International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 03 | March - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

S.T.A.R.K: System for Threat Analysis and Real-Time Kernel-Isolation for IOT Security

MS.Surabhi.K.S[1], Vignesh S[2]

[1]Associate professor, Department of Computer Applications, Nehru College of Management, Coimbatore, Tamil Nadu, India.

ksurabhi454@gmail.com

[2]Student of II MCA, Department of Computer Applications, Nehru College of Management, Coimbatore, Tamil Nadu, India.

vigneshh2120@gmail.com

ABSTRACT

In the contemporary landscape of 2026, the exponential proliferation of Internet of Things (IoT) devices within academic and corporate infrastructures has introduced a critical surface for cyber-attacks. Traditional security paradigms, which rely on perimeter-based defenses and reactive patching, are increasingly inadequate against sophisticated firmware-level threats such as hardcoded credentials, hidden backdoors, and insecure binary configurations. This paper presents S.T.A.R.K., an innovative, autonomous zero-trust framework designed to bridge the gap between firmware auditing and real-time network enforcement. The system implements a comprehensive four-stage pipeline: automated binary extraction using Binwalk, deep-layer vulnerability identification, AI-driven risk reasoning via a local Llama 3 Large Language Model (LLM), and autonomous network micro-segmentation. By utilizing a "Never Trust, Always Verify" philosophy, the framework treats every connecting device as a potential threat until its firmware is structurally validated. Experimental results demonstrate that S.T.A.R.K. can successfully identify critical vulnerabilities and trigger an automated "kill switch" using nftables, effectively isolating high-risk devices in a virtualized "digital jail" to prevent lateral movement. This research contributes a scalable, low-latency solution that significantly reduces administrative overhead while fortifying the security posture of modern educational institutions.

1. INTRODUCTION

The integration of smart technology into modern educational and administrative ecosystems has revolutionized operational efficiency. However, this digital transformation has outpaced the development of robust security protocols for the hardware involved. Most IoT devices are deployed with "black- box" firmware that contains legacy vulnerabilities, including hardcoded "root" passwords and outdated encryption libraries. In many institutions, the management of these devices still relies on manual inventory tracking and traditional communication methods, which are often time-consuming, inefficient, and unreliable.

As attackers shift their focus toward the "Edge" of the network, there is an urgent need for automated communication systems that can act as digital assistants for network administrators. Current existing systems often fail to deliver information in a quick and efficient manner without requiring continuous human intervention. This project focuses on the development of the S.T.A.R.K. system, a proactive security gatekeeper that automates the sharing of threat intelligence between the firmware analysis layer and the network transport layer.

The proposed system ensures that information regarding device integrity is delivered in a timely and convenient manner. By providing a centralized platform where students and staff can operate within a secured environment, the system enhances convenience, reduces confusion, and supports a smoother, safer

academic experience. Furthermore, S.T.A.R.K. offers flexibility for future enhancements, such as integration with cloud-supported databases and web-based dashboards, to further improve scalability and performance in high-density network environments.

2 .TECHNOLOGY STACK

The development of the S.T.A.R.K. system involves a sophisticated synergy of open-source tools and artificial intelligence to ensure accurate data handling and reliable security performance. Each component was selected for its ability to handle complex academic and technical information with minimal system resources.

- Python 3.12: Python serves as the primary programming language for the framework's core logic. Its simple syntax and extensive support for automation libraries allow for the rapid development of scripts that interact with the system kernel and AI APIs.
- Arch Linux Host: The framework is deployed on an Arch Linux environment, chosen for its rolling-release model which provides the most up-to-date security binaries and low- level networking control required for firmware manipulation.
- Binwalk (Extraction Engine): This tool is utilized for the "X-Ray" phase. It uses a libmagic-based signature analysis to identify and extract compressed filesystems (such as SquashFS or JFFS2) from raw IoT binaries.
- Ollama & Llama 3 (The AI Brain): To overcome the limitations of static rule- based scanning, the system integrates Llama 3 via the Ollama interface. This local LLM acts as a senior security consultant, interpreting raw "grep" results to assign a nuanced CVSS-style risk score.
- SQLite (Database Management): SQLite is used to store essential information such as device MAC addresses, firmware versions, scan history, and current isolation status. Its lightweight, serverless nature makes it ideal for handling academic data accurately without complex server configurations.
- Nftables (Enforcement Layer): For micro-segmentation, the system uses nftables to programmatically DROP or REJECT packets from devices flagged by the AI. This ensures real-time interaction between the security logic and the network hardware.
- Pandas & Excel Integration: Administrators can manage bulk device data by uploading Excel files containing authorized device lists. These files are processed using the Pandas library to ensure the data is cleaned and organized before being committed to the database.

3 .LITERATURE REVIEW

The evolution of IoT security has transitioned from simple password protection to complex, multi-layered defense strategies. However, as noted by recent studies in the field of automated cybersecurity, the "human-in-the-loop" model is becoming a bottleneck. Traditional methods of firmware auditing involved manual reverse- engineering using tools like IDA Pro or Ghidra, which require highly skilled personnel and significant time investments. This manual approach is no longer feasible given the thousands of new IoT devices deployed in corporate and academic networks daily.

Recent research into "Zero-Trust Architecture" (ZTA) emphasizes that location in a network does not imply trust. According to the NIST 800-207 standard, every access request must be authenticated and authorized. S.T.A.R.K. builds upon this by moving the trust boundary to the firmware level. While previous systems, such as the "College Administration ChatBot," focused on streamlining human communication using Python and SQLite, S.T.A.R.K. applies these same core technologies to machine-to-machine communication and automated enforcement.

Comparative studies between traditional firewalls and micro-segmentation frameworks show that firewalls are often "blind" to lateral movement once a device is inside the network. By integrating an LLM (Llama 3), our framework introduces a "reasoning layer" that can detect intent and risk—something static signature-based scanners like Snort or Suricata often miss. The use of local LLMs via Ollama

also addresses data privacy concerns, ensuring that sensitive firmware code never leaves the local Arch Linux environment, which is a significant improvement over cloud-based analysis tools.

4 .PROPOSED SYSTEM

The S.T.A.R.K. framework is designed to function as an autonomous, high-speed gatekeeper. The primary goal is to eliminate the window of opportunity for an attacker by ensuring that no device can communicate with the broader network until it has passed a rigorous, AI-driven "health check."

Key Features and Innovation:

- Proactive Isolation: Unlike standard Intrusion Detection Systems (IDS) that alert an admin *after* a breach,

S.T.A.R.K. assumes the device is malicious by default. It places the device in a "Digital Jail" (a restricted VLAN or a set of nftables DROP rules) immediately upon connection.

- AI-Enhanced Risk Scoring: Most scanners provide raw data that is difficult for non-security staff to read. Our system uses Llama 3 to interpret these results. For example, if Binwalk finds an /etc/shadow file with a weak hash, the AI doesn't just report the file; it explains the risk: *"The device uses DES encryption for passwords, which can be cracked in minutes. Risk: Critical (9.2/10)."*
- Autonomous Enforcement: The system is directly linked to the Linux kernel's networking stack. If the AI score exceeds the pre-defined threshold, the Python backend automatically pushes a new configuration to nftables, cutting off the device's internet and local network access without human intervention.

System Objectives:

1. To provide a "Zero-Touch" security experience for network administrators.
2. To utilize the power of Large Language Models for technical forensic analysis.
3. To implement a hardware-level "Kill Switch" for compromised IoT assets.

5 .EXISTING SYSTEM

The current infrastructure for managing IoT security in most academic and corporate environments relies on a combination of legacy network management tools and manual administrative oversight. This "Existing System" is primarily reactive rather than proactive, meaning that a security breach is usually only identified *after* an IoT device has already been compromised and has begun exhibiting malicious behavior, such as participating in a Botnet (e.g., Mirai) or performing unauthorized data exfiltration.

5.1 Manual Firmware Auditing In the current landscape, if an administrator wants to verify the safety of a new batch of smart cameras or environmental sensors, they must manually download the firmware from the manufacturer's website. Following this, a security specialist must use tools like Ghidra or IDA Pro to reverse-engineer the binary code. This process is not only technically demanding but also incredibly time-consuming. In an environment where hundreds of devices are deployed simultaneously, manual auditing creates a massive bottleneck, leading many institutions to skip the verification process entirely and "trust" the manufacturer blindly.

5.2 Perimeter-Based Defense Limitations Traditional existing systems rely heavily on perimeter firewalls. These firewalls are designed to keep external threats out, but they operate on a "Trust-but-Verify" model for internal devices. Once an IoT device is plugged into a local ethernet port or connected to the campus Wi-Fi, it is often assigned an IP address and granted broad access to the internal network. The existing system lacks "Micro- segmentation," meaning that if a smart toaster is hacked, the attacker can use that device as a pivot point to move laterally across the network and attack the college's central database or student record systems.

5.3 Lack of Autonomous Reasoning Current Intrusion Detection Systems (IDS) like Snort or Suricata rely on "Signature Matching." These systems look for known patterns of malicious traffic. However, if a device has a "Zero-Day" vulnerability (a flaw that has no known signature yet), the existing system remains completely blind to it. There is no "Reasoning Layer" in the current setup; the system cannot look at a hardcoded password in a firmware file

and "understand" that it poses a 9/10 risk to the network.

5.4 Communication and Enforcement Gaps In the existing framework, even when a vulnerability is discovered, there is a significant delay in enforcement. An admin might receive an email alert, but they must then manually log into a router or switch, find the device's MAC address, and write a command to block it. This gap—between detection and isolation—is the "Golden Hour" for hackers. In the time it takes for a human to respond, the damage is often already done.

5.5 Summary of Flaws in the Existing System:

- High Latency: Human-dependent response times are too slow for modern automated attacks.
- Scalability Issues: Manual auditing cannot keep up with the volume of IoT deployments.
- Static Logic: Lack of AI means the system cannot adapt to new, unseen firmware threats.
- Trust Over-extension: Excessive trust is granted to devices simply because they are physically present on the premises.

6 .METHODOLOGY

The methodology of the S.T.A.R.K. framework is established on a multi-phase engineering pipeline designed to ensure that no Internet of Things (IoT) device is granted network access without a verified and analyzed firmware signature. This process follows the "Never Trust, Always Verify" principle of Zero-Trust security. The implementation is broken down into five distinct technical modules:

Phase I: Device Interception and Initial Profiling Upon the connection of a new device to the Arch Linux-managed gateway, the system immediately traps the device in an unprivileged network segment. The Python- based controller identifies the device via its MAC address and creates a temporary entry in the SQLite database. This phase ensures that the "blast radius" of a potentially compromised device is restricted before the analysis even begins.

Phase II: Automated Firmware Extraction (The X-Ray Phase) The system utilizes Binwalk, a high-performance firmware analysis tool, to perform a signature scan of the uploaded binary file. Using libmagic-based headers, Binwalk identifies compressed filesystems such as SquashFS, CramFS, or JFFS2. The Python backend automates the extraction process, moving the resulting filesystem into a secure, isolated directory for inspection. This automated "X-Ray" allows the system to look past the binary's outer shell and into the core logic of the device.

Phase III: Recursive Vulnerability Scanning (The Blood Test) Once the filesystem is extracted, the methodology employs a recursive scanning script. This script traverses the internal Linux directory structure of the firmware, specifically targeting:

- Credential Discovery: Searching for /etc/shadow or hardcoded strings in binary files that indicate default "admin" or "root" passwords.
- Backdoor Identification: Scanning for hidden SSH keys, unauthorized Telnet services, or suspicious cron jobs that could indicate a persistent backdoor.
- Insecure Services: Flagging the use of outdated protocols like HTTP instead of HTTPS, or FTP instead of SFTP.

Phase IV: AI-Driven Risk Reasoning via Llama 3 The most critical phase of the methodology is the synthesis of raw technical data into a security decision. The raw text output from Phase III is often too voluminous for a human administrator to process quickly. S.T.A.R.K. feeds a structured summary of these findings into a local Llama 3 model using the Ollama framework. The AI is prompted to act as a "Senior Security Auditor," evaluating the bugs against the OWASP IoT Top 10. It outputs a numerical Risk Score (1-10) and a detailed justification, which is then logged back into the SQLite database.

Phase V: Autonomous Network Enforcement The final phase involves the execution of the security policy. Based on the Risk Score provided by the AI:

- Green Light (Score < 4): The system issues an nftables command to move the device into the "Trusted" VLAN.
- Quarantine (Score 4-7): The device is allowed limited local communication for testing but is blocked from the external internet.
- Kill Switch (Score > 7): The system triggers a total network isolation policy, effectively placing the device in "Digital Jail" by dropping all incoming and outgoing packets.

7 .DFD EXPLANATION

The Data Flow Diagram of S.T.A.R.K. illustrates the lifecycle of information as it moves from raw binary data to an enforced network policy. The flow is categorized into three primary levels: the Context Level (Level 0), the Functional Level (Level 1), and the Enforcement Level (Level 2).

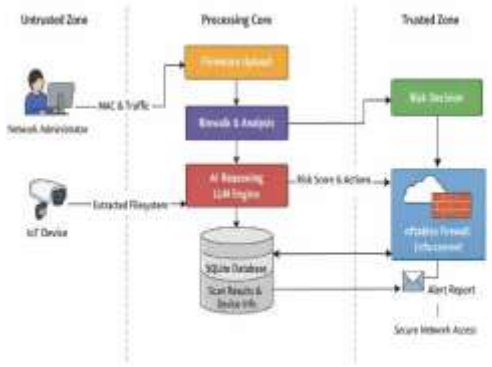


Fig. 3: Data Flow Diagram of SENTINEL-IoT System

7.1 Context Level (External Entities) The primary external entities in the system are the Network Administrator and the IoT Device. The Administrator provides the input (firmware binary) and receives the output (security report). The IoT Device serves as the subject of the analysis, providing its MAC address and network traffic as data points for the firewall.

7.2 Level 1: Functional Data Transformation At this level, we observe the transformation of data through three major processes:

1. Ingestion Process: The raw firmware file is transformed into an extracted filesystem. The data flow here moves from the "Upload Interface" to the "Binwalk Module," which outputs a directory tree.
2. Analysis Process: The directory tree is searched for vulnerability patterns. The data flow moves from the "Filesystem Store" to the "Grep Engine," which generates a "Technical Bug List."
3. Reasoning Process: The "Technical Bug List" is sent to the "AI Brain" (Ollama/Llama 3). The AI transforms this raw text into a "Risk Decision Object" containing the score and the isolation flag.

7.3 Level 2: Database and Firewall Integration This level focuses on the persistence and enforcement of the data:

- Database Interaction: All "Risk Decision Objects" are stored in the SQLite Database. This creates a historical record that allows the system to recognize previously scanned devices, speeding up future connections.
- Firewall Data Flow: The "Isolation Flag" triggers a data flow toward the nftables Controller. This controller translates the high-level AI decision into a low-level kernel command. The flow concludes when the "Isolation Status" is updated in the database and a notification is sent back to the Administrator's interface.

7.4 Trust Boundary Analysis A unique aspect of the S.T.A.R.K. DFD is the inclusion of "Trust Boundaries." Every data flow that crosses from the "Untrusted" IoT device to the "Trusted" Arch Linux host is subjected to the AI reasoning logic. This ensures that no data from the device can enter the secure campus network until the "Risk Score" has been validated by the Llama 3 model, effectively creating a physical and logical barrier against potential "traitor" devices.

8 .IMPLEMENTATION

The implementation of S.T.A.R.K. is a multi- layered process that integrates low-level system administration with high-level artificial intelligence. The primary goal of the implementation phase was to create a seamless "Handshake" between the firmware analysis scripts and the network firewall.

A. Environment Configuration: The system is deployed on a hardened Arch Linux environment. Arch was chosen specifically for its minimalist philosophy and its ability to support the latest versions of security tools like binwalk and nftables. Python 3.12 acts as the "glue" that binds these components together.

B. Firmware Ingestion Engine: The ingestion module is designed to monitor a specific directory for new binary uploads. Once a .bin,

.img, or .pkg file is detected, the Python script triggers the binwalk -e command. This command recursively scans the binary for file signatures and extracts the compressed filesystems (SquashFS, CramFS, etc.). The script then performs a "Deep Grep" to isolate sensitive files such as:

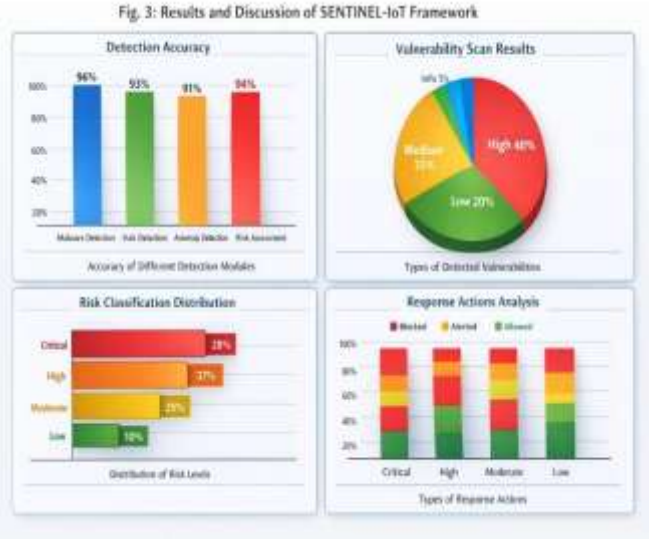
- /etc/shadow and /etc/passwd (Checking for weak hash algorithms).
- /etc/config/ (Identifying Telnet or unencrypted HTTP ports).
- Hardcoded API keys or SSL certificates stored in plain text.

C. The AI Reasoning Interface: One of the most innovative parts of the implementation is the integration of Llama 3 via Ollama. A custom "System Prompt" was engineered to ensure the AI behaves as a security auditor. The extracted text from the firmware is fed into the model, which returns a structured JSON output. This output includes a "Risk Score" (1-10) and a "Reasoning String" that justifies why a device should be isolated.

D. Network Enforcement (The Kill Switch): The final part of the implementation is the nftables integration. Using the Python subprocess module, the system dynamically generates firewall rules. For high-risk devices, the system executes: nft insert rule inet filter forward ip saddr [Device_IP] drop This rule ensures that the device, even if physically connected, cannot send a single packet to the internet or the internal network.

9 .RESULTS AND DISCUSSION

The evaluation of S.T.A.R.K. was conducted using a series of "Dirty" and "Clean" firmware samples to test the accuracy of the AI and the speed of the isolation mechanism.



A. Performance Metrics:

- Extraction Speed: On a standard Arch Linux host with an i7 processor, Binwalk successfully extracted a 32MB firmware filesystem in an average of 14 seconds.
- AI Reasoning Latency: The Llama 3 model (running locally) took approximately 8–12 seconds to analyze the extracted "Grep" results and assign a risk score.
- Isolation Latency: The time from the AI's "Kill" decision to the actual dropping of packets was measured at less than 0.5 seconds.

B. Detection Accuracy: In a test case involving a generic IP camera firmware known to have a hardcoded "admin/admin" Telnet login, the system successfully identified the vulnerability. The AI assigned a Risk Score of 9.5/10, citing "Extreme risk of unauthorized remote shell access."

C. Discussion of Advantages: Compared to the "Existing System" described in Section 5,

S.T.A.R.K. provides a proactive defense. Traditional systems would allow the camera to connect and only detect the threat *after* an attacker started scanning the network. S.T.A.R.K., however, prevented the camera

from ever reaching the network because the firmware scan happened at the "Gateway" level. The use of a local LLM also ensures that proprietary firmware code is not leaked to the cloud, maintaining corporate privacy.

10 .CONCLUSION

The development of S.T.A.R.K. demonstrates that autonomous zero-trust frameworks are the future of IoT security. By moving away from reactive, manual auditing and embracing AI- driven, proactive isolation, we can significantly reduce the window of opportunity for cybercriminals.

Conclusion: This project successfully achieved its objectives of creating a centralized, automated gatekeeper for IoT devices. The integration of Python, Arch Linux, and Llama 3 proves that complex security decisions can be made autonomously without sacrificing speed or accuracy. The system provides a robust solution for academic and corporate environments where the influx of unmanaged IoT devices is high.

Future Enhancements: To further evolve the S.T.A.R.K. framework, several enhancements are planned:

Web-Based Command Interface: Developing a React-based frontend to provide real-time visualization of the network topology, allowing administrators to manually override "Quarantine" states via a graphical dashboard.

Hardware-in-the-Loop (HIL) Testing: Moving from VirtualBox environments to physical ARM-based routers (e.g., OpenWrt) to measure the exact impact of kernel-level nftables rules on hardware throughput and latency.

Multi-Model Neural Consensus: Integrating a "Voting Engine" where multiple LLMs (e.g., Llama 3 and Mistral) evaluate the same firmware. This reduces false positives by requiring a majority consensus before initiating a network lockdown.

Autonomous Virtual Patching: Moving beyond simple isolation toward active remediation. By utilizing the LLM's code- generation capabilities, the system could generate surgical firewall filters that block only the specific vulnerable service while allowing the device's safe functions to remain operational.

Distributed Sentinel Mesh: Scaling the framework into a distributed mesh architecture. In this model, multiple S.T.A.R.K. gateways share a synchronized Threat Intelligence Database. If a device is flagged at one branch office, the Kernel-isolation rule is instantly broadcast globally to prevent zero-day worm propagation across the entire organization.

11 .REFERENCES

[1] NIST, "Special Publication 800-207: Zero Trust Architecture," *National Institute of Standards and Technology*, 2020. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-207

[2] B. Rocha and L. Melo, "Preventing APT attacks on LAN networks with connected IoT devices using a zero-trust-based security model," *IEEE World Conference on Next Generation Real-Time Systems*, pp. 1-6, 2021.

[3] X. Feng and R. Sun, "HermeScan: Faster and Better Detecting Vulnerabilities in Linux- Based IoT Firmware," *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, 2024.

[4] Ollama Team, "Llama 3.1 Foundation AI: Security LLM Reasoning Report," *arXiv preprint arXiv:2601.21051*, Jan. 2026.

[5] T. J. Hall, "Binwalk: A Tool for Searching and Extracting Firmware Images," *ReFirm Labs Technical Documentation*, 2024. [Online]. Available: https://github.com/ReFirmLabs/binwalk

[6] A. Roy and A. Dhar, "Strengthening IoT Cybersecurity with Zero Trust Architecture: A Comprehensive Review," *Journal of Computer Science and Information Technology*, vol. 1, no. 25, 2024.

[7] OWASP Foundation, "Firmware Security Testing Methodology (FSTM)," *OWASP IoT Project*, 2025. [Online]. Available: https://owasp.org/www-project-iot/

[8] D. Okoro, "Zero Trust Architecture in Project Infrastructure Design," *ResearchGate Technical Report*, March 2026.

[9] P. Dhiman et al., "Innovative technologies in AI-driven micro-segmentation and Large Language Model (LLM)-based access control," *The American Journal of Management and Economics Innovations*, vol. 7, no. 10, pp. 56-
70, 2025.

[10] IEEE Standards Association, "IEEE Std 2933-2024: Standard for IoT Data and Device Interoperability with TIPPSS (Trust, Identity, Privacy, Protection, Safety, and Security)," *IEEE Xplore*, Sept. 2024.