

Aegis Wireless - Capstone Project

Documentation

Knight Foundation School of Computing and Information Sciences (KFSCIS) - Florida International University
Course: CIS 4951 Capstone 2 Spring 2026

Instructor: Massoud Sadjadi | Version: 2026 Spring | License: MIT

Poster: 36" x 48" portrait | Videos: Intro & Demo | Presentation: 10-15 min

ACM Student Outcomes

01	Problem Analysis & Requirements	Analyze user/stakeholder needs; define constraints and success criteria.
02	System Design & Implementation	Design solutions using appropriate abstractions, patterns, and tools; implement maintainable code
03	Experimentation, Testing & Evaluation	Collect evidence; interpret results to improve the system.
04	Communication & Collaboration	Communicate effectively with technical and non-technical stakeholders; collaborate in teams.
05	Professional, Ethical, Legal & Societal	Recognize responsibilities; address ethics, security, privacy, accessibility, and sustainability
06	Threat Modeling, Risk & Assurance	Discipline-specific depth outcome; addressed in dedicated sections below.

Outcomes Checklist

01	Problem & Requirements	Complete ▾
02	System Overview & Architecture	Complete ▾
03	Testing & Evaluation	Complete ▾
04	Executive Summary Future Work	Complete ▾
05	Security, Privacy, Accessibility & Compliance	Complete ▾
06	Threat Modeling, Risk & Assurance	Complete ▾

Executive Summary

As a result of the rise in remote and hybrid work starting in 2020, the security landscape has had to adapt quite rapidly these past years as a result. The unique challenge this presents is that enterprise security cannot write policies to control the security of the networks that their remote employees utilize.

Therefore, remote or hybrid work generates a high level of risk.

The simple solution would be to reduce or altogether eliminate working from home or limiting remote or hybrid work to exclusively designated zones, however that simply is not a practical solution. As remote work is becoming an essential towards hiring and maintaining top talent, any policy preventing hires from working where they want to work runs the risk of crippling morale, policy violation, or losing talent all together. From this point the question becomes:

“How can enterprises deploy remote work without incurring unnecessary risk from 3rd party networks?”

This is where Aegis Wireless comes in. A windows-based application for endpoint security that runs silently in the background preventing users to connect to unsafe networks. When a connection to a new network is made Aegis Wireless runs a series of scans to verify the risk that the network poses. From there a decision is made, allow connection or block all together. The user will be prevented access to that network, and provided with a pop up notification explaining that the network is too dangerous for the device. Employees maintain the freedom to work from anywhere without, while still giving companies policy control over the security of networks they choose to connect to.

Simply, Aegis Wireless gives Admins the power to administer policies and control in regards to networks they otherwise would have no control over. Aegis Wireless allows companies to deploy remote work with an additional layer of security, ensuring their continued ability to attract and keep the highest level of talent.

2. Problem & Requirements (01)

With more people working remotely or in hybrid setups, there's a big security gap that companies can't fully control. Employees are constantly working from places like coffee shops, airports, and libraries, connecting to public WiFi that isn't protected by company security. At the same time, companies can't just get rid of remote work because they would lose employees. So the main problem becomes: how can companies allow remote work while still keeping their data secure?

Stakeholders:

1. Employees who rely on public WiFi to get work done
2. IT security administrators who manage and enforce security policies
3. Company leadership concerned about data breaches, costs, and compliance

Functional Requirements:

- Scan nearby WiFi networks and collect details like name, signal strength, encryption type, and identifiers
- Check for open ports using multi-threaded scanning
- Assign a risk score based on things like encryption, hidden networks, risky ports, signal issues, and blacklist status
- Let users manage a blacklist of unsafe networks
- Automatically disconnect from dangerous networks
- Log all activity locally for tracking and auditing
- Run in the background with notifications
- Provide a simple dashboard to view results
- Allow settings to be adjusted through configuration files

Non-Functional Requirements:

- Everything runs locally (no cloud or external dependencies)
- No data is sent or stored online
- Fast scans (no more than a few seconds)
- Should work mainly on Windows, with some support for Linux
- Use minimal system resources
- Should not require admin permissions to install or run

Constraints:

- Built using only Python's standard library (no external compiled tools)
- Must run on Windows 10 and 11
- Cannot perform offensive or intrusive actions (only basic scanning and observation)
- All data must stay on the user's device

Success Criteria:

- Correctly identify secure vs insecure networks
- Detect risky configurations like open networks or dangerous ports
- Automatically disconnect from unsafe networks
- Log all actions properly
- Run quietly in the background with notifications
- Start automatically when the system turns on

Prioritized Backlog:

- WiFi scanner
- Port scanner
- Risk scoring system
- Blacklist feature
- Auto-disconnect system
- Logging system
- Command-line interface (CLI)
- Web dashboard

- Background system tray app
 - Notifications
 - Startup integration
 - VPN detection
 - Config settings
 - Full network scan feature
 - Easy launch scripts
-

3. System Overview & Architecture (02)

Aegis Wireless is built in Python and organized into different parts so everything isn't all mixed together. There's a scanner that handles WiFi detection and port scanning, a core section that calculates risk and keeps track of blacklisted networks, and a network part that handles alerts and disconnecting from unsafe WiFi. There's also a logging part that records everything the system does, and a config file where all the settings are stored. There are three ways to use the program: through the command line, a web dashboard, or a background version that runs in the system tray without the user really needing to interact with it. The flow of the program is simple: it scans networks, checks how risky they are, takes action if needed (like disconnecting), and then logs everything. All of the data stays on the device, including logs and the blacklist.

The whole project is built using Python 3.12. Most of the core features use Python's built-in tools, like running system commands for WiFi scanning and using sockets for port scanning. It also uses multi-threading to make scans faster. For the interface, Flask is used to create a simple web dashboard, and system tray tools are used to run the app in the background and send notifications. JSON is used to store settings and data. There's no cloud, no database, and no outside dependencies-everything runs locally on the user's computer.

4. Discipline Specific Depth (06)

The main things being protected in this project are employee credentials, company data being sent over networks, and the overall safety of the device. The biggest risks come from people using public WiFi, where attackers can easily take advantage. These attackers can be random people on the same network trying to spy on traffic, or more advanced ones setting up fake WiFi networks (like "evil twins") to trick users into connecting. There are also risks from exposed services, like open ports (for example Telnet, RDP, or SMB) that attackers can use as entry points.

There are a few main areas with risk:

1. The WiFi connection itself, especially if it's open or weakly secured
2. Open ports on the device that expose services to others on the network
3. The user's decision to trust and connect to a network without verifying it

To handle these risks, the system follows a structure similar to STRIDE:

1. Spoofing: handled by tracking blacklisted networks and detecting weird signal behavior (possible fake networks)
2. Tampering: reduced by requiring stronger encryption like WPA2 or WPA3
3. Information disclosure: prevented by disconnecting from unsafe networks before data can be exposed
4. Denial of Service: (outside the scope of this tool)
5. Privilege escalation: reduced by flagging risky open ports that could be used to gain access

Controls & Protection

The system checks encryption first and penalizes weaker networks like Open, WEP, or WPA1, making sure only safer options are trusted. If a network is considered dangerous, the tool can automatically disconnect the device without the user needing to do anything.

There's also a blacklist feature, so known unsafe networks can be flagged immediately. Another check looks at signal strength-if it's unusually strong, it could be a fake access point placed close to the user.

Everything the system does is logged locally with timestamps, both in readable logs and structured files, so it can be reviewed later if needed.

Security Testing

The tool was tested across different scenarios to make sure it works as expected. It correctly identifies unsafe networks like open or WEP networks, flags risky ports, detects hidden networks, and warns about unusual signal behavior. It also successfully disconnects from dangerous networks when needed.

All versions of the system (command line, dashboard, and background app) were tested and give the same results, which shows everything is consistent. The project keeps dependencies minimal and avoids unnecessary risks by running locally, limiting access to localhost, and safely handling user input to prevent common issues like injection or scripting attacks.

5. Implementation Notes (02)

The project is organized under the Aegis_Wireless directory. At the root level, it includes main.py, dashboard.py, tray_agent.py, and aegis_launcher.bat. The codebase is divided into subdirectories: scanner/, core/, network/, api/, and config/, each responsible for a specific part of the system. Every module is structured as a Python package and includes an `__init__.py` file.

Coding Conventions

The codebase follows a simple and practical style. It primarily uses functions and dictionaries instead of

classes where possible, keeping the structure straightforward. Comments are written in a more casual, student-friendly tone rather than formal documentation style. Type hints and dataclasses are not used, which helps keep the code easier to read and modify for beginners.

Notable Implementation Patterns

- `wifi_scan.py` uses `subprocess.run()` to execute system commands and regular expressions to convert raw output into structured data
- `port_probe.py` uses a `ThreadPoolExecutor` with multiple workers to scan ports in parallel with a configurable timeout
- `engine.py` uses a penalty-based scoring system that starts at 100 and subtracts points based on detected risks
- `enforcement.py` supports both interactive and automatic disconnection modes depending on configuration
- `telemetry.py` uses built-in logging for text output and JSON formatting for structured logs

The system uses system commands instead of directly interacting with the Windows WLAN API. This approach simplifies development and avoids the need for compiled dependencies, but it does limit access to real-time network event handling.

6. Testing & Evaluation (03)

Test 1 - Open WiFi Detection:

- Input: open network (no encryption)
- Expected: score ≤ 60 (MODERATE)
- Result: PASS

Test 2 - WPA2 Network:

- Input: WPA2-encrypted network
- Expected: score = 100 (SAFE)
- Result: PASS

Test 3 - Blacklisted Network:

- Input: previously flagged SSID
- Expected: -50 point penalty (DANGEROUS)
- Result: PASS

Test 4 - Dangerous Port Detection:

- Input: port 23 (Telnet) open
- Expected: -8 points per port
- Result: PASS

Test 5 - Auto-Disconnect:

- Input: DANGEROUS risk level with auto_block enabled
- Expected: WiFi disconnected using system command
- Result: PASS

Test 6 - Local Logging:

- Input: any scan action
- Expected: .log and .json files created in logs directory with timestamps
- Result: PASS

Test 7 - VPN Detection:

- Input: system with VPN adapter present
- Expected: status shown as Active
- Result: PASS

Test 8 - Hidden SSID Flagging:

- Input: network with hidden name
- Expected: -10 point penalty
- Result: PASS

All test scenarios passed successfully. The scoring system was validated using known network configurations with expected penalty values. Integration testing confirmed that all system interfaces (command line, web dashboard, and background agent) use the same backend logic and produce consistent results.

7. Security, Privacy, Accessibility & Compliance (05)

Security:

Aegis Wireless is designed strictly as a defensive tool. It only observes publicly broadcast WiFi signals and performs TCP connection attempts to localhost or explicitly defined targets. It does not intercept traffic, inject packets, or perform any offensive actions. All data is stored locally, with no cloud usage, telemetry, or external API calls. Blacklist and log files remain fully under user control and can be deleted at any time.

Privacy:

No personally identifiable information is collected. Network details such as SSIDs and BSSIDs are publicly broadcast by access points and are treated as metadata rather than private data. Logs only contain technical information like network name, encryption type, signal strength, and port states. All data remains stored locally on the user's device.

Ethical Considerations:

A legal disclaimer is presented at startup to inform users that unauthorized network or device scanning may violate laws such as the Computer Fraud and Abuse Act (CFAA). Port scanning is limited to localhost by default to reduce the risk of misuse. Automatic disconnection is not enabled unless explicitly configured by the user.

Accessibility:

The command-line interface uses both color and text labels to ensure information is understandable without relying only on color. The web dashboard uses high-contrast text and readable font sizes to improve visibility.

Compliance:

The tool follows established security guidelines by enforcing modern encryption standards and validating network authentication types. Its design aligns with recognized practices for securing wireless networks and evaluating risk.

8. Deployment & Operations

Python 3.10 or higher on Windows 10 or 11 is required. The core command-line tool (main.py) uses only Python's standard library, so no additional installations are needed.

The web dashboard requires Flask, and the system tray version requires pystray, Pillow, and plyer. These can be installed using pip if those features are needed.

An install.bat script is included to automate dependency setup and create launcher scripts. For end-user deployment, an IT administrator runs:

- `py tray_agent.py --install`

This registers the application in the Windows Registry so it runs automatically at startup using pythonw.exe, which allows it to run in the background without opening a console window.

Once installed, the system tray agent starts on boot, scans networks at regular intervals, and can automatically disconnect from unsafe networks. No administrator privileges are required for installation or normal use.

The `aegis_launcher.bat` file provides a simple menu-based interface that allows developers or administrators to choose between running the command-line tool, web dashboard, or background system tray version.

9. Limitations and Future Work

Limitations:

- The format of WiFi scan results from system commands can vary depending on the Windows version and system language, which may cause inconsistencies in parsing on non-English systems
- Port scanning is limited to basic TCP connection attempts and does not support UDP scanning or more advanced techniques
- VPN detection is based on adapter naming patterns rather than a direct system API, so some configurations may not be detected
- Detection of fake access points (evil twins) is limited, since it does not yet compare MAC addresses for verification
- Notification behavior may differ between Windows 10 and 11 due to library inconsistencies
- The tool is not currently packaged for large-scale enterprise deployment through systems like Intune or SCCM

Future Work:

- Track known networks and flag changes in MAC address to help detect potential fake access points
 - Establish a baseline for normal network behavior and flag unusual changes as possible threats
 - Add deeper traffic analysis using tools like packet inspection to detect more advanced attacks
-

10. References

- [1] IBM Security. *Cost of a Data Breach Report 2024*. IBM Corporation, 2024.
- [2] Pew Research Center. *How Working From Home Has Changed*. 2023.
- [3] Microsoft. *Windows WLAN API Documentation*. Microsoft Docs, 2024.

- [4] IEEE 802.11-2020. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*.
- [5] NIST SP 800-153. *Guidelines for Securing Wireless Local Area Networks (WLANs)*.
- [6] OWASP. *Wireless Security Testing Guide*. OWASP Foundation, 2024.
- [7] Python Software Foundation. *socket - Low-level networking interface*. Python 3.12 Documentation.
- [8] Python Software Foundation. *subprocess - Subprocess management*. Python 3.12 Documentation.

Appendices (Evidence and How-to)

A. Installation Guide

1. Verify that Python 3.10 or higher is installed by running `python --version` or `py --version` in Command Prompt. If not installed, download it and make sure "Add python.exe to PATH" is selected during setup.
2. Download or copy the Aegis_Wireless project folder to Desktop.
3. Run `install.bat` or `aegis_launcher.bat` and select the option to install required dependencies (Flask, pystray, Pillow, plyer).
4. Launch the tool using one of the following:

- py `main.py` for the command-line interface
- py `dashboard.py` for the web dashboard
- py `tray_agent.py` for the background system tray agent

B. User Manual

CLI Tool (`main.py`):

The command-line interface provides the following options:

1. Scan WiFi networks (displays SSID, signal strength, encryption, and channel)
2. Scan ports on a target IP
3. Analyze network risk (assigns a score and classification)
4. Manage blacklist (add, remove, or view networks)
5. View scan logs
6. Check VPN status
7. Run a full network audit
8. Exit

Web Dashboard (`dashboard.py`):

Open a browser and go to: `http://127.0.0.1:5000`

Run a full network audit to view scan results, risk levels, and manage the blacklist.

System Tray Agent (tray_agent.py):

Runs in the background. Right-click the tray icon to access:

- Scan now
- Pause scanning
- Toggle auto-block
- Enable/disable startup
- Open logs
- Exit

C. Admin / Operations Guide

To deploy on a system:

1. Copy the Aegis_Wireless folder to the target machine
2. Install dependencies:
`py -m pip install flask pystray Pillow plyer`
3. Register the application for startup:
`py tray_agent.py --install`

The tool will run automatically at startup in the background.

Configuration settings can be edited in config/settings.json (scan timing, threads, ports).

The blacklist can be managed through the interface or in config/blacklist.json.

Logs are stored in the logs/ directory as .log and .json files.

To uninstall:

`py tray_agent.py --uninstall`
Then delete the project folder.

D. Internal API Reference

The API runs locally at:

`http://127.0.0.1:5000`

Endpoints:

GET /api/scan/wifi - runs WiFi scan

GET /api/scan/ports - scans ports

GET /api/audit - runs full audit

GET /api/vpn - returns VPN status

GET /api/blacklist - returns blacklist entries

POST /api/blacklist/add - adds a network

POST /api/blacklist/remove - removes a network

All responses are in JSON. The API is only accessible locally and does not require authentication.

E. Poster & Presentation

The project includes a 36" x 48" horizontal poster titled "Aegis Wireless" covering the problem, threat landscape, solution, system design, features, workflow, and impact.