International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 04 | April - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

CyberCore Elite: A Cloud-Based Cybersecurity Audit and Threat Intelligence Platform

Macharla Akshaya, Chava Uday Kiran, Kilugu Karthik, Bantubilli Manikanta, Maradana Akshaya

Raghu Engineering College

Abstract

Personal digital security has become harder to ignore as cyber threats grow more targeted and pervasive. Most people manage credentials for 10 to 20 online services but have no single place to assess how exposed they actually are. CyberCore Elite is a web and mobile platform that brings security auditing, credential management, and email threat detection into one interface. The system uses a Multinomial Naive Bayes classifier for spam detection, a cloud-hosted SQL backend for credential storage, and IMAP integration for live email analysis. Tested against real-world inputs, the platform reached 94.3% spam detection accuracy and produced measurable improvements in user security scores across two audit cycles.

Keywords: *cybersecurity, threat detection, Naive Bayes, credential management, cloud security, IMAP, JWT authentication.*

I. INTRODUCTION

Managing security across many online accounts is genuinely difficult. The average user in 2024 holds credentials for 15 or more platforms, and most have no consistent process for checking whether those credentials are safe. Password reuse is common. Multi-factor authentication often sits disabled on accounts people actually care about. Phishing emails keep landing in inboxes that have no real-time filtering beyond whatever the mail provider bundles in by default.

The consequences of this carelessness are well documented. The 2023 Verizon Data Breach Investigations Report found that 74% of breaches involved a human element — stolen credentials, phishing, misuse, or simple error. Yet the tools available to ordinary users remain fragmented and hard to act on. A password manager tells you nothing about your email exposure. An antivirus product does not know you reused the same password across eight different services.

CyberCore Elite was built to address this gap. Rather than adding another isolated tool to an already cluttered toolkit, the system consolidates audit, detection, and storage into one interface accessible from both a browser and an Android device. The goal was not to build something for security professionals. It was to build something that gives a non-technical user a clear, actionable picture of their security posture in under five minutes.

This paper covers the design decisions, underlying algorithms, database architecture, implementation details, and test outcomes of the CyberCore Elite platform. Section II defines the problem. Sections III through V set out the objectives, related work, and system architecture. Sections VI through XI describe the technology, algorithms, implementation, and results. Sections XII through XIV close with security analysis, conclusions, and future directions.

II. PROBLEM STATEMENT

Existing personal security tools share a structural problem: they solve one problem at a time. Password managers store credentials but do not audit usage habits. Antivirus software scans for malware but ignores behavioral risk factors like credential reuse or missing MFA. Email security gateways operate at the server level, giving end users no visibility into what was flagged or why.

This fragmentation creates security silos — gaps between tools where threats go undetected. A user might have a strong master password but reuse credentials across twelve applications. They might have MFA enabled on their bank account but not their email, which is typically the recovery account for everything else. No

single consumer-grade tool currently surfaces these compound risks together.

Three specific failure modes motivated this project. First, users have no aggregate view of their security posture across all applications. Second, phishing detection at the user level is entirely passive — users are expected to spot threats themselves. Third, credential storage tools exist in isolation and offer no feedback on credential quality or reuse patterns. CyberCore Elite addresses all three in one platform.

III. OBJECTIVES

The project set out four core objectives, each addressing a distinct gap identified in the problem statement:

1. Centralized Dashboard: Build a unified interface that consolidates security metrics across multiple accounts and application categories into a single scoring view, making security status legible at a glance.

2. AI-Based Email Classification: Implement a Multinomial Naive Bayes classifier with TF-IDF preprocessing to detect phishing and spam in connected email accounts, with classification latency under 150 milliseconds.

3. Encrypted Credential Vault: Provide a cloud-synced vault for storing application credentials, with server-side per-user access enforcement and client-side masking for sensitive fields.

4. Weighted Security Scoring: Develop a scoring engine that generates a Security Health Score from ten behavioral and technical parameters, each weighted by its estimated contribution to breach risk, and returns actionable recommendations for low-scoring areas.

IV. LITERATURE REVIEW

Research on personal cybersecurity tools points to a consistent tension between capability and usability. Biddle et al. [1] reviewed twelve years of graphical password research and concluded that the main barrier to adoption is not technical capability but user resistance to behavioral change — a finding that shaped how CyberCore Elite presents its audit results.

On the classification side, Mehta and Sharma [4] compared several ML algorithms for email spam detection and found that Naive Bayes with TF-IDF preprocessing consistently outperformed SVM and decision tree models on short-text classification tasks when training data was limited. This directly informed the choice of MNB over more complex architectures for the email threat module.

Felt et al. [3] introduced the concept of usable security indicators — the principle that security feedback needs to be presented in terms users can act on immediately, rather than raw technical scores. The Security Health Score design in CyberCore Elite, specifically the LOW / MODERATE / CRITICAL badge system with per-parameter recommendations, draws from this framework.

Harris et al. [6] studied credential management behavior in small organizations and found that password reuse was the most commonly observed insecure practice, present in over 60% of participants. That finding anchors the 15% weight assigned to the password reuse parameter in the audit engine. The literature collectively supports the design direction of CyberCore Elite: a tool that audits behavior, not just infrastructure, and communicates risk in plain terms.

V. SYSTEM ARCHITECTURE

The system is organized into three layers connected through REST API boundaries. Each layer is independently deployable, and the ML engine runs as a separate microservice to allow model updates without touching the main server.

A. Presentation Layer

The frontend is built with HTML5, CSS3, and ES6+ JavaScript using a Glassmorphism visual design. It handles UI state, renders Radar and Line charts through Chart.js using the Canvas API, and manages asynchronous API calls. An Android APK wraps the same application through a WebView component, providing mobile access without maintaining a separate native codebase.

B. Business Logic Layer

An Express.js server manages routing, JWT authentication middleware, audit computation, and orchestration of calls to the ML microservice. The audit engine runs entirely server-side: it receives user responses as a JSON payload, computes weighted

subscores per parameter, aggregates them into a total Security Health Score, and persists the result to the database before returning recommendations to the client.

C. Machine Learning Microservice

A separate FastAPI application written in Python serves the Naive Bayes classifier. This decoupling allows the model to be retrained and redeployed independently of the main Express server. Incoming email bodies are passed as plain text; the service returns a class label (Spam or Safe) alongside a posterior probability score that the frontend displays as a confidence percentage.

D. Data Persistence Layer

The database runs on Turso, an edge-native SQLite-compatible platform using the LibSQL protocol. Three normalized tables form the schema: Users (indexed unique usernames with Bcrypt-hashed passwords), History (audit records linked to user_id foreign keys for per-user trend analysis), and Vault (credential records with encrypted_password fields and user_id constraints).

VI. TECHNOLOGY STACK

Table I lists the technologies selected for each layer of the application, along with the primary purpose each serves within the system.

Table I: Technology Stack by Layer

Layer	Technology	Purpose
Frontend	HTML5 / CSS3 / JS	Glassmorphism UI, async API calls
Visualization	Chart.js	Radar & trend charts
Backend	Node.js + Express	API routing, auth middleware
Auth	JWT (JSON Web Token)	Stateless session management
Database	SQLite / Turso (LibSQL)	Edge-native global replication
ML Engine	Python + FastAPI	Spam classification endpoint
ML Library	Scikit-learn	TF-IDF + Naive Bayes training

VII. ALGORITHMS AND METHODOLOGY

A. Multinomial Naive Bayes (MNB)

The spam classifier uses Multinomial Naive Bayes, chosen for its strong performance on text classification tasks with discrete feature counts. Given a word vector x and class c, the classifier computes the posterior probability P(c|x) proportional to P(x|c) multiplied by the prior P(c). The naive independence assumption — that each word contributes to class probability independently of its neighbors — sacrifices linguistic precision but delivers fast, accurate classification on short-form text.

Training used a labeled corpus of 8,500 emails drawn from the SpamAssassin public dataset and supplemented with 1,200 locally collected phishing samples. The model was trained using Scikit-learn's MultinomialNB implementation with Laplace smoothing (alpha=1.0) to handle unseen words in production email streams.

B. TF-IDF Vectorization

Raw email text is preprocessed into numeric feature vectors using TF-IDF weighting before classification. TF-IDF assigns each term a score equal to its frequency in the current document multiplied by the inverse of how many documents in the training corpus contain that term. The effect is to amplify rare, high-signal words ('suspended', 'urgent', 'verify your identity') and suppress common low-signal words ('the', 'and', 'please'). The resulting sparse matrix is fed directly into the MNB classifier.

C. Bcrypt Password Hashing

User authentication passwords are hashed with Bcrypt before storage. Bcrypt generates a unique random salt per password and applies a configurable cost factor — currently set to 12 rounds — that makes each hash computation take approximately 250ms on commodity hardware. This slows brute-force enumeration to a computationally infeasible rate. Because the salt is stored with the hash, rainbow table attacks are ineffective regardless of whether an attacker obtains the database.

D. Software Development Lifecycle

The project followed the Waterfall model. Phase 1 gathered requirements and finalized the JWT

authentication and ML design specs. Phase 2 produced the database schema, API contract, and architecture diagrams. Phase 3 implemented the Node.js API and Python ML engine concurrently. Phase 4 ran unit, integration, regression, and adversarial tests on the full system before deployment.

VIII. SECURITY AUDIT PARAMETERS

The audit engine evaluates ten parameters drawn from common breach patterns. Parameter weights were assigned based on frequency data from the 2023 Verizon DBIR, which identified absent MFA, credential reuse, and phishing as the three primary routes to account compromise. Higher-weighted parameters carry greater influence over the final Security Health Score and generate higher-priority recommendations when scored low. Table II shows the complete parameter set.

Table II: Security Audit Parameters and Assigned Weights

Security Parameter	Weight (%)	Risk Level
Multi-Factor Authentication (MFA)	20%	Critical
Password Reuse Across Apps	15%	High
Password Strength Index	15%	High
Phishing Email Exposure	15%	High
Inactive / Orphaned Accounts	10%	Medium
Recovery Email Security	10%	Medium
Password Update Frequency	5%	Medium
App Permission Auditing	5%	Low
Login Location Monitoring	3%	Low
Security Software Coverage	2%	Low

The scoring logic applies each parameter's weight to a binary or scaled input. MFA status, for example, contributes 0 or 20 points depending on whether MFA is enabled on the user's primary accounts. Password strength is scored on a five-point scale normalized to its 15% maximum weight. The aggregate score determines the Risk Level badge displayed on the dashboard.

IX. MODULE IMPLEMENTATION

A. Security Audit Module

The audit module presents ten questions covering the parameters in Table II. Responses are submitted as a single JSON payload to the POST /audit endpoint. The server computes a weighted sum, generates category-level subscores across Finance, Social, and Communication app groups, and returns the Risk Level badge alongside specific recommendations ranked by potential score improvement. Results are persisted to the History table, enabling the Trend Chart to show score changes over time.

B. Credential Vault Module

The vault implements full CRUD operations. Credentials are stored in the Vault table with an encrypted_password field. Every query to the vault endpoint filters on the JWT-decoded user_id, so a user cannot retrieve another user's credentials regardless of the request payload. Client-side, password fields are masked by default. A visibility toggle reveals the value temporarily without persisting it to the DOM in cleartext. The Radar Chart on the dashboard reflects vault coverage — an empty Finance segment, for example, indicates that no financial accounts are recorded, flagging a potential blind spot in the user's security overview.

C. Live Email Stream Module

The email module connects to the user's inbox over IMAP using port 993 with TLS. The simpleParser library strips attachments, MIME metadata, and headers from incoming messages, isolating the plain-text body before passing it to the FastAPI classification endpoint. The ML service returns a label and confidence score within approximately 112ms. Results are appended to the session threat log and surfaced in the dashboard's real-time feed. Users see each incoming message classified as Safe or Spam with its confidence score displayed inline.

X. RESULTS AND PERFORMANCE ANALYSIS

Table III compares the CyberCore Elite ML pipeline against a baseline Naive Bayes classifier trained on the same corpus without TF-IDF preprocessing. The full pipeline — TF-IDF vectorization followed by MNB classification — outperformed the baseline on every

measured metric. Classification latency dropped from 280ms to 112ms, primarily because TF-IDF's sparse representation reduces the feature dimensionality that MNB must process per call.

Table III: Classification and System Performance Metrics

Metric	Baseline (%)	CyberCore (%)
Spam Detection Accuracy	87.2	94.3
False Positive Rate	8.4	3.1
False Negative Rate	9.1	4.7
Classification Latency (ms)	280	112
Vault Retrieval Time (ms)	—	48

User behavior data collected across 47 test sessions showed a median Security Health Score of 52 at first audit. After completing a second audit cycle — typically two to three days later, after acting on the first round of recommendations — the median score rose to 70. MFA enablement accounted for most of the gain, which was expected given its 20% weight in the scoring model.

The Trend Chart visualization proved to be the feature users interacted with most during testing. Seeing a numeric score improve over time created a feedback loop that motivated further action — a behavioral outcome that aligns with findings from Felt et al. [3] on the value of legible security indicators.

XI. TESTING FRAMEWORK

Testing covered five levels: unit, integration, regression, adversarial, and load. Table IV summarizes the scope and outcome of each test type.

Table IV: Testing Framework — Scope and Outcomes

Test Type	Scope	Outcome
Unit Testing	Express routes (POST /register, GET /vault)	All routes passed
Integration Testing	Score → Recommendation → DB persistence	Data integrity confirmed
Regression Testing	Vault logic after IMAP addition	No regressions found
Adversarial Testing	JWT bypass attempts on protected routes	Middleware blocked all attempts
Load Testing	500 concurrent requests to API	Avg. response < 200 ms

Unit tests verified each Express route in isolation, including edge cases such as empty vault entries and malformed audit payloads. Integration tests walked the full data path from user input through score calculation, recommendation generation, and database persistence, confirming that no data was lost or corrupted between layers.

Adversarial testing targeted the JWT middleware specifically. Attempts included presenting expired tokens, tokens signed with an incorrect secret, and tokens with a tampered payload (user_id substitution). All attempts were rejected at the middleware layer before reaching any route handler. No unauthorized data access was achieved through any of these methods.

XII. SECURITY HARDENING MEASURES

Security hardening was applied at three levels across the stack:

JWT Middleware: Every protected route — vault operations, audit history retrieval, and email scanning — passes through a decode-and-verify middleware function before any business logic executes. Expired or malformed tokens return HTTP 401 immediately, with no information about the failure reason exposed in the response body to prevent oracle attacks.

SQL Parameterization: All database interactions use parameterized queries through the LibSQL execute interface, with ? placeholders for all user-supplied values. No string interpolation or concatenation is used anywhere in the database access layer, eliminating the SQL injection surface entirely.

Environment Variable Management: All sensitive configuration — JWT signing secrets, Turso database URLs, and IMAP credentials — is stored in .env files that are excluded from version control via .gitignore. Hardcoded credentials do not appear anywhere in the

committed codebase, verified through automated secret scanning on each push.

XIII. CONCLUSION

CyberCore Elite demonstrates that enterprise-grade security concepts — behavioral audit scoring, email threat classification, and encrypted credential management — can be made useful for everyday users without requiring technical expertise. The system reached 94.3% spam classification accuracy in testing. Users who completed two audit cycles improved their median Security Health Score by 18 points, with MFA adoption driving most of the gain.

The more telling result is how straightforward the underlying fixes are. MFA. Unique passwords. Awareness of which accounts sit exposed. None of this is new knowledge. What CyberCore Elite provides is a single surface where these risks are visible together — and that, it turns out, is often enough to prompt action.

The two-column architecture separating the ML microservice from the main API server proved its value during testing: the classifier was retrained twice during the test period without any downtime to the audit or vault functionality. That separation will matter more as the ML layer grows in complexity toward the BERT-based model planned for the next phase.

XIV. FUTURE WORK

Three development directions are planned following this phase:

BERT-Based Classification: The MNB classifier will be replaced with a fine-tuned BERT transformer model. BERT's bidirectional context encoding should improve detection of sophisticated phishing emails that use indirect phrasing or obfuscated language — the type of content that word-frequency models tend to miss because the individual terms appear benign in isolation.

Blockchain Audit Logging: Forensic audit logs will be written to a private blockchain ledger to ensure tamper-evident records. This is primarily relevant for enterprise deployments where compliance requirements demand non-repudiable audit trails. The immutability guarantees of the ledger would make it impossible to quietly alter or delete audit history.

Mobile Push Notifications: The Android application will be extended with real-time push notifications triggered when the IMAP stream classifies an incoming email as spam or phishing. Users will receive an alert with the message subject and confidence score without needing the dashboard open.

OAuth Integration: Future versions will support OAuth 2.0 to allow the system to pull account security metadata directly from connected services — eliminating the current reliance on self-reported audit answers and replacing it with verified data from the source.

XV. REFERENCES

[1] R. Biddle, S. Chiasson, and P.C. van Oorschot, "Graphical passwords: Learning from the first twelve years," ACM Computing Surveys, vol. 44, no. 4, pp. 1–41, 2012.

[2] Z. Li, S. Alrwais, Y. Xie, F. Yu, and X. Wang, "Finding the linchpins of the dark web: A study on topologically dedicated hosts on malicious web infrastructures," IEEE Symposium on Security and Privacy, 2014.

[3] A.P. Felt, R.W. Reeder, A. Ainslie, H. Harris, M. Walker, C. Thompson, M. Acer, E. Morant, and S. Consolvo, "Rethinking connection security indicators," USENIX Symposium on Usable Privacy and Security (SOUPS), 2016.

[4] A. Mehta and V. Sharma, "A comparative study of machine learning algorithms for spam email detection," International Journal of Computer Applications, vol. 183, no. 8, pp. 12–18, 2021.

[5] Verizon, "2023 Data Breach Investigations Report," Verizon Business, Tech. Rep., 2023. [Online]. Available: https://www.verizon.com/business/resources/reports/dbir/

[6] M.A. Harris, K.P. Patten, and E. Regan, "Mobile and connected device security considerations: A dilemma for small and medium enterprise business mobile device use," Issues in Information Systems, vol. 14, no. 2, pp. 1–10, 2013.

[7] O. Almomani, A. Al-Sewailem, and B. B. Gupta, "Phishing website detection with semantic features based on machine learning classifiers," International Journal on Semantic Web and Information Systems, vol. 16, no. 4, pp. 1–20, 2020.