Surge: Agentic Network Security Analysis Platform

Submitted on April 10th, 2026 by

Jonathan Contreras, Brianna Hinds, Sean Moning, Taurean Muhammad

Project Advisors:

Dr. Stephen Lyford, Professor Manuel Aguilar

Houston Christian University

Department of Engineering

TABLE OF CONTENTS

1. FORMAL PROJECT DESCRIPTION 5
2. ENGINEERING DESIGN SPECIFICATION 8
a. Primary Intended Use 8
b. Predictable Unintended Use 9
c. Special Purpose Features 9
d. Requirements 10
i. Functional Performance 10
ii. Operating Environment 12
iii. Geometric Limitations 13
iv. System Maintenance 13
v. Failures and System Recovery 14
3. DESIGN ALTERNATIVES 15
LLM Serving: vLLM vs. Ollama 15
Multi-GPU Inference vs. Isolated GPU Assignment 15
Database: Convex/Supabase vs. Self-Hosted PostgreSQL 16
Agent Framework: LangChain Agents vs. LangGraph StateGraph 16
Context Window and Fine-Tuning Decisions 17
Frontend Framework: React SPA vs. Next.js 17
Network Topology: Flat Network vs. VLAN Segmentation 18
4. TECHNICAL DETAILS OF THE SOLUTION 19
4.1 Agent Pipeline Architecture 19
4.2 Backend API Server 21
4.3 Database Schema 22
4.4 Frontend Dashboard 24
4.5 Network Infrastructure 27
4.6 Governance and Security Layer 30
4.7 Machine Learning Model 31
4.8 Codes and Standards 33
4.9 Testing and Validation 33
5. CONSTRAINTS 36
a. Public Health, Safety and Welfare Factors 36
b. Global Factors 36

c. Cultural Factors 36
d. Social Factors 37
e. Environmental Factors 37
f. Economic Factors 37
Additional Applicable Constraints 38
6. MATERIAL LEARNED OUTSIDE OF THE CLASSROOM 39
7. COST ANALYSIS 42
8. FUTURE EXPANSION OF PROJECT 43
REFERENCES 46

TABLE OF FIGURES

Figure 1. Full System Architecture 7
Figure 2. LangGraph agent pipeline directed acyclic graph (DAG) with LLM tier assignments 21
Figure 3. Entity-relationship diagram of the Surge database schema. 24
Figure 4. Dashboard main view showing network topology, device list, activity feed, and vulnerability summary. 25
Figure 5. Exploits page showing the filterable CVE priority queue. 26
Figure 6. Lab network topology with VLAN segmentation, routing, and firewall configuration. 29
Figure 7. Tiered governance model for nmap flag sanitization. 31
Figure 8. Machine learning feature engineering pipeline for CVSS score prediction. 32

1. FORMAL PROJECT DESCRIPTION

Network vulnerability assessment is one of the most critical and labor-intensive tasks in cybersecurity. Industry-standard tools such as Nessus, OpenVAS, and Metasploit provide powerful scanning capabilities, but they demand significant expertise to configure, interpret, and act upon. A typical manual assessment of even a small network requires a trained analyst to select appropriate scan parameters, interpret raw nmap output, cross-reference discovered services against CVE databases, evaluate risk severity, and compile findings into actionable reports—a process that can take hours or days depending on network size. In educational settings, this complexity creates a steep barrier: students learning penetration testing must simultaneously master networking fundamentals, scanning tool syntax, vulnerability taxonomy (CVE/CVSS/CPE/CWE), and report writing, often with limited lab time and instructor availability. There is a clear need for a platform that can automate the mechanical aspects of this workflow while preserving the analytical depth that makes the results meaningful.

Surge is an autonomous network security analysis platform developed as a senior capstone project at Houston Christian University during the 2025–2026 academic year. The platform addresses this challenge directly by orchestrating a pipeline of AI agents that automate the full penetration-testing workflow—from initial reconnaissance through vulnerability discovery to risk-prioritized reporting—within a controlled, ethically-bounded lab environment. Unlike existing tools that rely on static rule sets and signature matching, Surge leverages large language models to make adaptive decisions at each stage of the assessment, producing human-readable analysis rather than raw scan dumps.

The system employs LangGraph, a Python-based agent orchestration framework, to coordinate a sequence of specialized agents: Reconnaissance, Enumeration (OS Fingerprinting), Vulnerability Analysis, CVSS Risk Scoring, and Reporting. Each agent in the pipeline operates with clearly defined inputs and outputs, passing structured state through a shared AgentState object. The Reconnaissance agent conducts iterative nmap scanning with LLM-driven escalation decisions; the Vulnerability agent queries live CVE databases and normalizes results using large language models; and the CVSS Scoring agent applies a pre-trained XGBoost regressor to predict risk scores that augment raw CVSS data. All agent decisions and scan parameters are constrained by a tiered flag-sanitization governance layer that prevents dangerous scan configurations and command injection.

The platform is served through a dual-component architecture: a FastAPI backend providing RESTful endpoints and WebSocket-based real-time streaming, and a Next.js/TypeScript frontend dashboard deployed on the HCU Engineering server. The dashboard provides live scan monitoring with a D3-powered force-directed network topology graph, device inventories with severity indicators, a CVE exploitation queue, and multi-audience report generation (executive, technical, public, and comprehensive final reports). The backend persists all scan data—devices, vulnerabilities, activity events, and generated reports—in a self-hosted PostgreSQL database.

LLM inference is powered by two locally-hosted models running on dual Nvidia Quadro RTX 8000 GPUs (48 GB VRAM each) via Ollama, with supplemental cloud-based reasoning through OpenRouter for analysis-heavy tasks. This hybrid local/cloud approach balances latency-sensitive structured extraction against reasoning-intensive synthesis, while keeping sensitive network data on-premises.

Figure 1 illustrates the end-to-end system architecture of the Surge platform, organized into four logical layers. On the left, the lab network provides the scan targets across two VLAN-segmented subnets, connected through Cisco switching and routing infrastructure with firewall-enforced perimeter security. The center layer shows the LangGraph agent pipeline, which progresses from initial reconnaissance through parallel OS fingerprinting and analysis branches, converging into vulnerability discovery, ML-based CVSS scoring, and final report generation. The agents are served by dual Ollama LLM instances on isolated GPUs, supplemented by an OpenRouter cloud endpoint for analysis-heavy reasoning tasks. On the right, the FastAPI backend exposes REST and WebSocket interfaces to the Next.js dashboard, with all scan data persisted in a self-hosted PostgreSQL database. Tailscale VPN provides secure remote access to the server infrastructure.

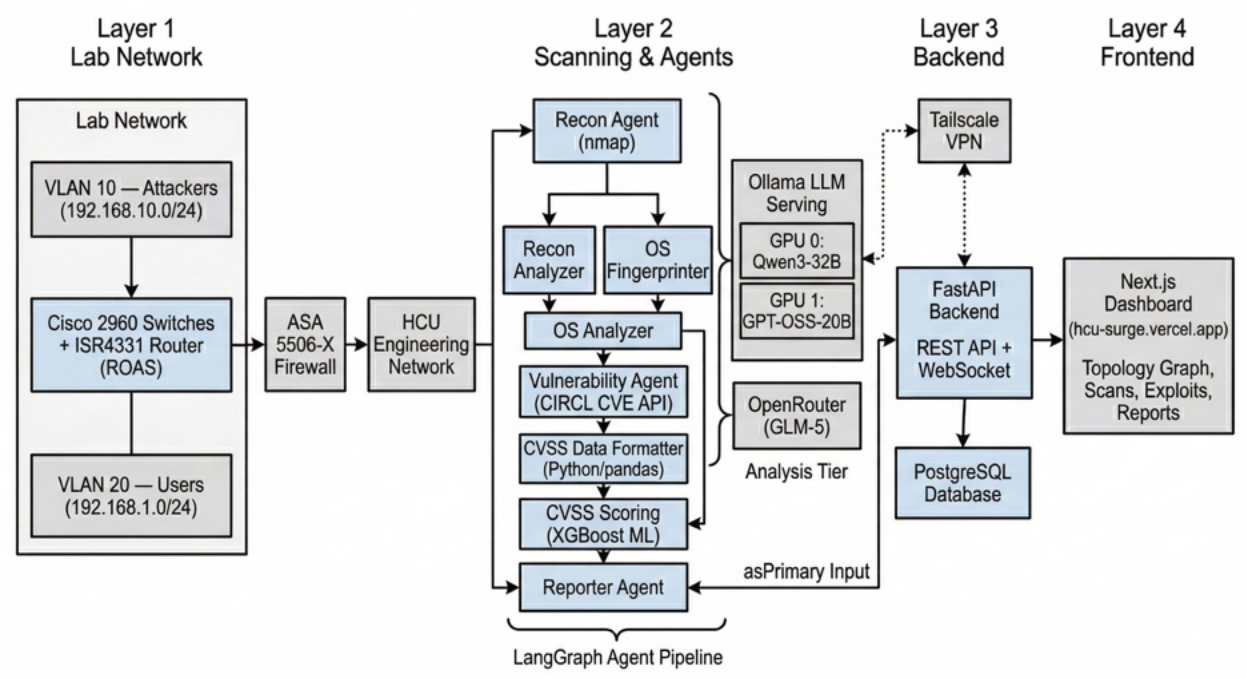

Figure 1. Full System Architecture

2. ENGINEERING DESIGN SPECIFICATION

This section contains the Engineering Design Specification (EDS) for the Surge platform. It details the system's intended use, potential for misuse, distinguishing features, and the functional and non-functional requirements the system must satisfy.

a. Primary Intended Use

Surge is designed for use by cybersecurity students, educators, and lab administrators who need to conduct network vulnerability assessments within controlled academic environments. The platform's primary intended workflow begins with automatic network detection: a Python script identifies the network interface of the machine Surge is running on and derives the local subnet as the default scan target. The operator may also specify a custom IP range through the web dashboard if they wish to target a particular segment. After selecting a scan type (quick, normal, deep, or stealth), the system autonomously executes a multi-agent pipeline that discovers hosts, fingerprints operating systems, identifies known CVEs against discovered services, predicts CVSS risk scores using a trained machine learning model, and generates a prioritized security report.

The intended end-users are Houston Christian University engineering and computer science students, faculty advisors overseeing security labs, and potentially other academic institutions seeking an accessible, self-hosted penetration-testing training tool. The system is explicitly scoped for lab use against networks the operator is authorized to scan; it is not intended for use against production or third-party networks.

b. Predictable Unintended Use

Because Surge orchestrates real nmap scans and queries live CVE databases, the most significant foreseeable misuse would be directing the platform at networks outside the authorized lab environment. To mitigate this, the system operates within a VLAN-segmented network behind a Cisco ASA 5506-X firewall, and scan targets are validated at the API layer. The tiered flag-sanitization module in the governance layer enforces strict whitelists on nmap flags per scan tier, preventing shell injection and blocking excessively aggressive scan configurations at lower tiers.

Another predictable misuse involves interpreting Surge's automated vulnerability reports as a complete security audit. The system is a training and analysis aid; its outputs depend on the completeness of nmap scan results and the currency of the CIRCL CVE database, and should always be reviewed by a human analyst before any remediation decisions are made.

c. Special Purpose Features

Several features distinguish Surge from existing network security tools and academic lab platforms:

LLM-Orchestrated Agent Pipeline: Unlike traditional vulnerability scanners that follow static rule sets, Surge uses large language models to make adaptive decisions during reconnaissance. The Recon agent evaluates scan deltas between iterations and autonomously decides whether to escalate scan intensity, providing more thorough coverage than a single-pass scanner.

Hybrid CVSS Scoring: Surge augments raw CVSS scores from the National Vulnerability Database with predicted scores from a custom-trained XGBoost regressor. The ML model is trained on historical CVE data with features including access vector, complexity, authentication requirements, and impact metrics, plus sentence-BERT embeddings of vulnerability summaries.

This provides a second risk signal that can surface vulnerabilities whose raw CVSS may understate practical exploitability.

Real-Time Dashboard with Network Topology: The Next.js dashboard renders a live, interactive D3 force-directed graph of the network topology as scans progress. Devices are color-coded by severity, and clicking a node reveals its associated CVEs with links to the NIST National Vulnerability Database.

Multi-Audience Report Generation: The reporting module generates four report types—executive (board-level risk summary), technical (CISO/security team detail), public (no sensitive data), and final (comprehensive)—each produced by specialized LLM system prompts that tailor content to the target audience.

Tiered Governance Layer: A flag-sanitization system categorizes nmap parameters into low, medium, and high tiers with explicit whitelists, port limits, timeout budgets, and blocked shell metacharacters. This is critical for a platform that grants AI agents autonomous tool use.

d. Requirements

This section details the requirements that the Surge platform must meet.

i. Functional Performance

The system comprises three primary functional subsystems: the agent pipeline, the API server, and the frontend dashboard.

Agent Pipeline: The LangGraph-orchestrated pipeline must execute five sequential agent phases (Reconnaissance, OS Fingerprinting, Vulnerability Analysis, CVSS Data Formatting, and CVSS Scoring) with parallel fan-out at the Recon Analysis/OS Finder stage and convergence at the

Reporter. The Reconnaissance agent supports iterative scanning with configurable convergence criteria: a maximum of 10 scan iterations, 4 consecutive no-change iterations for early termination, and a 2-hour time budget. All agents communicate through a shared AgentState TypedDict containing fields for scan configuration, results from each phase, network topology, and the final report.

Backend API: The FastAPI server exposes RESTful endpoints for scan lifecycle management (POST /scans, GET /scans, GET /scans/{id}), device enumeration (GET /devices), vulnerability querying (GET /vulns), dashboard data (GET /dashboard/{scan_id}), and report generation (POST /reports/generate, GET /reports/{id}). A WebSocket endpoint (WS /ws/scans/{scan_id}) provides real-time scan progress streaming with per-agent-node completion percentages. The API must handle concurrent scan operations using asyncio with thread offloading for CPU-bound agent tasks.

Frontend Dashboard: The Next.js application must render six primary views: Dashboard (network topology, device list, activity feed, vulnerability chart, exploit queue), Scans (launch, monitor, and review scans), Topology (dedicated full-screen network graph), Exploits (CVE priority queue with filtering by severity, exploit availability, and status), Reports (template selection and generation), and Settings (agent configuration, network defaults, appearance). The dashboard supports three view modes—Live, Latest, and History—with automatic mode switching based on active scan detection.

LLM Inference: The system operates a two-tier LLM configuration. The fast tier uses a locally-hosted GPT-OSS-20B model via Ollama (port 11435, GPU 1) for structured JSON extraction tasks such as recon decisions and data formatting. The analysis tier routes to an OpenRouter-hosted model (GLM-5 or DeepSeek) for reasoning-intensive tasks including vulnerability

analysis, report generation, and OS analysis. An environment flag (USE_ONLINE=1) enables routing all traffic through OpenRouter for testing.

ii. Operating Environment

Surge operates within a purpose-built lab network at Houston Christian University's Engineering department. The network infrastructure consists of a VLAN-segmented topology with two logical zones:

VLAN 10 (Attackers): Subnet 192.168.10.0/24, housing simulated attacker machines (PC2 at 192.168.10.5, PC3 at 192.168.10.6) and the AI scanner (192.168.1.9). Default gateway: 192.168.10.1.

VLAN 20 (Users): Subnet 192.168.1.0/24, hosting simulated internal user machines (PC0 at 192.168.1.7, PC1 at 192.168.1.8). Default gateway: 192.168.1.1.

The network uses two Cisco 2960-24TT Layer 2 switches connected via an 802.1Q trunk link (interface F0/5). A Cisco ISR4331 router performs inter-VLAN routing using a router-on-a-stick (ROAS) configuration with subinterfaces g0/0/1.1 (VLAN 10) and g0/0/1.2 (VLAN 20). A Cisco ASA 5506-X firewall enforces perimeter security between the internal network and a second ISR4331 router simulating internet connectivity. NAT overload (PAT) translates internal addresses through the router's external interface (g0/0/0, 10.10.1.160) to the HCU Engineering network.

The Surge server itself runs on a university-provided engineering server with dual Nvidia Quadro RTX 8000 GPUs (48 GB VRAM each, Turing architecture, compute capability 7.5), accessible remotely via Tailscale VPN. The software stack requires Python 3.11+, Node.js 18+,

PostgreSQL 14+, and nmap installed on the host system. Docker Compose is used for deployment orchestration.

iii. Geometric Limitations

As a software platform, Surge does not have physical geometric constraints in the traditional sense. However, the system's computational geometry is bounded by the GPU hardware: each RTX 8000 provides 48 GB of VRAM, which constrains the maximum model size that can be loaded. The Qwen3-32B model (approximately 20 GB at Q4_K_M quantization) and the GPT-OSS-20B model (approximately 14 GB at Q4_K_M) each fit within a single GPU's memory, but larger models or higher-precision quantizations would exceed available VRAM. The system's network scanning scope is bounded by the lab's VLAN topology: two /24 subnets providing a maximum of 508 addressable hosts.

iv. System Maintenance

Surge's maintenance considerations span software, model, and infrastructure layers. The codebase is maintained in version-controlled Git repositories (currently being consolidated from two repositories into a monorepo). Dependency versions are pinned in requirements.txt and package.json with specified version ranges to prevent breaking changes during updates.

The XGBoost CVSS prediction model and its associated encoders (one-hot encoder, TF-IDF vectorizer, sentence-BERT embeddings) are serialized as .pkl and .json files in the model/ directory. Model retraining would require updated CVE training data (currently sourced from a merged_cve.csv dataset) and re-execution of the training notebooks in the notebooks/ directory.

The Ollama LLM instances require periodic model updates as new quantized versions become available. GPU driver updates must maintain CUDA compatibility with the Turing architecture

(compute capability 7.5). The PostgreSQL database requires standard maintenance (vacuuming, index rebuilding) as scan data accumulates.

v. Failures and System Recovery

The system is designed with several failure-handling mechanisms. At the API layer, the FastAPI lifespan context manager runs stale-scan cleanup on startup, marking any scans left in "running" state from a previous crash as "failed." The scan background task wraps the entire agent pipeline in try/except blocks, persisting error messages to the database and updating scan status to "failed" if any agent raises an unhandled exception.

Within the agent pipeline, the Reconnaissance agent implements convergence guards: if four consecutive scan iterations produce no new hosts or ports, the agent terminates gracefully rather than looping indefinitely. The nmap scanner tool has configurable per-scan timeouts, and the CVE search tool implements rate limiting (0.5s between queries) to avoid API throttling.

For LLM failures, the system uses a fallback model configuration: if the primary analysis-tier model is unavailable, the USE_ONLINE flag can redirect to alternative providers. The JSON extraction utility (execution/json_extract.py) uses regex-based parsing with fallback strategies to handle malformed LLM outputs.

In the event of total server failure, recovery involves restarting Docker Compose services. Scan data persisted in PostgreSQL survives service restarts. The frontend, hosted on Vercel, operates independently and degrades gracefully to cached/sample data when the backend is unreachable.

3. DESIGN ALTERNATIVES

Throughout development, the team faced several significant architectural decisions where multiple viable approaches existed. Each choice involved careful evaluation of trade-offs between performance, feasibility, hardware constraints, and timeline. This section documents the alternatives considered, the reasoning behind each decision, and the consequences of those choices for the final system. These decisions are presented in roughly chronological order, reflecting the sequence in which they arose during development.

LLM Serving: vLLM vs. Ollama

The team initially planned to use vLLM, a high-performance LLM inference engine, for serving local models. vLLM's PagedAttention mechanism and continuous batching offered significant throughput advantages over simpler serving solutions. However, vLLM's MXFP4 quantization format requires GPU compute capability 8.0 or higher (Ampere architecture and newer). The project's RTX 8000 GPUs are Turing-generation with compute capability 7.5, making vLLM incompatible without falling back to higher-precision formats that would not fit in VRAM for the target model sizes. Ollama was chosen as a pragmatic alternative because it natively supports GGUF quantization formats (Q4_K_M, Q5_K_M) that are compatible with any CUDA-capable GPU.

Multi-GPU Inference vs. Isolated GPU Assignment

The dual RTX 8000 configuration presented an opportunity for multi-GPU inference, which would allow a single large model to be split across both GPUs. However, the two GPUs are connected via PCIe rather than NVLink, meaning inter-GPU communication would traverse the

PCIe bus with significant latency overhead. For the project's workload—which involves multiple independent agent tasks that can run concurrently—assigning separate models to separate GPUs (using CUDA_VISIBLE_DEVICES isolation) proved more efficient. GPU 0 (port 11434) hosts Qwen3-32B for reconnaissance analysis, while GPU 1 (port 11435) hosts GPT-OSS-20B for other agent tasks. This allows both models to serve requests in parallel without contention.

Database: Convex/Supabase vs. Self-Hosted PostgreSQL

Cloud-hosted database solutions including Convex and Supabase were initially evaluated for their ease of setup and built-in real-time subscription features. However, several factors favored self-hosted PostgreSQL: (1) scan data includes network topology and vulnerability information that should remain on-premises for security and compliance reasons; (2) the scan pipeline generates large intermediate state objects that benefit from local database latency; (3) self-hosting eliminates dependency on third-party service availability and pricing tiers. SQLAlchemy 2.0 with async drivers (asyncpg for PostgreSQL, aiosqlite for local development) provides the necessary abstraction layer.

Agent Framework: LangChain Agents vs. LangGraph StateGraph

The initial prototype used LangChain's agent executor pattern, where a single LLM decides which tools to invoke in a reactive loop. This approach proved insufficient for the multi-stage pipeline because: (1) a single agent loop could not cleanly separate concerns between reconnaissance, enumeration, and vulnerability phases; (2) intermediate results needed to persist across stages; and (3) parallel execution of independent agents (e.g., Recon Analyzer and OS Finder) was not naturally supported. LangGraph's StateGraph abstraction provides declarative

DAG-based orchestration with typed state passing, fan-out/fan-in parallelism, and clear node boundaries, which maps directly to the desired pipeline topology.

Context Window and Fine-Tuning Decisions

The Qwen3-32B model supports a 128K-token context window, but using the full window for every agent task significantly impacts inference throughput. Through experimentation, the team found that reducing context windows to task-appropriate sizes (e.g., 8K tokens for reconnaissance and enumeration tasks) yielded substantial throughput gains with no measurable quality degradation for structured extraction tasks. Fine-tuning local models on domain-specific security data was considered but deprioritized due to timeline constraints; the combination of carefully-crafted system prompts (663 lines of prompt engineering in agents/prompts.py) and appropriate model selection proved sufficient for the project's requirements.

Frontend Framework: React SPA vs. Next.js

A plain React single-page application (SPA) with Create React App or Vite was initially considered for the dashboard frontend. While simpler to set up, a plain SPA would have required manual configuration for routing, environment variable management, API proxying, and production build optimization. Next.js was selected because its App Router provides file-system-based routing that maps naturally to the dashboard's page structure (scans, exploits, reports, settings), built-in API route handlers that simplified development-time proxying to the FastAPI backend, and automatic code splitting that keeps initial page load times fast despite the heavyweight D3 visualization library. The server-side rendering capabilities of Next.js were not a primary driver for this project since the dashboard is deployed as a static export to Vercel, but

the framework's developer experience and ecosystem maturity (TypeScript-first, Tailwind CSS integration) significantly accelerated frontend development.

Network Topology: Flat Network vs. VLAN Segmentation

The simplest lab network option would have been a flat Layer 2 network with all devices on a single subnet, requiring only a single switch and no routing configuration. While this would have been faster to deploy, it would not have tested Surge's ability to discover and scan across network boundaries—a common scenario in real enterprise environments. The team chose a VLAN-segmented design with two distinct subnets (VLAN 10 for attackers at 192.168.10.0/24 and VLAN 20 for users at 192.168.1.0/24), connected through router-on-a-stick inter-VLAN routing and protected by an ASA firewall. This design mirrors the kind of segmented corporate network that a security analyst would encounter professionally, and it validates that Surge's auto-detection and scanning logic works correctly across subnet boundaries with NAT translation and firewall traversal in the path.

4. TECHNICAL DETAILS OF THE SOLUTION

This section provides the detailed technical narrative of the Surge platform's design and implementation, organized by subsystem.

4.1 Agent Pipeline Architecture

The core of Surge is a LangGraph StateGraph that orchestrates eight agent nodes in a directed acyclic graph. The graph is defined in workflow/graph.py and executes as follows:

Recon Agent (recon): The entry point of the pipeline. This agent conducts iterative network reconnaissance using nmap. On each iteration, it presents the LLM (fast tier) with the current scan results and asks it to decide whether to escalate scan intensity, adjust targets, or terminate. The agent tracks discovered hosts, newly-opened ports, and service banners across iterations. Convergence is governed by three criteria: a maximum of 10 iterations, 4 consecutive no-change iterations, and a 2-hour wall-clock time budget. Scan flags are validated through the governance/sanitization.py whitelist before execution. Output includes discovered hosts, parsed network topology (via utils/topology.py), and raw XML content.

Recon Analyzer (recon_analyzer) and OS Finder (os_finder): These two nodes execute in parallel (fan-out) after the Recon Agent completes. The Recon Analyzer uses the analysis-tier LLM to produce a high-level markdown summary of the network environment and writes an early dashboard snapshot. The OS Finder conducts aggressive OS fingerprinting (-sS -sV -O --osscan-guess) against discovered hosts in batches of 10, extracting CPE (Common Platform Enumeration) identifiers for downstream CVE correlation.

OS Analyzer (os_analyzer): A fan-in node that receives results from both parallel branches. It uses the analysis-tier LLM to process OS fingerprinting data in batches, slimming payloads to stay within context limits and identifying unusual or outdated OS versions.

Vulnerability Agent (vulnerability): Queries the CIRCL CVE API (https://cve.circl.lu/api/search/) for each product identified during OS fingerprinting, extracting the top 10 CVEs per product. The LLM then normalizes raw CVE data into a structured CVEEntry schema with consistent CVSS fields. Rate-limited to 0.5s between API calls.

CVSS Data Formatter (cvss_data_formatter): A data-cleaning stage that normalizes all vulnerability entries to the feature schema expected by the ML model. Handles null values, data type validation, and categorical field consistency for the six CVSS metric fields (access_authentication, access_complexity, access_vector, impact_availability, impact_confidentiality, impact_integrity).

CVSS Scoring (cvss_scoring): Applies a pre-trained XGBoost regressor (model/xgb_regressor.json) to predict CVSS scores. The model was trained on historical CVE data using 137+ features including one-hot encoded categorical CVSS fields, TF-IDF vectors of CWE names, and sentence-BERT embeddings of vulnerability summaries. The scoring agent generates the final dashboard payload including network topology with severity coloring.

Reporter (reporter): The terminal node that aggregates all pipeline outputs and generates a comprehensive markdown security report using the analysis-tier LLM. The reporter also supports on-demand specialized reports (executive, technical, public) through the API's /reports/generate endpoint, each using a distinct system prompt tailored to the target audience.

Figure 2 visualizes the LangGraph agent pipeline as a directed acyclic graph. The Recon Agent node at the top feeds into a parallel fan-out where the Recon Analyzer and OS Fingerprinter execute concurrently. Both branches converge at the OS Analyzer node, which feeds sequentially into the Vulnerability Agent, CVSS Data Formatter, CVSS Scoring, and finally the Reporter. Dashed arrows indicate LLM tier assignments: nodes connected to the fast-tier Ollama instance for structured extraction, and nodes connected to the analysis-tier OpenRouter endpoint for reasoning-heavy synthesis.

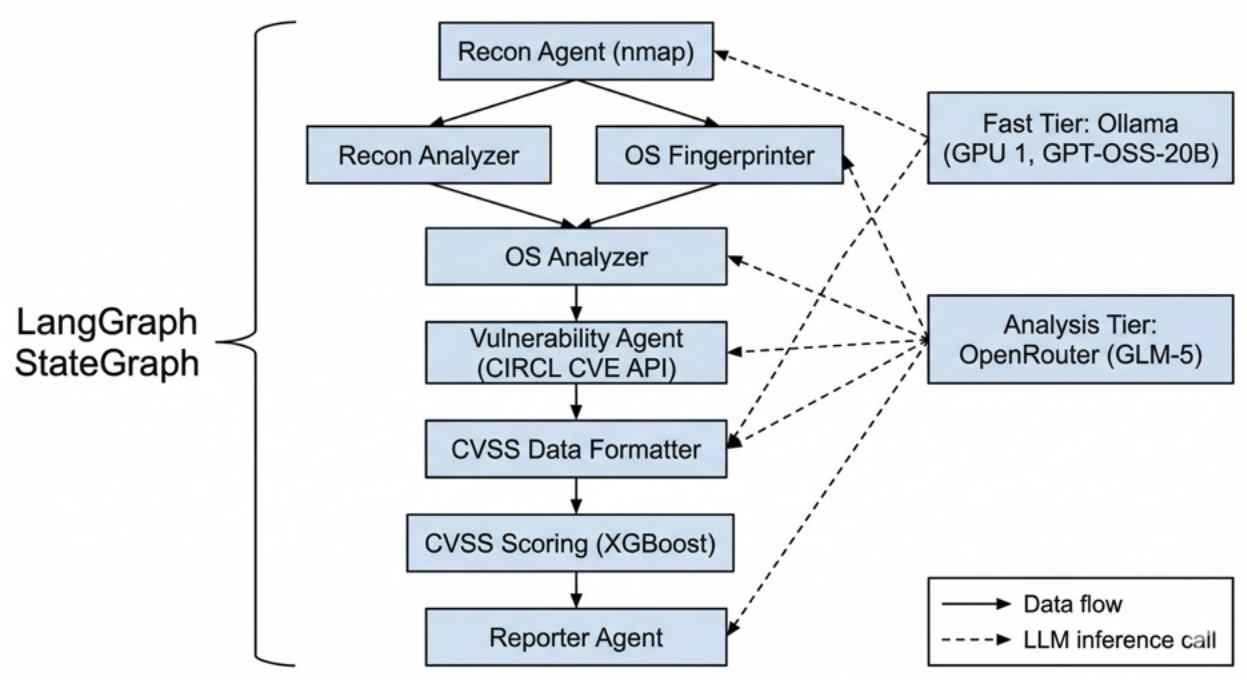

Figure 2. LangGraph agent pipeline directed acyclic graph (DAG) with LLM tier assignments

4.2 Backend API Server

The backend is built with FastAPI (v0.115+) and runs on Uvicorn as an ASGI server. The API is organized into five route modules:

Scans (api/routes/scans.py): Manages the full scan lifecycle. The POST /scans endpoint accepts a ScanRequest with optional name, target (auto-detected via ARP if omitted), scan_type (quick/normal/deep/stealth mapped to low/medium/high tiers), and agent_mode (autonomous). Scans execute as background tasks using asyncio.to_thread to run the synchronous LangGraph pipeline without blocking the event loop. ContextVar propagates the scan_id into thread workers for activity logging. The WebSocket endpoint streams ScanProgressEvent objects with per-node completion percentages (recon: 15%, recon_analyzer: 30%, os_finder: 30%, os_analyzer: 45%, vulnerability: 65%, cvss_data_formatter: 75%, cvss_scoring: 85%, reporter: 100%).

Devices (api/routes/devices.py): Returns discovered network devices for a given scan, including IP, hostname, OS, device type, severity classification, and CVSS score.

Vulnerabilities (api/routes/vulns.py): Returns CVE records for a scan with fields including CVE ID, raw and predicted CVSS scores, severity, affected host/product, exploit availability, and remediation guidance.

Dashboard (api/routes/dashboard.py): Serves real-time dashboard data including network topology (nodes with id, label, nodeType, status; links with source, target, type), aggregate statistics, and the activity event feed.

Reports (api/routes/reports.py): Lists available report templates, generates reports from the agent pipeline's final_report.md output using specialized system prompts, and retrieves generated reports.

4.3 Database Schema

The database is implemented using SQLAlchemy 2.0 with async session support. Five primary tables store scan data:

ScanModel: The master scan record with UUID primary key, target range, scan type, agent mode, status (running/completed/failed), timestamps, duration, error messages, and aggregate statistics (device count, vulnerability count, average CVSS).

DeviceModel: Discovered network devices linked to scans via foreign key. Stores IP, hostname, MAC address/vendor, device type, OS name/accuracy, severity, CVSS score, subnet, and raw JSON for OS fingerprint and service data.

VulnerabilityModel: CVE records linked to both scans and devices. Stores CVE ID, raw and predicted CVSS scores, severity, summary, exploit availability, remediation guidance, all six CVSS metric fields, and CWE classification.

ReportModel: Generated security reports linked to scans. Stores template ID, raw markdown, and parsed sections (executive summary, scope, methodology, key findings, recommendations, risk counts by severity).

ActivityEventModel: Real-time activity feed events linked to scans. Stores event type (info/warning/critical/patch/exploit), message, detail, associated IP, and originating agent node.

Figure 3 presents the entity-relationship diagram for the Surge database. The ScanModel table serves as the root entity, with one-to-many relationships to DeviceModel, VulnerabilityModel, ReportModel, and ActivityEventModel. VulnerabilityModel has a dual foreign key relationship, linking to both ScanModel (for scan-level queries) and DeviceModel (for device-level vulnerability lookups). This schema enables efficient querying at multiple granularities: all vulnerabilities for a scan, all vulnerabilities for a specific device, or all activity events for a given agent node.

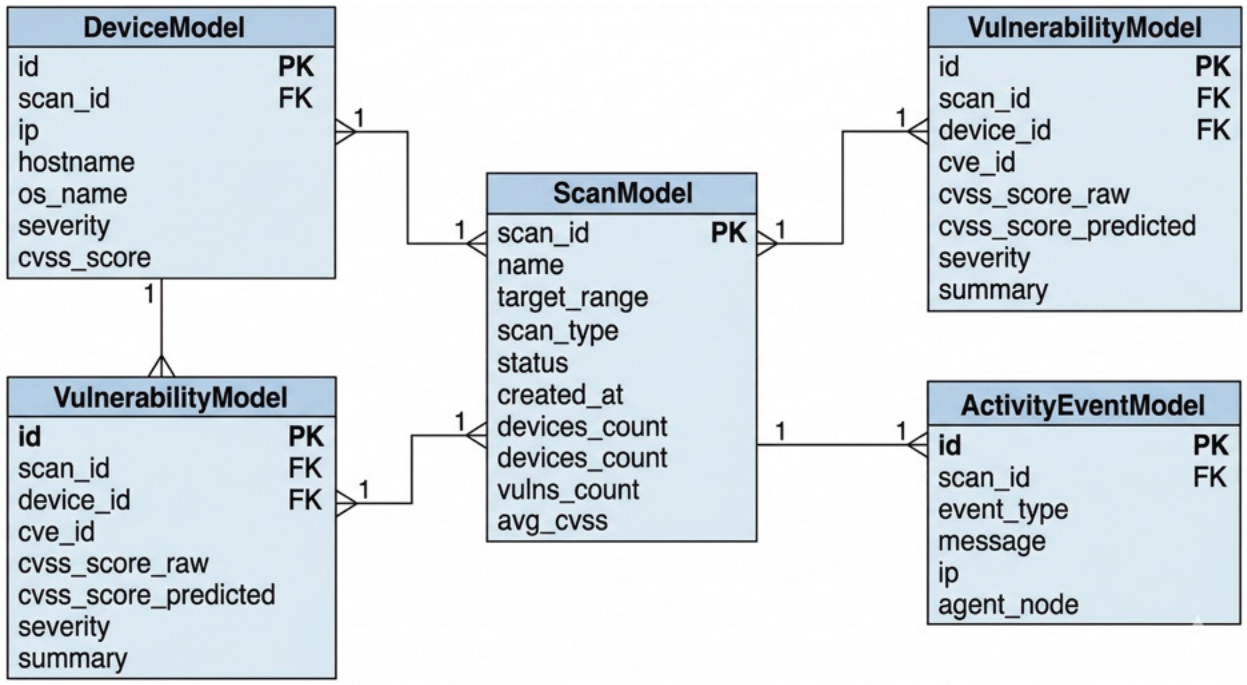

Figure 3. Entity-relationship diagram of the Surge database schema.

4.4 Frontend Dashboard

The frontend is a Next.js 16 application written in TypeScript, styled with Tailwind CSS v4, and deployed on Vercel at hcu-surge.vercel.app. The application uses client-side tab-based navigation (not Next.js file-based routing) with six primary views.

The Dashboard view is the primary operational interface, combining a D3 force-directed network topology graph (using d3-force v3.0.0 for physics simulation with pan/zoom and node selection), a device list with severity indicators, a live activity feed polling every 5 seconds, a vulnerability severity breakdown chart, and a top-10 CVE exploitation queue. The dashboard supports three view modes: Live (during active scans), Latest (most recent completed scan), and History (scan selector dropdown). Mode switching is automatic—when the system detects active scans via the /agents/status endpoint, it switches to Live mode, and flips to Latest when scans complete.

Figure 4 shows the main Dashboard view during an active scan. The D3 force-directed network topology graph occupies the upper-left quadrant, with discovered hosts rendered as color-coded nodes (green for low severity, amber for medium, red for critical) and edges representing network connections. The device list on the right enumerates each discovered host with its IP address, hostname, operating system, and maximum CVSS score. Below the topology, the activity feed streams real-time events from the agent pipeline, while the vulnerability severity chart and top-10 exploit queue provide at-a-glance risk prioritization.

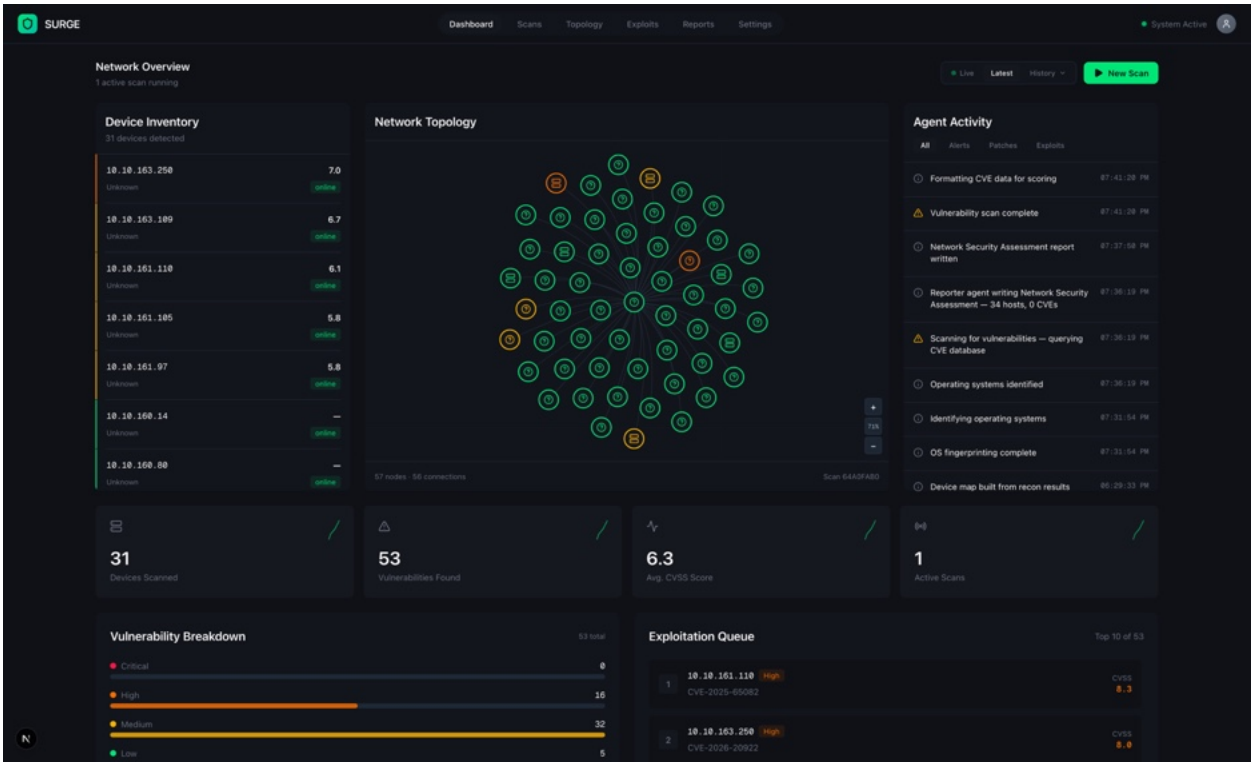

Figure 4. Dashboard main view showing network topology, device list, activity feed, and vulnerability summary.

The Scans view allows operators to launch new scans by specifying a name, target CIDR, and scan type (Quick/Normal/Deep). Running scans display progress bars driven by WebSocket events. The Exploits view presents a searchable, filterable vulnerability table with columns for CVE ID, affected device, CVSS scores (raw and predicted), severity, exploit availability

(public/private/none), and status (queued/in_progress/exploited/patched), with links to the NIST NVD for each CVE.

Figure 5 shows the Exploits view with its filterable vulnerability table. Each row displays the CVE identifier, affected device IP, raw and ML-predicted CVSS scores, severity badge, exploit availability indicator, and current remediation status. The search bar and filter dropdowns allow operators to narrow results by severity level, exploit availability, and status. Selecting a row expands a detail panel with the full vulnerability summary and a direct link to the NIST NVD entry.

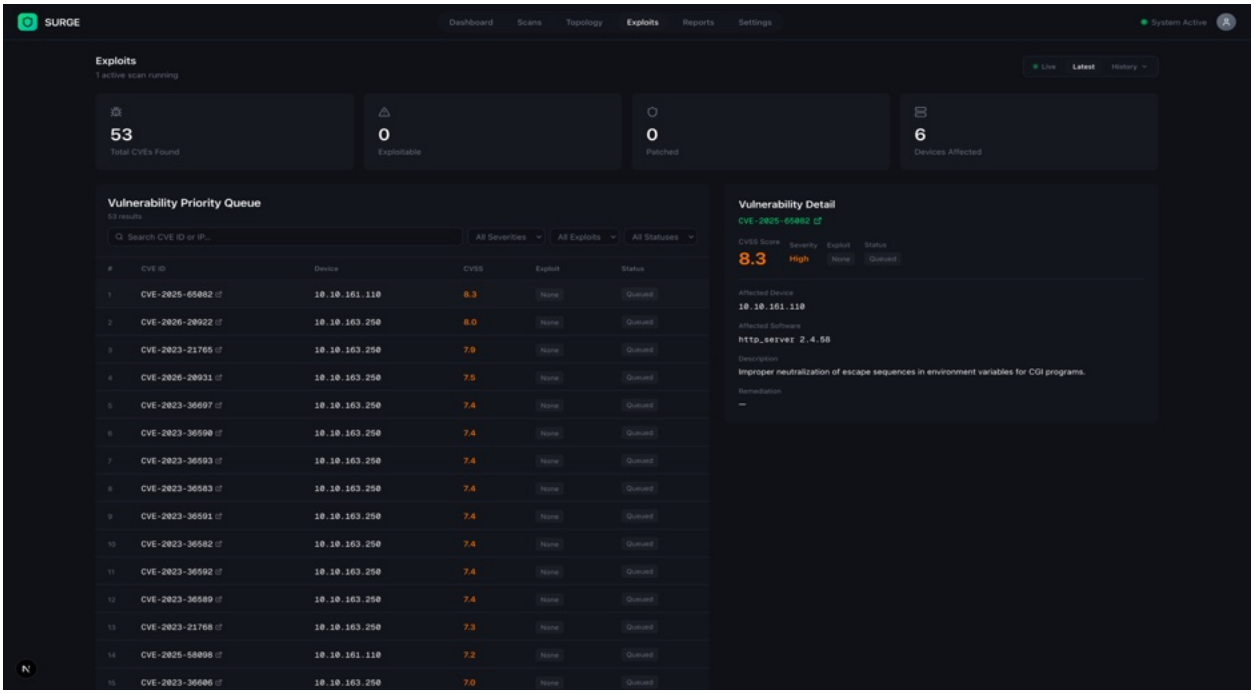

Figure 5. Exploits page showing the filterable CVE priority queue.

The Reports view provides a template picker with four options (Executive, Technical, Public, Final), each targeting a different audience, and renders generated reports with template-specific formatting including financial impact estimation. The Settings view (currently frontend-only, not

persisted to backend) offers configuration panels for agent parameters, network defaults, notification preferences, and appearance themes.

Key frontend dependencies include React 19, d3-force for graph simulation, lucide-react for iconography, and motion for animations. The application communicates with the backend through a centralized API client (lib/api.ts) with typed request/response interfaces.

4.5 Network Infrastructure

The lab network was designed and configured by the cybersecurity team members (Sean and Tauren) to provide a realistic, segmented environment for Surge to scan. The design uses VLAN-based segmentation to create distinct attacker and user network zones, connected through a router-on-a-stick configuration that enables controlled inter-VLAN communication.

Physical Topology: Two Cisco 2960-24TT Layer 2 access switches are connected via an 802.1Q trunk link on interface F0/5, carrying VLANs 10 and 20. Switch 1 hosts attacker devices on VLAN 10 access ports; Switch 2 hosts user devices on VLAN 20 access ports. Switch 1 connects to a Cisco ISR4331 router via a trunk port for router-on-a-stick routing. The router's external interface (g0/0/0, 10.10.1.160) connects to a Cisco ASA 5506-X firewall, which in turn connects to a second ISR4331 router simulating internet connectivity through the HCU Engineering network.

Routing and NAT: The internal router uses subinterfaces g0/0/1.1 (192.168.10.1, VLAN 10) and g0/0/1.2 (192.168.1.1, VLAN 20) for inter-VLAN routing. NAT overload (PAT) is configured on the router's external interface, translating all internal 192.168.0.0/16 addresses to 10.10.1.160 for external communication. The default route (0.0.0.0/0 via 10.10.1.1) directs internet-bound traffic through the firewall.

DHCP: The router provides dynamic IP assignment for both VLANs. The VLAN 10 pool serves the 192.168.10.0/24 network with a default gateway of 192.168.10.1, while the VLAN 20 pool serves 192.168.1.0/24 with a default gateway of 192.168.1.1. Both pools exclude the gateway address from allocation and configure DNS resolution through 8.8.8.8. This mirrors real-world enterprise DHCP deployment and ensures that end devices—whether attacker machines or user workstations—obtain IP configurations automatically without manual assignment.

Verification: The network team validated the infrastructure through a series of tests: intra-VLAN ping (success), inter-VLAN ping via router (success), external interface reachability (10.10.1.160 reachable), routing table verification (default route and connected routes present), NAT operation (65 active translations, 3986 hits, 0 misses), and end-to-end internet connectivity (successful reachability of 8.8.8.8 and multiple public IPs).

Traffic Flow: Inter-VLAN communication follows a four-step path: (1) a device sends traffic to a destination outside its subnet, (2) the packet is forwarded to the device's default gateway on the router subinterface, (3) the router routes the packet to the destination VLAN's subinterface, and (4) the destination switch delivers the packet to the target device. For internet-bound traffic, the path extends through the router's external interface to the ASA firewall, which applies security policies before forwarding allowed traffic to the external router simulating HCU Engineering's network. This layered path ensures that all traffic—including Surge's scan packets—traverses realistic routing and filtering boundaries.

Security Design Considerations: The VLAN segmentation serves a dual purpose: it reduces lateral movement between the attacker and user networks at Layer 2, and it provides the router with controlled inter-VLAN routing that could be further restricted with access control lists in future iterations. The attacker network is isolated from the user network, meaning a

compromised device on one VLAN cannot directly reach devices on the other without traversing the router. The architecture was explicitly designed to support future IDS/IPS integration, where a monitoring device could be placed on the trunk link to inspect inter-VLAN traffic in real time.

Figure 6 presents the lab network topology as configured for Surge's scanning environment. The diagram shows the physical and logical layout including both VLANs, the trunk link between switches, the router-on-a-stick subinterface configuration, the ASA firewall, and the external connectivity path to the HCU Engineering network. Device IP assignments, VLAN memberships, and interface labels are annotated to provide a complete picture of the scan target environment.

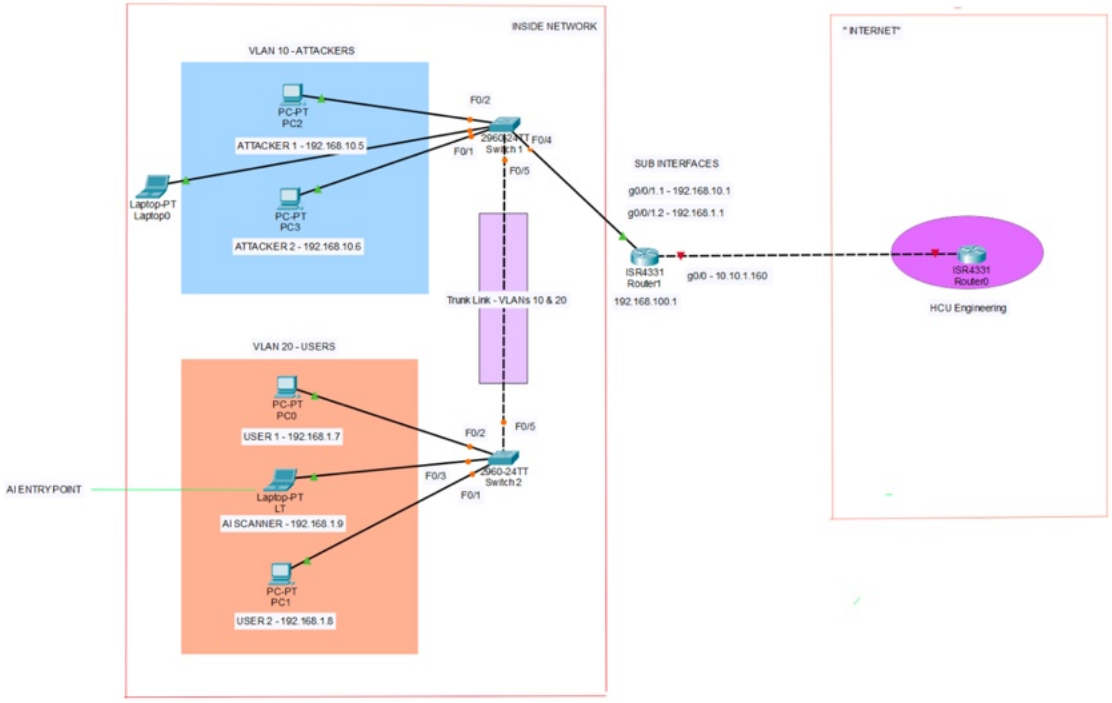

Figure 6. Lab network topology with VLAN segmentation, routing, and firewall configuration.

4.6 Governance and Security Layer

Surge implements a governance layer (governance/sanitization.py) that controls what actions the AI agents can take. This is critical because the system grants LLM agents the ability to invoke real network scanning tools. The sanitization module enforces a tiered whitelist system:

Low Tier (Quick Scan): Host discovery only (-sn), basic timing flags (-T0 through -T4), no port scanning, no service detection.

Medium Tier (Normal/Stealth Scan): Port and service enumeration (-sS, -sV, -O), light scripting (-sC), version detection, maximum 4096 ports, 20-minute timeout.

High Tier (Deep Scan): Full reconnaissance (-A, --script vuln), all 65535 ports, 90-minute timeout, version tracing.

All tiers block shell metacharacters (semicolons, ampersands, pipes, backticks, dollar signs) to prevent command injection. Port expressions are validated against a regex pattern. Script arguments are parsed and validated. The system automatically appends the -oX flag for XML output if not present.

Figure 7 summarizes the three-tier governance model as a visual reference. Each tier is represented as a column showing allowed nmap flags, port limits, timeout budgets, and blocked operations. The diagram makes clear the progressive expansion of scanning capability from the restrictive Low tier (host discovery only) through the Medium tier (service enumeration with

bounded scope) to the High tier (full vulnerability scanning), while all tiers uniformly block shell metacharacters and enforce output formatting.

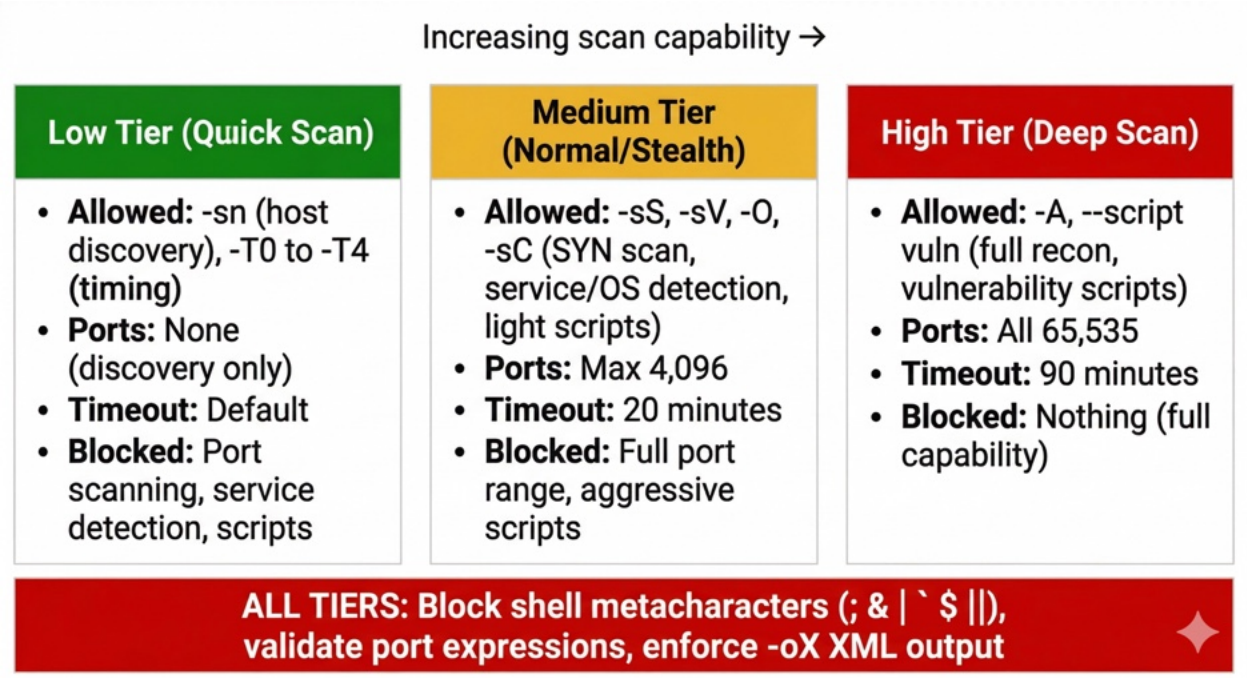

Figure 7. Tiered governance model for nmap flag sanitization.

4.7 Machine Learning Model

The CVSS prediction model is an XGBoost gradient-boosted regressor trained to predict CVSS scores from vulnerability metadata. The feature engineering pipeline produces 137+ features from each CVE entry:

Six categorical CVSS fields (access_authentication, access_complexity, access_vector, impact_availability, impact_confidentiality, impact_integrity) are one-hot encoded using a pre-fitted scikit-learn OneHotEncoder (model/ohe_encoder.pkl). CWE (Common Weakness Enumeration) names are vectorized using a pre-fitted TF-IDF encoder (model/tfidf_encoder.pkl).

Vulnerability summaries are embedded using sentence-transformers (all-MiniLM-L6-v2) to capture semantic similarity between vulnerability descriptions.

The model and encoders are serialized in the model/ directory and loaded at inference time by the cvss_scoring agent. Feature alignment ensures that the input DataFrame matches the model's expected schema, filling missing columns with zeros. The model outputs predicted CVSS scores on a 0–10 scale, which are displayed alongside raw NVD scores in the dashboard.

Figure 8 illustrates the ML feature engineering pipeline that transforms raw CVE data into the 137-feature vector consumed by the XGBoost regressor. Raw vulnerability entries enter from the left; categorical CVSS fields are one-hot encoded, CWE names are TF-IDF vectorized, and vulnerability summaries are embedded via sentence-BERT (all-MiniLM-L6-v2). These feature groups are concatenated, aligned to the model's expected schema, and passed to the XGBoost regressor, which outputs a predicted CVSS score on a 0–10 scale.

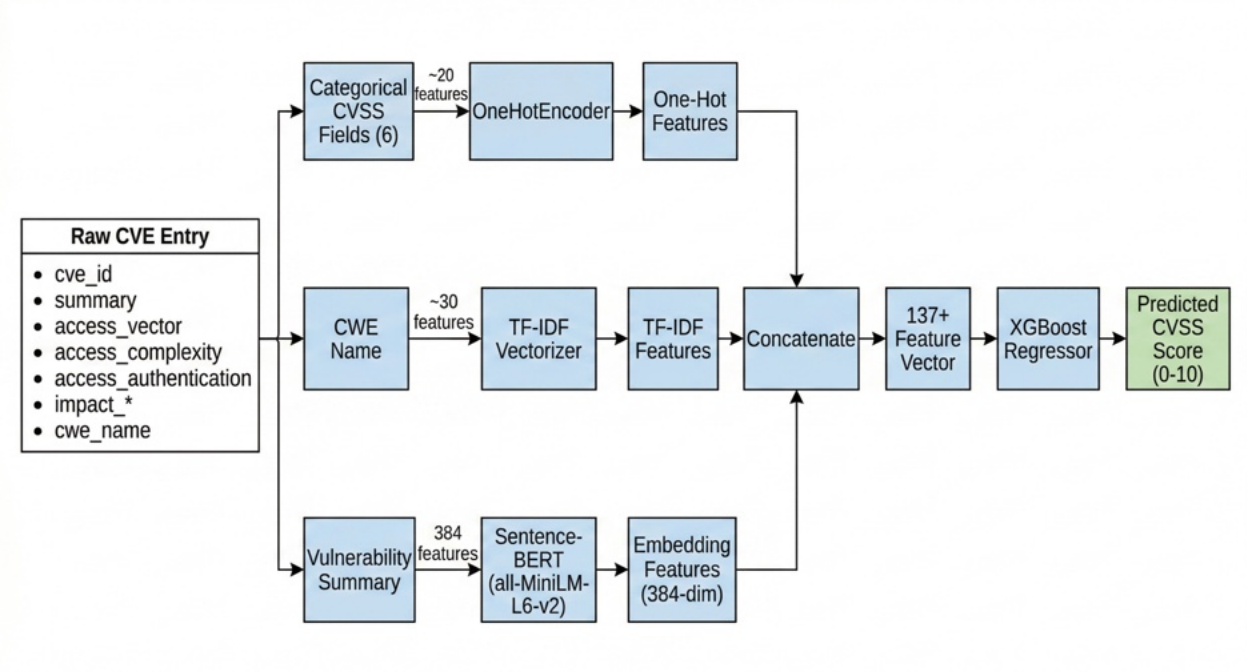

Figure 8. Machine learning feature engineering pipeline for CVSS score prediction.

4.8 Codes and Standards

The following codes, standards, and frameworks are relevant to the Surge platform's design:

MITRE ATT&CK Framework: The agent pipeline's reconnaissance and enumeration phases map to the Discovery and Credential Access tactics. Vulnerability findings are correlated with ATT&CK techniques for contextual risk assessment.

Common Vulnerability Scoring System (CVSS) v2.0/v3.x: The industry-standard vulnerability severity scoring system used as both raw input (from NVD) and prediction target (for the XGBoost model).

Common Vulnerabilities and Exposures (CVE): The standard identifier system for publicly disclosed vulnerabilities, sourced from the CIRCL API and linked to the NIST National Vulnerability Database.

Common Platform Enumeration (CPE): The structured naming scheme for IT systems, used by the OS Fingerprinting agent to correlate discovered software with known vulnerabilities.

Common Weakness Enumeration (CWE): The community-developed list of software weakness types, stored per vulnerability and used as a feature in the ML model.

IEEE 802.1Q: The VLAN tagging standard implemented on the trunk links between switches for network segmentation.

NIST SP 800-115 (Technical Guide to Information Security Testing and Assessment): The NIST guidelines for penetration testing methodology that inform the agent pipeline's phased approach.

4.9 Testing and Validation

Validation of the Surge platform was conducted in two phases: controlled lab testing and live network testing, with results reviewed by both the development team and the project advisor.

Lab Network Testing: The primary testing environment was the VLAN-segmented lab network described in Section 4.5. The software development team (Brianna and Jonathan) ran full scan cycles across all four scan types (quick, normal, deep, and stealth) targeting both VLAN 10 and VLAN 20 subnets. The network team members (Sean and Taurean) independently reviewed the generated scan reports and vulnerability findings, confirming that the discovered hosts, open ports, operating system identifications, and CVE correlations accurately reflected the known state of the lab devices. This cross-team validation was performed multiple times throughout development as new agents were added to the pipeline, ensuring that each stage produced correct and consistent output.

HCU Engineering Network Testing: To validate Surge's performance beyond the controlled lab environment, the team obtained permission to run scans against the HCU Engineering department network. This live network presented a more complex and realistic target environment with a wider variety of devices, operating systems, and service configurations than the lab setup. Scan results and generated reports from the Engineering network were reviewed by Dr. Stephen Lyford, the project advisor, who confirmed that the scans were accurate and the reports reflected the actual state of the network. This external validation was critical for demonstrating that Surge generalizes beyond the specific lab topology it was developed against.

Network Infrastructure Verification: The underlying lab network was independently validated by the network team through a structured series of connectivity and routing tests. These included intra-VLAN ping tests (confirming Layer 2 isolation), inter-VLAN ping tests (confirming router-on-a-stick functionality), external interface reachability checks, routing table verification (default

route and connected routes present), NAT operation validation (65 active translations with 3,986 hits and zero misses), and end-to-end internet connectivity tests reaching external addresses including 8.8.8.8. All tests passed successfully, confirming that the network infrastructure was correctly configured to support Surge's scanning operations.

5. CONSTRAINTS

This section identifies how the Surge platform's design addresses the following constraint categories.

a. Public Health, Safety and Welfare Factors

Surge is a software platform that does not directly interact with the physical environment or human health. However, the system's security governance layer is designed to prevent the platform from being used to disrupt network services that could indirectly affect safety-critical systems. The tiered scan sanitization ensures that even at the highest scan tier, nmap operations are time-bounded and flag-restricted. The system is deployed on an isolated lab network, ensuring that any scanning activity cannot reach production HCU infrastructure or external networks.

b. Global Factors

The platform is designed for use within a single university's engineering lab. Global considerations include the use of the CIRCL CVE API (hosted in Luxembourg) for live vulnerability data, which introduces a dependency on international internet connectivity. The MITRE ATT&CK framework and CVSS scoring standards used by the platform are internationally recognized, ensuring that Surge's outputs are interpretable by security professionals worldwide.

c. Cultural Factors

This constraint does not directly apply to the Surge platform. The system's interface is in English, consistent with the technical and academic context at HCU.

d. Social Factors

Surge contributes to cybersecurity education by lowering the barrier to entry for hands-on penetration testing experience. By automating the complex, multi-tool workflow that traditionally requires years of expertise, the platform enables students to focus on understanding vulnerability analysis and risk assessment concepts rather than memorizing tool syntax. The multi-audience report generation feature also teaches students to communicate security findings to different stakeholders—a critical professional skill.

e. Environmental Factors

The primary environmental consideration is the energy consumption of the dual RTX 8000 GPUs, each with a 260W TDP. The system is deployed on shared university infrastructure rather than dedicated hardware, amortizing its environmental footprint. The decision to use locally-hosted LLMs rather than cloud-only inference reduces network traffic and avoids the energy overhead of cloud data center cooling, though it concentrates GPU power consumption on the local server.

f. Economic Factors

The project was developed within an academic context with minimal direct costs. The primary hardware (dual RTX 8000 GPUs, engineering server) and all Cisco networking equipment (2960 switches, ISR4331 routers, ASA 5506-X firewall) were provided by HCU's Engineering department. No out-of-pocket hardware purchases were made. Software costs are minimized through the use of open-source technologies (FastAPI, LangGraph, Next.js, PostgreSQL, Ollama, nmap) and open-weight LLMs (Qwen3, GPT-OSS). The only recurring cost is OpenRouter API usage for the analysis-tier LLM (GLM-5), funded by a $50.00 credit allocation

in mid-January 2026. As of April 9, 2026, only $3.43 of that allocation has been consumed, leaving $46.57 remaining. The frontend is deployed on Vercel's free tier with no domain costs, using the default Vercel subdomain.

The total project budget was effectively $50.00, consisting entirely of prepaid API credits. All other infrastructure was either university-provided or free-tier cloud services, making the project highly cost-effective.

Additional Applicable Constraints

Accessibility: The dashboard follows standard web accessibility patterns through semantic HTML and keyboard-navigable components, though a formal WCAG audit has not been conducted.

Ethical Considerations: Surge is explicitly scoped for authorized lab use. The governance layer prevents scanning of unauthorized targets. All vulnerability data is sourced from public CVE databases. The system does not perform actual exploitation—it discovers and reports vulnerabilities without attempting to leverage them.

Legal Considerations: Network scanning is legal only with explicit authorization from the network owner. Surge's deployment within HCU's lab environment, with departmental approval, satisfies this requirement. The system should not be deployed against networks without written authorization.

Schedule: The project timeline of August 2025 to April 2026 necessitated scope management decisions including the deprioritization of fine-tuning, the adoption of a PRODUCIBLE/BUILDABLE/CUT/DEFER triage framework for dashboard features, and the decision to use pre-trained models rather than training from scratch.

6. MATERIAL LEARNED OUTSIDE OF THE CLASSROOM

The Surge project required the team to acquire significant knowledge beyond the scope of the undergraduate curriculum. The following topics were self-taught through documentation, technical manuals, research papers, and online resources:

LLM Agent Orchestration (LangGraph/LangChain): The design and implementation of multi-agent AI systems using LangGraph's StateGraph abstraction, including state management, conditional routing, fan-out/fan-in parallelism, and the integration of LLM reasoning with external tool execution. This required study of the LangGraph documentation, LangChain's tool-calling interfaces, and research on agentic AI system design patterns.

GPU Computing and LLM Inference Optimization: Understanding CUDA compute capabilities, VRAM requirements for quantized models, the differences between quantization formats (GGUF Q4_K_M vs. MXFP4 vs. AWQ), Ollama's model serving architecture, and GPU isolation via CUDA_VISIBLE_DEVICES. This also included evaluating and benchmarking multiple inference engines (vLLM, Ollama, llama.cpp).

Network Penetration Testing Methodology: The phases of a penetration test (reconnaissance, enumeration, vulnerability assessment, reporting), nmap's scanning modes and flag semantics, OS fingerprinting techniques, CVE/CVSS/CPE/CWE ecosystems, and the MITRE ATT&CK framework. While some networking concepts were covered in coursework, the depth of nmap configuration and vulnerability correlation required was well beyond classroom coverage.

VLAN Segmentation and Router-on-a-Stick Configuration: Configuring Cisco 2960 switches with 802.1Q trunk links, ISR4331 routers with subinterfaces for inter-VLAN routing, ASA

5506-X firewall policies, NAT/PAT configuration, and DHCP pool management. This was learned through Cisco documentation and Packet Tracer simulations.

XGBoost for CVSS Prediction: Training gradient-boosted regression models on vulnerability data, feature engineering with one-hot encoding, TF-IDF vectorization, and sentence-BERT embeddings. Model serialization, feature schema alignment, and inference pipeline integration.

FastAPI with Async Patterns: Building asynchronous REST APIs with WebSocket support, managing background tasks with asyncio.to_thread, using ContextVar for thread-safe state propagation, and integrating SQLAlchemy 2.0's async session management.

Tailscale VPN Networking: Configuring Tailscale for secure remote access to the university server, enabling the team to develop and test against the GPU-equipped server from off-campus locations.

Prompt Engineering for Agentic Systems: Designing effective system prompts for each of the eight agents in the pipeline required extensive experimentation with prompt structure, output formatting constraints, and chain-of-thought reasoning. Each agent's prompt was iteratively refined to produce consistent, parseable output while maintaining analytical depth—a skill not covered in any coursework.

D3.js Force-Directed Graph Visualization: Building the interactive network topology visualization on the dashboard required learning D3's force simulation API, including node/link data binding, collision detection, zoom/pan behavior, and dynamic graph updates as scan results stream in via WebSocket. While general web development was familiar, D3's data-driven DOM manipulation paradigm required dedicated self-study.

Ubuntu Server Administration for GPU Workloads: Configuring the university's Ubuntu server to host both local LLMs simultaneously required learning NVIDIA driver installation and management, CUDA toolkit configuration, GPU memory allocation strategies, Ollama service configuration for multi-model serving, and systemd service management—all topics outside the standard curriculum.

7. COST ANALYSIS

The Surge platform was developed primarily using university-provided infrastructure and open-source software, minimizing direct costs.

The table below summarizes all project costs. The team allocated a $50.00 budget for OpenRouter API credits in mid-January 2026; as of April 9, 2026, only $3.43 has been consumed across all development and testing.

Item	Cost	Notes
HCU Engineering Server (dual RTX 8000)	$0	Provided by HCU Engineering department
Cisco 2960-24TT Switches (x2)	$0	Provided by department
Cisco ISR4331 Routers (x2)	$0	Provided by department
Cisco ASA 5506-X Firewall	$0	Provided by department
OpenRouter API Credits	$3.43	$50 allocated Jan 2026; $46.57 remaining as of Apr 9
Vercel Hosting (Free tier)	$0	Free tier sufficient for dashboard
Domain (hcu-surge.vercel.app)	$0	Vercel subdomain
Software Licenses	$0	All open-source
Total Project Cost	**$3.43**	

8. FUTURE EXPANSION OF PROJECT

Given additional time and resources, the following improvements and extensions could be made to the Surge platform:

Authentication and Authorization (2–3 weeks): The dashboard currently has no authentication layer. Implementing JWT-based authentication with role-based access control (operator vs. admin vs. viewer) would be necessary for any deployment beyond a single-user lab environment.

MITRE ATT&CK Visualization (1–2 weeks): While the backend correlates findings with ATT&CK techniques, the frontend does not yet display this mapping. Adding an ATT&CK Navigator-style matrix view to the dashboard would significantly enhance the educational value of scan results.

Fine-Tuned Domain Models (4–6 weeks): Training or fine-tuning a local model on cybersecurity-specific data (vulnerability descriptions, nmap output interpretation, risk assessment reports) would improve agent accuracy and reduce dependence on cloud LLM providers.

Automated Exploitation (6–8 weeks): Extending the pipeline with an Exploitation agent that can safely test identified vulnerabilities in the lab environment, moving from assessment to validation. This would require significant additional governance controls.

ACL and Firewall Rule Implementation (2–3 weeks): The network team identified implementing ACLs between VLANs and configuring firewall NAT and filtering rules as planned next phases beyond the current network design.

Monorepo Consolidation (1 week): The two current repositories (backend for the Python/LangGraph backend and frontend for the Next.js frontend) are being consolidated into a single monorepo for simplified deployment and version management.

Formal Test Suite (2–4 weeks): The project currently lacks a formal test suite. Adding unit tests for the agent pipeline, integration tests for the API endpoints, and end-to-end tests for the scan workflow would improve reliability and enable confident refactoring.

Natural Language Threat Query Interface (3–4 weeks): Implementing a conversational interface that allows operators to query scan results using plain English—for example, "show me all critical vulnerabilities on Windows machines in VLAN 10" or "which hosts have unpatched services older than 90 days?" This would use a retrieval-augmented generation (RAG) pipeline over the PostgreSQL database, letting the local LLM translate natural language into structured queries and return contextual, conversational answers rather than raw table data.

Automated Remediation Playbook Generation (4–6 weeks): Extending the reporter agent to not only identify vulnerabilities but also generate step-by-step remediation scripts tailored to each discovered host. For example, if a Linux machine is found running an outdated OpenSSH version, the system could produce a ready-to-execute shell script with the appropriate package manager commands, configuration hardening steps, and post-remediation verification checks—transforming Surge from a diagnostic tool into an actionable security assistant.

Live Threat Intelligence Feed Integration (2–3 weeks): Connecting Surge to real-time threat intelligence feeds (such as CISA's Known Exploited Vulnerabilities catalog and abuse.ch threat data) would allow the platform to flag not just known CVEs but actively exploited vulnerabilities in the wild. This would add an urgency score layer on top of the existing CVSS predictions,

helping operators prioritize patching based on real-world exploitation activity rather than theoretical severity alone.

Network Digital Twin Simulation (6–10 weeks): Building a virtual replica of the scanned network using the topology and service data Surge already collects. By feeding discovered hosts, operating systems, open ports, and vulnerability profiles into a containerized simulation environment (using tools like GNS3 or Containerlab), the platform could allow operators to safely rehearse attack scenarios, test remediation strategies, and predict the blast radius of a compromised host—all without touching the live network. Combined with the agent pipeline, this would enable a closed-loop "scan → simulate → remediate → verify" workflow that transforms Surge from a passive assessment tool into a full cyber range orchestrator.

REFERENCES

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

[2] NIST, "National Vulnerability Database," https://nvd.nist.gov/

[3] CIRCL, "CVE Search API," https://cve.circl.lu/

[4] LangGraph Documentation, https://langchain-ai.github.io/langgraph/

[5] FastAPI Documentation, https://fastapi.tiangolo.com/

[6] Ollama Documentation, https://ollama.ai/

[7] XGBoost Documentation, https://xgboost.readthedocs.io/

[8] Cisco, "Router-on-a-Stick Configuration Guide"

[9] NIST SP 800-115, "Technical Guide to Information Security Testing and Assessment"

[10] Nmap Project, "Nmap Reference Guide," https://nmap.org/book/man.html

[11] Next.js Documentation, https://nextjs.org/docs

[12] OpenRouter API Documentation, https://openrouter.ai/docs

[13] D3.js Documentation, https://d3js.org/

[14] PostgreSQL Documentation, https://www.postgresql.org/docs/

[15] SQLAlchemy 2.0 Documentation, https://docs.sqlalchemy.org/