ChatGPT

Design and Implementation of Machine Learning for Network Security

- **Student ID:** 21/03/03/108
- **Supervisor:** Egr. Aishatu Ibrahim Birma

Chapter One: Introduction

Network security is critical in today's highly connected environment, as the proliferation of devices and internet usage has led to a rise in cyber threats such as malware, phishing, and DDoS attacks [1]. Traditional defenses (firewalls, antivirus, etc.) often rely on static signatures and thus struggle to detect novel or polymorphic threats [2] [3]. In response, machine learning (ML) methods have been adopted in cybersecurity to identify anomalous patterns in data and adapt to evolving attacks [2] [4]. For example, anomaly detection algorithms can spot unusual network traffic, and classification models can learn features of malware files. This project combines ML with threat intelligence APIs to build an accessible security platform for under-resourced settings (Maiduguri, Nigeria). In particular, the platform leverages the VirusTotal API—which aggregates results from over 70 antivirus engines and URL scanners—to analyze files, URLs, and IPs in real time [5] [6]. By integrating VirusTotal with ML risk scoring and anomaly detection, the system provides users with faster, more accurate threat insights than signature-based tools alone [7] [4].

The primary objectives of this work are: (1) to design a network security system that integrates the VirusTotal API with customized ML models for threat analysis, (2) to implement the platform using Python and Streamlit, and (3) to evaluate its performance. These goals involve creating a user-friendly interface for uploading files and URLs, securing the system with robust authentication, and visualizing threat data through interactive dashboards [8] [9]. Addressing cybersecurity in Maiduguri and similar regions is significant because many local organizations lack sophisticated defenses and expertise [10] [11]. An accessible, API-driven solution (built on open-source tools) can help bridge this gap, offering scalable threat detection to improve the digital safety of under-resourced users [12] [7]. The scope of the study focuses on the design and implementation of an ML-based platform with real-time VirusTotal integration, tailored for environments with limited resources and technical expertise [11] [7].

Chapter Two: Literature Review

Network Security Fundamentals: Network security aims to protect the confidentiality, integrity, and availability of data and systems. Common measures include firewalls (traffic filtering), intrusion detection/prevention (IDPS), and encryption [13]. Modern practice emphasizes *defense-in-depth*: multiple overlapping layers protect against sophisticated threats. For instance, firewalls may be combined with anomaly-based IDPS that monitor traffic patterns for deviations (zero-day or advanced threats) [14] [15]. However, each layer has limits. For example, traditional antivirus and firewalls rely on known threat signatures or rules, making them ineffective against novel attacks [16] [14].

Machine Learning in Security: ML techniques address some of these limits by learning from data. Supervised models (e.g. SVMs, random forests, neural nets) have been trained on labeled benign vs. malicious samples to detect malware or intrusion [17] [4]. Unsupervised methods (e.g. clustering, anomaly detection) identify outliers without needing labels, useful for spotting unusual network behavior [18] [19]. Researchers have shown ML can detect new threats and adapt over time [19] [20]. For example, ML has been used for endpoint protection, analyzing device telemetry to flag compromises [21]. However, ML models face challenges (data quality, training overhead, adversarial evasion) [19], so practical systems often combine ML with other tools.

Threat Types: Common cyber threats include **malware** (viruses, ransomware, etc.), which can steal data or disrupt operations (particularly harmful in low-resource settings) [22] [7]; **phishing**, which exploits human weakness to gain credentials [23]; **DDoS attacks**, which flood networks to cause downtime [24]; and **APTs** (advanced persistent threats) that stealthily compromise sensitive systems [25]. These threats impact organizations severely, especially where defensive infrastructure is weak. A layered approach – combining traditional tools with ML analysis and threat intelligence – is recommended to mitigate such varied threats [26] [27].

Threat Intelligence APIs (VirusTotal, etc.): Online multi-scanner platforms like VirusTotal emerged to overcome limitations of single-engine antivirus [4] [6]. VirusTotal aggregates scan results from numerous antivirus engines and URL/domain blacklists, giving a comprehensive "consensus" verdict on submitted files, URLs, or IP addresses [4] [6]. Its API enables automated querying: files and URLs can be uploaded and then repeatedly polled until a complete multi-engine analysis report is ready [28] [29]. In practice, VirusTotal provides immediate, detailed insights (e.g. number of antivirus detections, malicious domains in a URL's history) that greatly enhance threat detection [28] [4]. Security systems use these APIs to supplement their own detection. For example, the platform developed here sends user-submitted files/URLs to VirusTotal and retrieves multi-engine analysis results (malicious/suspicious counts, detailed engine verdicts) for further processing [30] [28].

However, reliance on external APIs brings constraints. The VirusTotal API imposes strict rate limits on free-tier usage, which can delay high-volume scanning if not managed properly [27] [31]. Also, by design VirusTotal can only flag what its engines know; brand-new, undetected threats may slip through. Therefore, many systems combine VirusTotal data with additional ML analysis. For instance, after retrieving VirusTotal results, our platform computes a **risk score** (a 0–100 scale based on counts of malicious/suspicious detections) and checks for file anomalies (e.g. suspicious extensions or sizes) to catch malware traits not directly captured by signatures [32] [33]. These ML-derived insights complement the raw VirusTotal output and help quantify threat severity beyond simple signature counts.

Related Platforms: Previous work has built streamlined UIs for security analysis using frameworks like Streamlit. Streamlit enables rapid development of interactive data apps with minimal frontend coding [34] [4]. Its simplicity is well-suited for providing clear dashboards of security scans. Other systems (e.g. [source withheld]) similarly integrate threat intel APIs with visualization libraries (like Plotly) to present scan results graphically [35] [4]. The literature shows that combining APIs and ML improves threat detection: for example, specialized SIEM tools often embed VirusTotal queries and ML analytics for richer alerts [36]. This project draws on these ideas to create a user-friendly, intelligent security platform tailored to resource-limited contexts.

Chapter Three: Methodology

3.1 System Analysis and Design

We first analyzed existing solutions and their shortcomings in our target context. In Maiduguri and similar regions, many systems still rely on signature-based detection, which fails against novel threats [37]. They often lack integration with broad threat databases like VirusTotal, limiting their visibility into new malware and malicious domains [37]. High-end security tools also tend to be cost-prohibitive and require skilled admins, which is challenging for under-resourced organizations [38]. These gaps motivated our new design.

The proposed platform overcomes these issues by integrating the VirusTotal API with ML in a streamlined, affordable solution. Key design decisions include: - **Advanced Threat Detection:** By using VirusTotal's multi-engine scans along with ML algorithms, the system can detect both known and emerging threats more robustly than signature-only tools [7] [32]. - **Cost-Effectiveness:** We build on open-source tools (Python, Streamlit, SQLite, etc.) to minimize costs and ease deployment in constrained settings. - **Real-Time Analysis:** The system is designed for asynchronous, real-time scanning. Uploaded files or URLs are immediately sent to VirusTotal, and results are polled until completion to give prompt feedback [20] [39]. - **User-Centric Interface:** A Streamlit front-end provides an intuitive UI requiring little technical expertise [34]. - **Scalability:** The architecture is modular, so additional analyses (e.g. more ML models or threat sources) can be integrated later with minimal changes.

Figure references (not shown here) illustrate the system's Data Flow Diagrams (DFDs) and UML diagrams. For example, a Level-0 (context) DFD shows interactions between the **User**, the **VirusTotal API**, and our platform. A Level-1 DFD breaks this down into main processes (file/URL submission, analysis, result storage). Likewise, a UML use-case diagram highlights user actions (login, scan file, scan URL, view reports) and a class diagram models core entities (User, ScanRecord, ThreatReport, etc.) [40] [41]. These designs guided the implementation structure. The database design includes tables for **users** (id, username, email, password hash, role) and **scans** (scan id, type, target, hashes, VirusTotal analysis id, results summary, timestamps) [42] [43]. An ER diagram (not shown) captures relationships between users and their scan results.

3.2 System Architecture

The platform follows a multi-tier architecture. The **frontend** is built in Streamlit, a Python framework for interactive apps. It provides pages for user registration/login, a dashboard, file scanning, and URL scanning [34] [8]. For example, the dashboard page dynamically displays recent scans, threat distribution charts, and personalized security insights. Interactive visualizations (using Plotly) show trends such as count of detected threats over time or breakdown of malicious vs. clean scans [35] [44].

The **backend** is implemented in Python and handles business logic. Core modules include: - **Authentication (utils/auth_utils.py):** Manages user sessions and access control. User passwords are hashed using bcrypt before storing [45], and Streamlit's `st.session_state` is used to track login state and enforce page-level authentication. - **Database Utilities (utils/db_utils.py):** Interfaces with a SQLite database (`app.db`) to store users and scan records. On each new scan request, the backend creates a pending record linked to the user. - **VirusTotal API Integration (utils/virustotal_api.py):** Encapsulates all communication with VirusTotal. Functions like `upload_file_to_virustotal` and `scan_url_with_virustotal` send data to the appropriate VT API endpoints [46] [47]. For example, uploading a file is done via a `requests.post` to the `/files` endpoint with the file as multipart data [47].

Because VT scans are asynchronous, the backend polls the `/analyses/{id}` endpoint until the analysis is complete, then retrieves the report. - **Machine Learning (utils/ml_utils.py):** Processes VirusTotal results to derive deeper insights. It computes a **risk score** (0–100) based on the number of malicious/suspicious detections [32]. It also performs **file anomaly detection** by examining file metadata (e.g. double extensions, unusually small executables) that often indicate malicious intent [33]. Historical scan data can be analyzed to detect **threat patterns** (common attack vectors) and generate **security recommendations** for users [48]. - **Visualization (utils/visualization_utils.py):** Prepares data for charts. It creates Plotly figures (bar charts of threat types, timelines of scans, risk gauge dials, etc.) that the frontend embeds directly into the Streamlit pages [35].

All data (user accounts, scan history, ML results) is stored in a local **SQLite** database for persistence [49]. The schema is simple: a `users` table (id, username, email, password hash, role) and a `scans` table (id, user_id, scan_type, target, file_hash, analysis_id, status, malicious_count, suspicious_count, clean_count, result_summary, timestamp). This enables quick data retrieval for the user dashboard and ensures user data privacy.

Overall, the architecture enables a smooth data flow: when a user submits a file/URL, the backend creates a scan record, sends it to VirusTotal, waits for the report, then runs the ML analysis and updates the record [20] [39]. Finally, all results (raw VT scan data plus ML-derived risk score and recommendations) are sent back to the frontend to display to the user. Meanwhile, the dashboard aggregates historical data from the database to show trends. This layered design (Streamlit UI → Python backend → SQLite storage with VT/ML integration) provides scalability and clear separation of concerns [50] [51].

Chapter Four: Implementation and Results

4.1 Development Environment and Tools

The system was implemented entirely in **Python**. Key dependencies (listed in `requirements.txt`) include **streamlit** (for the web UI), **requests** (for API calls), **python-dotenv** (for managing environment variables like `VIRUSTOTAL_API_KEY`), and **bcrypt** (for password hashing) [52]. The Python 3.11 environment and these libraries are sufficient to run the application on a standard development machine. Streamlit abstracts away most front-end work, allowing rapid creation of forms and charts with simple Python code.

For example, setting up a file upload widget in Streamlit required only a few lines of code. When a user uploads a file, the backend reads it in binary mode and calls a function like:

```
analysis = upload_file_to_virustotal(uploaded_file.read())
```

which posts to VirusTotal's `/files` endpoint (see code snippet) [47]. The app then enters a polling loop (`wait_for_analysis_completion`) until the VT analysis is done. Similar code handles URL scans. All VT API communications include the secret API key in the header (secured server-side via an environment variable, not exposed to the client).

4.2 User Interface

The front-end is delivered as a single Streamlit app (`app.py`) with multiple pages. The **Login/Register** page uses the `auth_utils.py` module: passwords entered by users are immediately hashed with bcrypt [45] and stored in SQLite. A `require_auth` decorator (using `st.session_state`) guards the main pages so only authenticated users can access scanning functions [53].

After logging in, the **Dashboard** page displays personalized security metrics: recent scan history, charts of threat distribution, and risk gauges. For instance, we implemented a horizontal bar chart (via Plotly) showing counts of detected malicious and suspicious results per user. The interface is clean and responsive, making advanced features accessible to non-technical users [34] [44].

The **Scan File** page lets users upload a file for analysis. Upon submission, the file is immediately sent to VirusTotal (with a "pending" status shown to the user). Once VirusTotal finishes, the app presents the results: it shows the number of antivirus engines that flagged the file, a table of engine results, and the ML **risk score** (e.g. "Risk: 85/100" if many engines found malware). The **Scan URL** page works similarly for URLs, checking domain and history. All scan results (file hash, URL, malicious count, etc.) are stored in the database so users can review past results. In summary, the Streamlit UI provides an intuitive workflow for uploading data, seeing progress, and viewing detailed results (see screenshots in Appendix, not shown here). Feedback from testers confirmed the interface is simple and actionable [54].

4.3 VirusTotal Integration

Integration with VirusTotal is a core feature. The `virustotal_api.py` module contains functions like `upload_file_to_virustotal(file_content)` and `scan_url_with_virustotal(url)`, which wrap the HTTP calls. For example, uploading a file uses code:

```
url = f"{VIRUSTOTAL_BASE_URL}/files"
headers = {"X-Apikey": VIRUSTOTAL_API_KEY}
files = {"file": file_content}
response = requests.post(url, headers=headers, files=files)
```

This routine handles errors and returns the JSON response containing an analysis ID [47]. For URL scans, a similar approach posts to `/urls`. Because VT scans are not instantaneous, the system then polls the `/analyses/{id}` endpoint every few seconds, checking the `status` field until it reports "completed". This asynchronous handling is crucial: it allows the UI to remain responsive (Streamlit shows a loading spinner) while waiting for the scan to finish. Once the report is ready, the raw JSON (with engine results) is fetched and parsed.

After retrieving the VT report, the backend extracts key metrics (malicious count, suspicious count, file hash/URL, etc.) and updates the `scans` table. We also pass the report to the ML module for further processing.

4.4 Machine Learning Components

The platform's ML logic is implemented in `ml_utils.py`. When a VirusTotal report is obtained, the system performs several analyses:

- **Risk Scoring:** We designed a simple proprietary algorithm that assigns a numeric risk score (0–100) based on the proportion of engines flagging the target as malicious or suspicious [32]. For example, if 40 out of 70 engines detect malware, the score might be (40/70)*100 ≈ 57. This gives a quick, quantitative measure of threat severity.
- **File Anomaly Detection:** The system examines file metadata (extension, size, filename patterns) in combination with the scan results [33]. For instance, if an executable file has a double extension ("something.pdf.exe") or is suspiciously small yet flagged malicious, the module raises an anomaly flag. This helps catch clever obfuscations or new malware that might otherwise appear "clean" to some engines.
- **Threat Pattern Analysis:** The tool aggregates a user's historical scan data to detect trends: for instance, it can report if 80% of scanned URLs in the past week were phishing links. This is shown as an insight on the dashboard.
- **Security Recommendations:** Based on the risk score and patterns, the system generates text advice (e.g. "This file is high-risk; avoid execution or consult an IT administrator."). These contextual recommendations are displayed alongside scan results.

Unit tests confirmed these ML routines work as intended: functions like `calculate_risk_score`, `detect_file_anomalies`, and `generate_security_recommendations` were tested on sample inputs to ensure meaningful outputs [55].

4.5 Data Flow

The full data flow proceeds as follows:

1. **User Action:** The user logs in and navigates to "Scan File" or "Scan URL" (Streamlit pages).
2. **Input Submission:** The user selects a file or enters a URL, then clicks "Scan." Streamlit captures this input.
3. **Record Creation:** The backend creates a new "pending" scan record in the SQLite database (associating it with the user).
4. **VirusTotal Submission:** The file content or URL is sent to VirusTotal via the API functions [20].
5. **Polling:** The backend periodically polls VirusTotal for the analysis status.
6. **Report Retrieval:** Once complete, the raw VirusTotal analysis report is received.
7. **ML Analysis:** The `ml_utils.py` processes the report: risk score is calculated and anomalies checked [32] [33].
8. **Database Update:** The scan record is updated to "completed," and fields like malicious_count, suspicious_count, risk_score, and result_summary are filled.
9. **Display Results:** The processed results (including charts and recommendations) are sent to the Streamlit UI and shown to the user.
10. **Dashboard Update:** The user's dashboard is refreshed to include the new scan data (threat distribution charts, updated statistics) [56].

This pipeline ensures all steps are tracked and the user sees up-to-date information. In testing, the data flow was confirmed end-to-end: each scan triggered a database entry, VT API interaction, ML processing, and UI update without errors [57] [58].

4.6 Performance Evaluation and Challenges

The system was evaluated through unit, integration, and user testing [59] [60]. **Unit tests** validated each module (e.g. authentication hashing, API calls, ML functions) in isolation. **Integration tests** verified that file/URL scanning from the UI all the way through ML and back to the dashboard works correctly [57]. Error handling was tested by simulating API failures and invalid inputs to ensure graceful messaging.

In terms of **performance**, the main factor affecting scan time is the VirusTotal API response. On average, small files returned results in a few seconds, but large files or cold scans (file never seen by VT) could take up to 2–3 minutes due to queuing [31]. Our polling mechanism handles this variability smoothly (showing a progress indicator), but users must wait longer for big files. Dashboard loading (retrieving and plotting historical data) was fast for typical histories (hundreds of records) because SQLite queries and Plotly renderings completed in under a second [61]. CPU and memory usage of the Streamlit app remained modest, making it feasible to run on a standard desktop or small server [62].

Challenges Encountered: The asynchronous nature of VirusTotal scans required careful design: we needed to manage API rate limits and avoid blocking the UI thread. We addressed this by using non-blocking loops and spacing out requests. Rate-limiting itself is a limitation: with the free API, only a limited number of scans per minute were allowed [27]. In high-usage scenarios, this could delay scans. Additionally, since the system relies on VirusTotal data, zero-day threats not yet in VT's database might be underreported; our ML anomaly checks help mitigate this. Finally, ensuring usability across different devices was challenging, but Streamlit's responsiveness and simple UI design helped; informal user tests indicated the interface was intuitive, though users suggested adding help tips for technical terms [63].

Chapter Five: Summary, Conclusions, and Recommendations

This project has successfully designed and implemented a machine learning–powered network security platform with integrated VirusTotal threat intelligence. Using **Streamlit** and Python, we built a clean, interactive web interface that lets users submit files or URLs for analysis [4] [34]. Each submission is scanned by VirusTotal's multi-engine database, and the results are enhanced by ML algorithms that compute a **risk score** and detect anomalies [32] [33]. The platform stores scan histories and displays them in a personalized dashboard with charts and insights. Key achievements include a user-friendly interface for complex analysis, robust use of VirusTotal for comprehensive scanning, and intelligent risk assessment beyond static signatures [64] [65]. Testing confirmed the system's functionality and stability; for example, end-to-end scans reliably produced accurate outputs and the application handled expected error scenarios gracefully [60]. Overall, the platform fulfills its goal of providing an accessible cybersecurity tool that helps users in under-resourced areas detect and understand threats in real time.

Recommendations and Future Work: Although the core objectives are met, several enhancements could strengthen the platform. Notably, more advanced ML models (deep learning or predictive analytics) could improve detection of sophisticated or future threats [66]. Integrating additional threat intelligence sources (such as Passive DNS, sandbox analysis, or commercial feeds) would broaden coverage beyond what VirusTotal provides [67]. Improving scalability is also advised: for large deployments, migrating from SQLite

to a scalable database (e.g. PostgreSQL) and employing asynchronous task queues (e.g. Celery) would speed up concurrent scans [68]. User experience can be enhanced by making the UI mobile-responsive, adding tooltips or guided tutorials, and supporting multiple languages [69]. Finally, for real-world deployment, containerization (Docker/Kubernetes) and security hardening (HTTPS, firewall rules) should be implemented. These future improvements would ensure the system remains effective and adaptable in the dynamic cybersecurity landscape.

In conclusion, this project demonstrates that combining open-source development (Python, Streamlit) with widely-available threat intelligence (VirusTotal) and ML techniques can yield a powerful, scalable security platform. By focusing on usability and API-driven analysis, it provides a practical tool for proactive threat detection, particularly suited to settings with limited resources [70] [71]. The solution and lessons learned here can guide future work to extend and refine intelligent cybersecurity tools for broader impact.

References: (Selected from literature and project documentation) Sommer & Paxson (2010); Al-Janabi & Saeed (2016); Papernot et al. (2016); Streamlit (n.d.); VirusTotal (n.d.) [72] [6] [27] [4] [64].

---

[1] [2] [5] [6] [7] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [21] [22] [23] [24] [25] [26] [27] [28] [36] [37] [38] [40] [41] [42] [43] [51] [72] Design & Implementation of Machine Learning UNIMAID Baffun.docx
file://file-LcnyVgKwLUWzHSLm9C1W8y

[3] [4] [8] [9] [20] [29] [30] [31] [32] [33] [34] [35] [39] [44] [45] [46] [47] [48] [49] [50] [52] [53] [54] [55] [56] [57] [58] [59] [60] [61] [62] [63] [64] [65] [66] [67] [68] [69] [70] [71] Network_Security_Platform_A_Streamlit-Based_AI-Powered_Threat_Detection_System.pdf
file://file-6SmqiZ6vksX49KgBaDpkCD