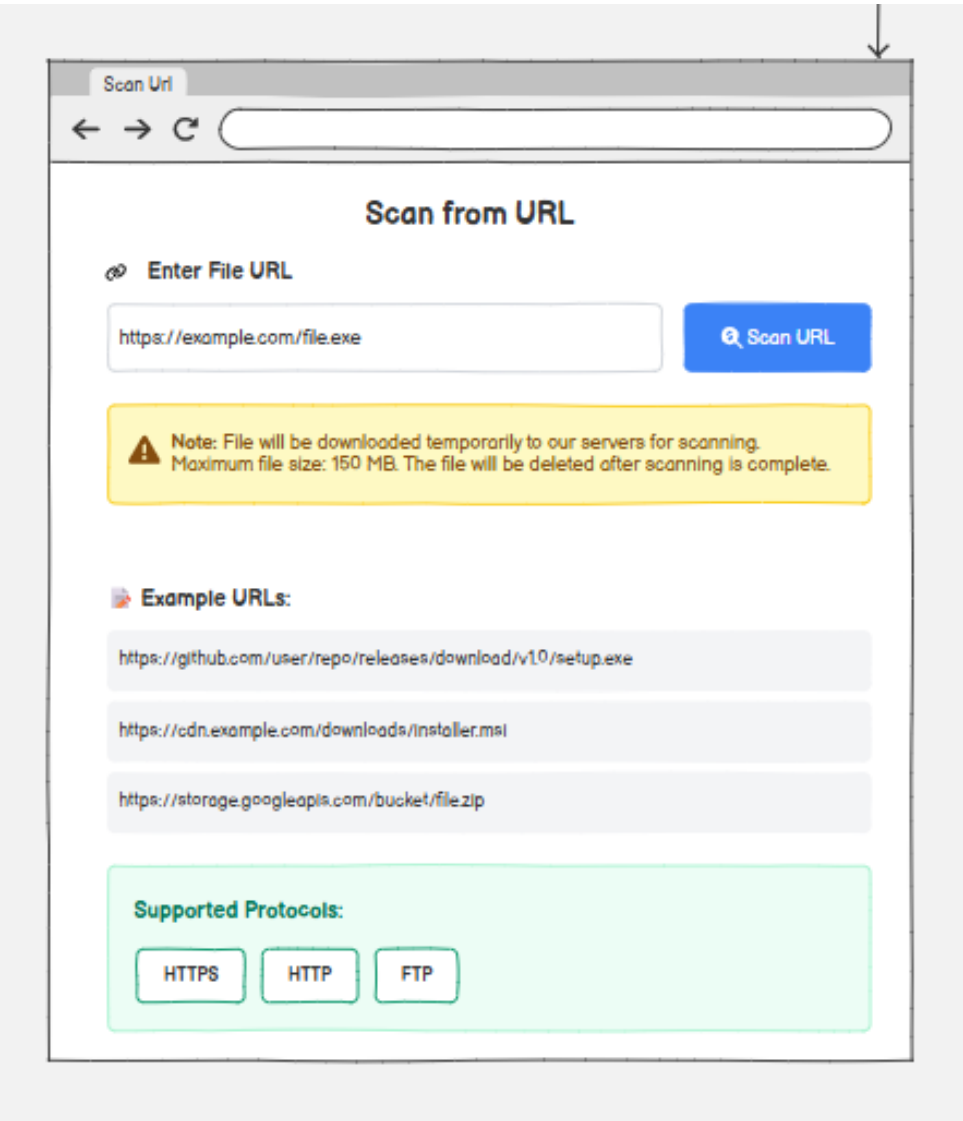

Figure 4.24: Scan URL Interface

The URL Scan Interface allows users to submit web addresses for security analysis. It features a simple input field for entering the URL and a primary scan button to initiate analysis. A warning message informs users that files will be temporarily downloaded with a maximum size of 100 MB. Example URLs demonstrate proper format, and supported protocols (HTTPS, HTTP, FTP) are clearly displayed. This screen emphasizes simplicity and ease of use for quick URL verification.

4.5.13 User Profile

This screen displays the user's profile information on the left sidebar, including name, email address, and key statistics such as total scans performed, threats found, and clean files processed. Navigation menu options include Account Settings, View Dashboard, Scan History, Notifications, and a prominent Logout button. The layout provides a quick overview of user activity and easy access to account-related features.

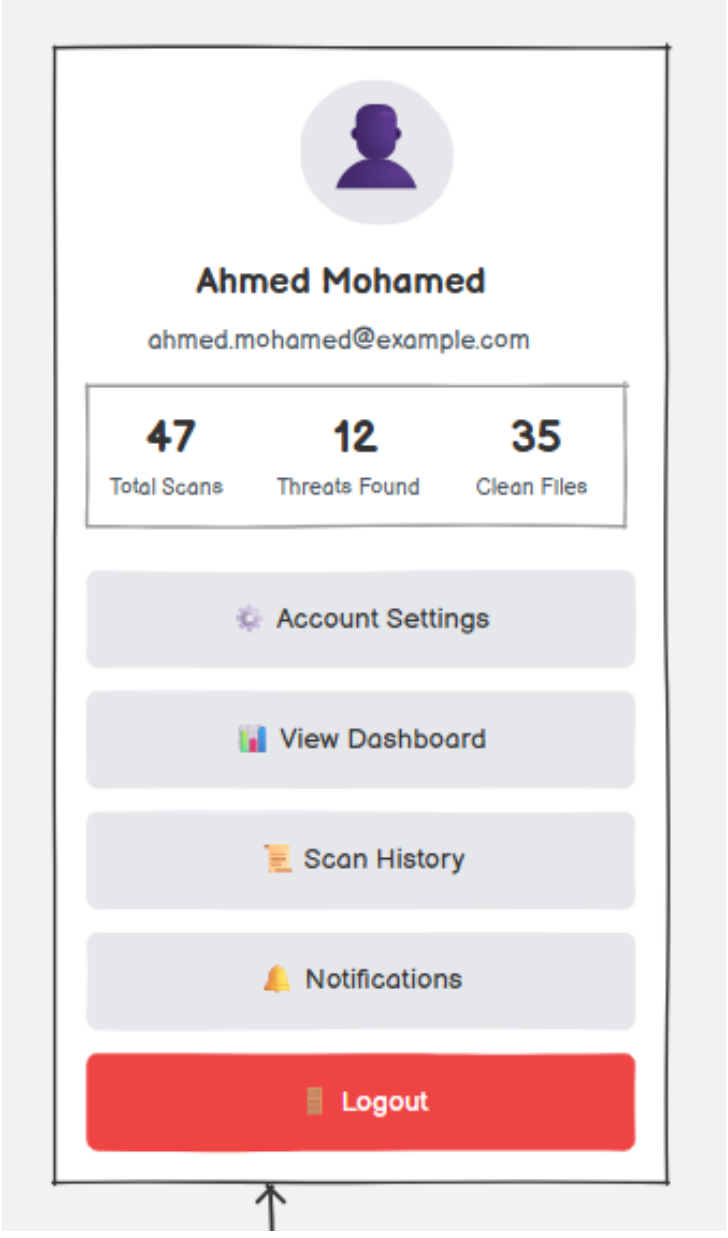

Figure 4.25: User Profile

4.5.14 System Settings

This screen allows customization of user settings. It includes sections for selecting default antivirus engines with checkboxes, choosing appearance mode (light or dark), language selection, and notification toggles for email notifications, browser alerts, and update reminders. A "Save Preferences" button applies the changes. The design focuses on personalization to enhance the user experience.

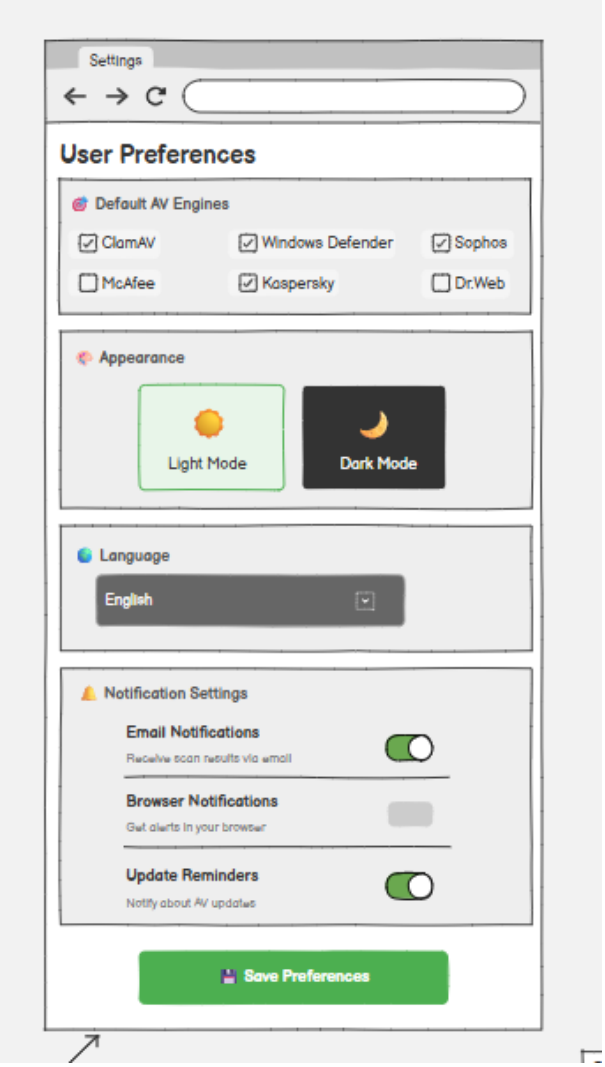

Figure 4.26: System Settings

4.5.15 Help & Support Screen

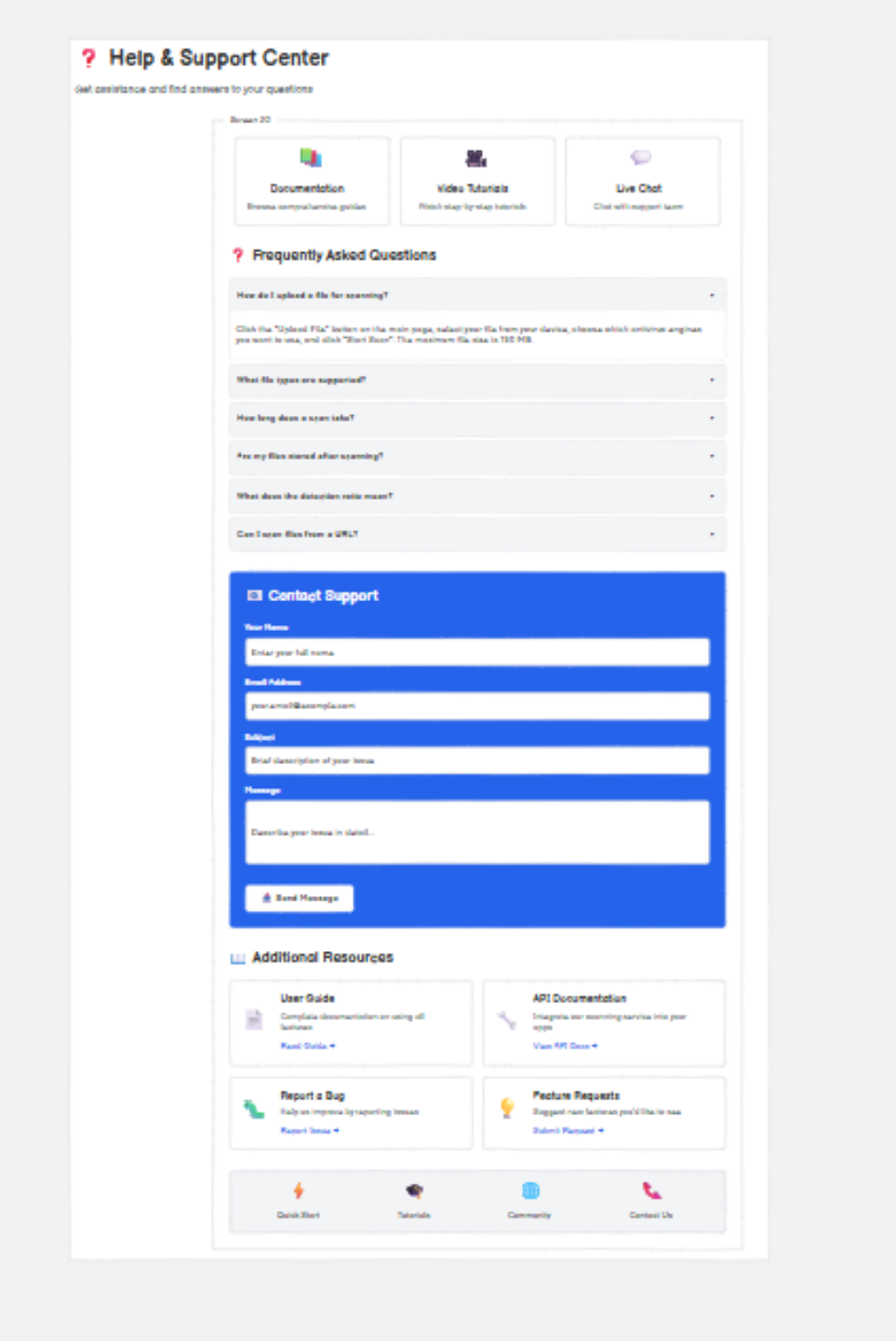

Figure 4.27: Help & Support Screen

This screen provides comprehensive user assistance. It features quick access tiles for Documentation, Video Tutorials, and Live Chat. A Frequently Asked Questions section lists common queries with expandable answers. The Contact Support form allows submission of name, email, subject, and message. Additional Resources include links to User Guide, API Documentation, Report a Bug, Feature Requests, Quick Start, News, Community,

and Contact Us. The layout centralizes support options for efficient issue resolution and self-help.

4.6 Summary

This chapter has presented the complete system design for the TIBSA platform, detailing an eight-layer architecture that delivers comprehensive threat intelligence and security analysis capabilities. The design leverages modern cloud-native technologies and open-source security tools to create a scalable, secure, and maintainable platform.

The key architectural components include:

- User and Edge Layers (1-2): Provide secure access points with CDN, WAF, DDoS protection, and bot detection through Cloudflare integration.
- Frontend Layer (3): Delivers a responsive user interface using Next.js 15, Tailwind CSS, and shadcn/ui, with robust client-side security controls and resumable file upload capabilities.
- API Engine (4): Centralizes request handling through KrakenD with Authentik providing comprehensive authentication, MFA, and RBAC functionality.
- Backend Services (5): Implements core business logic including scan coordination, threat intelligence via MISP, ML-powered detection using Ollama, dynamic malware analysis through CAPE Sandbox, and automated threat modeling with Threagile.
- Data Storage Layer (6): Utilizes Neon Serverless Postgres, Upstash Redis, and Cloudflare R2 for structured data, caching, and object storage respectively.
- Monitoring Layer (7): Provides comprehensive observability through Wazuh SIEM, Grafana, Prometheus, and Loki for security monitoring, metrics, and log aggregation.
- TIBSA Core (8): Automates the complete threat modeling workflow including DFD generation, STRIDE analysis, MITRE ATT&CK mapping, and risk scoring.

The architecture ensures security through multi-layered protection, scalability through serverless components, reliability through background job processing and retry logic, performance through caching and CDN, observability through comprehensive monitoring, and compliance through audit trails and RBAC. The user interface wireframes demonstrate a clean, intuitive design that guides users through authentication, scanning, threat analysis, and administrative functions. This modular and well-documented design provides a solid foundation for implementing, testing, and extending the TIBSA platform.

Chapter 5

Conclusion and Future Work

5.1 Summary

This project proposes a **unified, intelligence-driven security assessment framework** that bridges predictive analysis and practical validation. By integrating **threat modeling**, **penetration testing**, and **threat intelligence**, the approach empowers organizations to **prioritize real risks**, **respond faster**, and **continuously strengthen** their defenses in an ever-evolving threat landscape.

The eight-layer TIBSA platform architecture demonstrates how modern security tools and methodologies can be integrated to provide comprehensive, automated, and continuous security assessment capabilities.

5.2 Contributions

The main contributions of this project include:

1. A comprehensive framework integrating threat modeling with penetration testing
2. An eight-layer architecture design for the TIBSA platform
3. Integration of 18 different security tools and services
4. Automated threat-to-test mapping methodology
5. Continuous feedback loops for security improvement

5.3 Future Work

Potential areas for future enhancement include:

- Development of the web-based analytical tools dashboard

- Implementation of advanced AI/ML models for threat prediction
- Integration with additional threat intelligence feeds
- Enhanced automation of penetration testing workflows
- Mobile application development for security monitoring

List of References

[1] D. S. Xu, B. S. Chaparro, X. Ou, and P. Rao, "Automated Security Test Generation with Formal Threat Models," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2012.

[2] B. Marback, C. Doerr, P. Rao, B. S. Chaparro, and X. Ou, "A Threat Model-Based Approach to Security Testing," *Software: Practice and Experience*, 2013.

[3] R. Palanivel and A. P. Selvadurai, "Risk-Driven Security Testing Using Risk Analysis with Threat Modeling Approach," *SpringerPlus*, 2014.

[4] F. A. Rahim, N. Jamil, Z. C. Cob, L. M. Sidek, and N. I. I. Sharizan, "Risk Analysis of Water Grid Systems Using Threat Modeling," *Journal of Physics: Conference Series*, vol. 2261, no. 1, p. 012015, 2022.

[5] C. E. Alozie, "Threat Modeling in Health Care Sector," *International Journal of Engineering and Modern Technology (IJEMT)*, vol. 10, no. 11, pp. 199–205, 2024.

[6] D. Granata and M. Rak, "Systematic analysis of automated threat modelling techniques: Comparison of open-source tools," *Software Quality Journal*, vol. 32, pp. 125–161, 2024.

[7] B. Shin, A. Elkins, L. Larson, M. Perez, and L. Cameron, "Actionable Intelligence-Oriented Cyber Threat Modeling Framework," in *Proc. 55th Hawaii Int. Conf. on System Sciences (HICSS)*, pp. 6782–6791, 2022.

[8] M. Dekker and L. Alevizos, "A Threat-Intelligence Driven Methodology to Incorporate Uncertainty in Cyber Risk Analysis and Enhance Decision Making," *Computers & Security*, 2023.

[9] J. Smith, R. Johnson, and L. Williams, "Performing Ransomware Detection through Predictive Behavioral Mapping to Autonomous Threat Identification," *IEEE Access*, vol. 10, pp. 14532–14548, 2022.

[10] B. Bin Sarhan and N. Altwaijry, "Insider Threat Detection Using Machine Learning Approach," *Applied Sciences*, vol. 13, no. 259, pp. 1–19, 2023.

[11] M. Alharbi, E. Al-Shaer, and N. Almakhdhub, "Actionable Intelligence-Oriented Cyber Threat Modeling Framework (TIME)," in *Proceedings of the 55th Hawaii International Conference on System Sciences (HICSS)*, pp. 6782–6791, 2022.

[12] G. Chu, "Automation of Penetration Testing," Ph.D. dissertation, Univ. of Liverpool, Liverpool, U.K., Sept. 2021.

[13] J. Huang and Q. Zhu, "PenHeal: A Two-Stage LLM Framework for Automated Pentesting and Optimal Remediation," *arXiv preprint*, vol. abs/2407.17788, 2024.

[14] K. Chauhan, "Insider Threats Mitigation: Role of Penetration Testing," Department of Computer Science, University of Guelph, Guelph, Canada, 2024.

[15] D. P. R. Sanagana, "Mitigating Network Threats: Integrating Threat Modeling in Next-Generation Firewall Architecture," *Journal of Science Technology and Research (JSTAR)*, vol. 4, no. 1, pp. 202–209, 2023.

[16] S. P. Maniraj, C. S. Ranganathan, and S. Sekar, "Securing web applications with OWASP ZAP for comprehensive security testing," *Int. J. Adv. Sig. Img. Sci.*, vol. 10, no. 2, pp. 12–23, 2024.

[17] Nmap, "Network Mapper," Wikipedia. [Online]. Available: `https://en.wikipedia.org/wiki/Nmap`

[18] Metasploit Framework, Wikipedia. [Online]. Available: `https://en.wikipedia.org/wiki/Metasploit`

[19] OWASP ZAP, OWASP Foundation. [Online]. Available: `https://www.zaproxy.org/`

[20] Burp Suite, Wikipedia. [Online]. Available: `https://en.wikipedia.org/wiki/Burp_Suite`

[21] SQLmap, Wikipedia. [Online]. Available: `https://en.wikipedia.org/wiki/Sqlmap`

[22] Tenable, "Nessus Vulnerability Scanner." [Online]. Available: `https://www.tenable.com/products/nessus`

[23] Microsoft, "Threat Modeling Tool," Microsoft Learn. [Online]. Available: `https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool`

[24] OWASP Threat Dragon, OWASP Foundation. [Online]. Available: `https://owasp.org/www-project-threat-dragon/`

[25] MITRE Corporation, "MITRE ATT&CK Framework." [Online]. Available: https://attack.mitre.org/

[26] MITRE, "CAPEC – Common Attack Pattern Enumeration and Classification." [Online]. Available: https://capec.mitre.org/about/documents.html

[27] MISP Project, "Malware Information Sharing Platform & Threat Sharing." [Online]. Available: https://www.misp-project.org/