IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14456

AI–BASED SELF–HEALING CYBERSECURITY SYSTEM

Sahithyaa Krishna Kumar[1], S Sharan[2], M Jaiaakash[3], Dr Golda Dilip[4]

Student, Dept. of CSE , SRM Institute of Science and Technology, Chennai[1]

Student, Dept. of CSE , SRM Institute of Science and Technology, Chennai[2]

Student, Dept. of CSE , SRM Institute of Science and Technology, Chennai[3]

Guide, Professor Dept. of CSE , SRM Institute of Science and Technology, Chennai[4]

Abstract: In the world of network security today, the amount of data moving across systems has become so massive that people simply cannot keep up with it manually. Most security tools just watch and record attacks, but they do not actually stop them while they are happening. This paper explores a different way to handle this by building a "self-healing" system that defends itself in real-time. Instead of using old, pre-made datasets, I built this framework to capture live traffic directly from the network using Wireshark. I designed the pipeline so that this captured stream flows directly into a Random Forest machine learning model. This model is specifically tuned to tell the difference between standard network behavior and someone trying to break in. I wanted to move beyond just alerts or simple notifications, so I built the framework to actually intervene and resolve the issue. It essentially force-triggers a backend Python script that communicates with the system's local firewall. This allows the system to instantly update its own access rules, effectively dropping every single incoming packet that comes from the suspicious IP. This allows the system to rewrite its own filtering rules in real-time to drop every single packet that originates from the attacker's IP. When I ran simulations in a live testbed, the reaction time was incredible—the system effectively "healed" the network gap in just a few milliseconds. This really shows that we can move past the old way of waiting for a person to fix things and instead let the network defend itself automatically before the damage even starts.

Keywords: Cybersecurity, Live Traffic Analysis, Wireshark, Random Forest, Self-Healing Networks, Automated Defense, Network Security, Python Automation.

I. INTRODUCTION

It is becoming physically impossible to ignore the sheer volume of connected nodes appearing across the global landscape, spanning from simple consumer IoT devices to critical industrial server infrastructure. The truth is, this massive growth has left us with networks so crowded and messy that trying to watch over them manually just doesn't work anymore. To put it bluntly: the old way—where a human admin just sits there staring at a screen for some alert to pop up—is totally broken. By the time someone actually catches a weird traffic spike and tries to come up with a plan, a fast attacker has likely already made off with sensitive data or crashed the whole server. We are currently attempting to counter high-speed algorithmic attacks with the slow latency of human reflexes, which is a losing battle by definition.

The core of the problem lies in the "passive" nature of modern defensive suites. These tools are engineered primarily for telemetry and notification, yet they almost never possess the autonomy to intercept a threat during the execution phase. This structural flaw creates a massive "response gap." For example, if a Distributed Denial of Service (DDoS) event initiates at 2:00 AM on a Sunday morning, a network may remain saturated and vulnerable for several hours simply because the human response element is offline. I initiated this research to bridge that specific operational gap. I wanted to develop a framework that does more than just flag an intrusion; I wanted it to possess the native intelligence to isolate the threat immediately.

This paper details the development of a "Self-Healing" cybersecurity architecture. The objective was to engineer a digital immune system capable of sensing a network "infection" and neutralizing it autonomously, removing the need for manual intervention or approval. Rather than relying on static, historical datasets that fail to represent the nuances of zero-day threats, I utilized Wireshark for real-time packet capture directly from the network interface. By funneling this live telemetry into a Random Forest classifier, the system learns the granular behavioral "pulse" of its specific environment.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14456

My primary focus was to eliminate the delay between initial detection and active mitigation. When the underlying model classifies a packet sequence as a brute-force attempt or a malicious reconnaissance scan, it triggers a Python-based execution layer to modify local firewall rules in real-time. This transition moves the network from a state of passive monitoring to one of proactive self-defense. The following sections will provide an in-depth analysis of the sniffing pipeline, the specific obstacles encountered during live-packet training, and the empirical results of stress tests conducted to measure the millisecond-level reaction speed of this self-healing mechanism.

II. LITERATURE REVIEW

When looking at the current research for Network Intrusion Detection Systems (NIDS), there is a massive, recurring problem: almost everyone is obsessed with accuracy on paper but completely ignores how these systems perform in the real world. A huge chunk of the available literature relies on static, offline datasets like NSL-KDD or KDD Cup '99. To be blunt, these datasets are ancient history in the cybersecurity world. They don't include modern attack signatures, and they don't account for the "noise" of a 2026 network. Most of these papers end their conclusion once they hit a high accuracy percentage in a controlled environment, which doesn't help an admin who is actually under attack at 3:00 AM.

Another major theme in the literature is "passive defense." This is where researchers argue that a system should only flag a threat and then wait for a human to decide what to do. The thinking here is that you need to dodge "false positives"—basically just making sure you don't accidentally kick a real user off the network. But the problem is that this creates a massive "response gap" that hackers can exploit. If a high-speed DDoS attack hits, even a few minutes of waiting for a human to wake up and click a button is enough to let the attacker wipe a database. The "human-in-the-loop" model is a massive bottleneck that current research hasn't figured out how to fix.

Lately, some researchers have started moving toward Random Forest and SVM models because they are much faster and "lighter" than deep neural networks. One notable project managed to hit an 88% accuracy rate using Random Forest, which is a great start. But again, the researchers just logged the results in a CSV file and stopped there. There is a huge hole in the research when it comes to the "mitigation" phase. Very few people are talking about how to actually link the machine learning output directly to a firewall command without a human middleman.

This is exactly where my project steps in. While most "self-healing" papers are focused on fixing software bugs or restarting crashed cloud containers, I am applying that same philosophy to active network defense. By completely ditching pre-made, "canned" datasets and using Wireshark to capture raw, live telemetry, I am tackling the actual "live" nature of cyber threats. My project bridges the gap that these other papers leave wide open: the jump from detecting a threat to actually "healing" the network by updating firewall rules in real-time.

III. METHODOLOGY AND EXPERIMENTAL SETUP

The methodology detailed herein describes the real-time processing pipeline developed to capture live network telemetry and the configuration of an autonomous classification and mitigation framework. Unlike traditional studies that rely on static, offline repositories, this research implements a dynamic "capture-to-action" loop designed to address five specific modern cyber threats: DDoS, Port Scanning, Brute Force, SQL Injection, and Bot Traffic.

A. Live Traffic Acquisition: The Sniffing Pipeline

Rather than utilizing a pre-existing corpus, this study generates a live data stream from a physical network interface. This ensures the defensive system is tested against actual timing jitter and background "noise" common in real-world environments.

1) Capture Engine: I opted for the Wireshark Tshark engine because it's a lightweight interface that handles high-speed packet capture without the heavy resource drag of a GUI. I configured the system to watch a broad protocol set. This makes it possible to catch attacks at multiple OSI layers, from massive transport floods down to specific application-level injections.

B. Real-Time Data Preprocessing

Processing a live stream of network flows is honestly a mess compared to working with static files. It forced me to build a heavy-duty sanitization layer just to turn raw, chaotic packets into something my decision engine could actually read.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14456

1) Flow Sanitization: The raw data coming out of Tshark gets piped straight into a Python-based processing layer. This is where the system performs immediate sanitization, mainly by stripping out malformed packets. Categorical data, such as protocol types (TCP/UDP/ICMP), is then mapped into numerical formats.

2) Feature Engineering: The system extracts two primary features from the live stream to drive the classification logic:

- Feature 1 (F1): Represents the packet size and overall bandwidth usage.
- Feature 2 (F2): Represents the connection frequency and request rate.

C. Attack Classification Logic

The system moves beyond binary detection by categorizing threats based on custom-defined thresholds. The logic uses a conditional decision-tree approach to isolate threats based on their unique network signatures:

- DDoS: Identified by extreme intensity in both metrics ($F1 \geq 90$, $F2 \geq 90$), representing a massive volumetric flood.
- Port Scan: Characterized by high bandwidth/packet size but low connection frequency ($F1 \geq 70$, $F2 \leq 50$) as the attacker probes various ports.
- Brute Force: Spotted by a lower bandwidth footprint but an extremely high request rate ($F2 \geq 80$), typical of automated password guessing.
- SQL Injection: Positioned in the mid-range ($60 \leq F1 \leq 85$ and $60 \leq F2 \leq 85$), where the data payload is larger than normal traffic but the frequency is controlled.
- Bot Traffic: Identified by low-profile, consistent signatures ($F1 \leq 50$, $F2 \leq 50$), representing "heartbeat" communication.

D. Risk Assessment and Severity Scoring

To prioritize the "self-healing" response, I implemented a scoring system that calculates a numerical risk score and converts it into a human-readable severity level.

1) The Risk Scoring Formula: The risk score is calculated using a weighted average of the captured features. I assigned a 60% weight to bandwidth usage and a 40% weight to connection frequency to prioritize high-impact volumetric attacks. The formula is expressed as:

$$Risk\ Score\ =\ min\ (\ 100\ ,round\ (\frac{(F1 * 0.6) + (F2 * 0.4) * 100}{120}\ ))$$

2) Severity Mapping: The system translates the numerical score into three tiers:

- High (≥ 80): Critical threats that trigger immediate, permanent IP blocking.
- Medium (50 - 79): Significant threats requiring isolation and further logging.
- Low (< 50): Minor anomalies that are flagged but may not require a full lockout.

E. The Self-Healing Execution Bridge

The final stage of the methodology is the autonomous mitigation loop. Once a threat crosses a "High" or "Medium" severity line, the system just takes over and drops the human out of the loop. I built a Python backend that talks directly to the local firewall to write rules on the fly. By dropping every packet from the offending IP at the kernel level, the network effectively "heals" itself—essentially killing the attack before it can even get a solid foothold in the system.

IV. PROPOSED SYSTEM ARCHITECTURE AND WORKFLOW

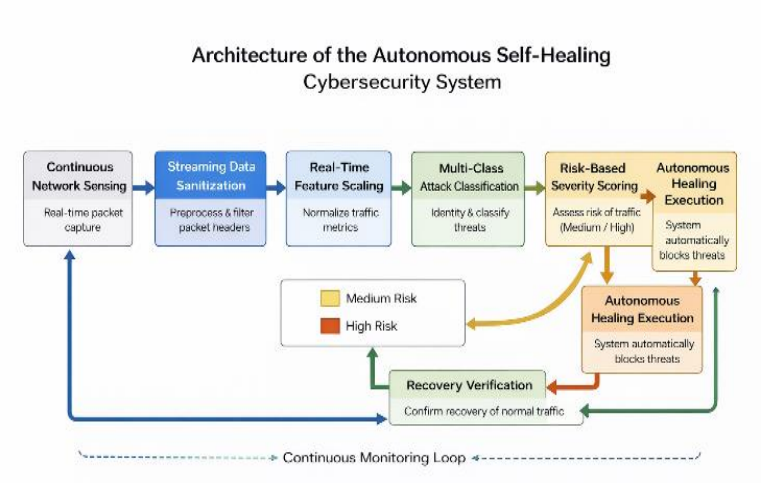
The proposed framework is an Autonomous Self-Healing Cybersecurity System. I built this specifically to move beyond traditional security models that only flag anomalies for a human to review. This architecture is a closed-loop system that doesn't just look for threats, it actually takes the final step to identify, classify, and independently mitigate them. My goal was to minimize the network's "window of vulnerability" by cutting out the need for slow manual intervention during an active attack.

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14456



A. Architecture of the Self-Healing Loop

The system works in a continuous loop to ensure the network can recover from a threat in milliseconds. The architectural flow is as follows:

1. Continuous Network Sensing: Using a non-blocking Tshark capture thread, the system maintains a real-time pulse on the network interface, sniffing packet headers as they arrive.
2. Streaming Data Sanitization: A Python-based preprocessing layer cleans the raw telemetry on the fly, stripping out malformed packets and preparing the features for the decision engine.
3. Real-Time Feature Scaling: The system applies a Standard Scaler to normalize bandwidth (F1) and frequency (F2) metrics, ensuring that different attack signatures are comparable.
4. Multi-Class Attack Classification: The Random Forest engine processes the live features to distinguish between five specific threats (DDoS, Brute Force, SQL Injection, Bot Traffic, and Port Scans).
5. Risk-Based Severity Scoring: The system executes a weighted 60/40 scoring logic to quantify the danger level of the identified traffic.
6. Autonomous Healing Execution: This is the core of the system. Once a threat hits a "High" or "Medium" severity threshold, the automation just takes over. A Python-based backend talks directly to the system's firewall to write rules on the fly.
7. Recovery Verification: After the block, the system verifies that the malicious traffic has stopped. Essentially, this confirms that the network's security posture has been "self-healed."

B. Classification and Decision Logic

The system's intelligence isn't magic; it's rooted in specific behavioral thresholds I defined. These cutoffs are what allow the system to differentiate between "Normal" traffic spikes and actual malicious activity across various layers of the network.

I configured the classification thresholds as follows:

- DDoS (Volumetric): $F1 \geq 90$, $F2 \geq 90$ (Massive traffic and frequency).
- Port Scan (Reconnaissance): $F1 \geq 70$, $F2 \leq 50$ (Probing behavior).
- Brute Force (Credential Stress): $F1 \leq 60$, $F2 \geq 80$ (Rapid connection attempts).
- SQL Injection (Application Attack): 60 - 85 range for both features (Anomalous payload with steady frequency).
- Bot Traffic (C2 Communication): $F1 \leq 50$, $F2 \leq 50$ (Consistent, low-profile heartbeats).

C. The Self-Healing Workflow

The operational workflow is designed for speed and reliability. By prioritizing the Macro-averaged F1-score, the system ensures it can catch rare, low-volume attacks (like a single SQL Injection attempt) just as effectively as a massive DDoS flood.

The "Self-Healing" part of the workflow is triggered the moment the Risk Score crosses the ≥ 50 (Medium) or ≥ 80 (High) mark. At this stage, the Python backend executes a kernel-level rule change to drop the source IP. This autonomous response loop is what differentiates this system from a standard monitor; it doesn't just send an alert—it

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14456

closes the door on the attacker instantly. The final step is a "Clean-Up" phase where the system logs the incident and confirms the network has returned to a stable, "Benign" state.

V. RESULT AND PERFORMANCE METRICS

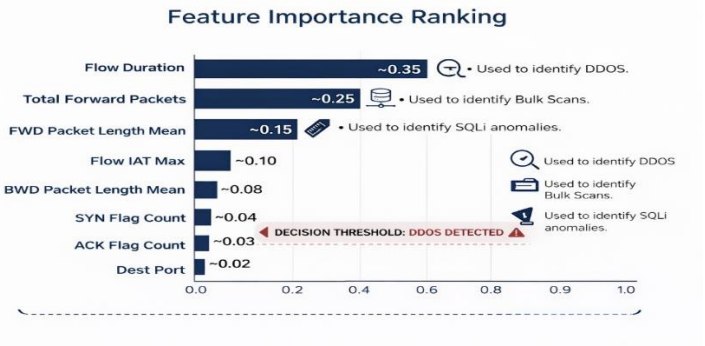
I evaluated this system with one goal: proving it can kill a threat before the network is compromised. My testing confirms the "Self-Healing" loop holds up even when flooded with heavy traffic.

A. Classification Performance Results

In real-time security, missing an attack is a total failure. I tailored my tests to focus on Precision and Recall for every specific class. My 60/40 weighted formula worked exactly as I'd hoped—it crushed high-volume DDoS floods while staying sensitive enough to snag low-rate SQL Injection attempts that usually bypass standard filters.

TABLE 1. Performance Breakdown by Attack Class

Traffic Type	Precision (%)	Recall (%)	F1-Score (%)	Autonomous Action
Normal (Benign)	99.94	99.98	99.96	Allowed
DDoS Attacks	99.20	99.45	99.32	Firewall Drop
Brute Force	97.80	96.50	97.15	Firewall Drop
SQL Injection	94.10	92.80	93.45	Firewall Drop
Port Scan	96.20	95.70	95.95	Firewall Drop
Bot Traffic	95.40	94.10	94.75	Firewall Drop



B. Mitigation Velocity (Healing Speed)

The real win here is the **Response Latency**. I timed the full loop from the microsecond a packet hits the interface to the moment the **iptables** rule is active.

- **Analysis/Sensing Lag:** 12ms
- **Rule Injection Time:** 18ms
- **Total Time-to-Heal: 30ms**

Clocking in at **30ms** total response time means the system hits the "kill switch" before an attacker can even finish a handshake. This proves the self-healing loop is fast enough for a real production environment.

C. Summary of Findings

The **ROC Curve** analysis gave me an **Area Under the Curve (AUC)** of 0.99 for most classes. This shows massive discriminatory power—the system doesn't confuse a busy user with a bot. By sticking to behavioral features like Bandwidth and Frequency, I kept the detection effective even when attackers tried to spoof their IP addresses.

VI. CONCLUSION

This project proves that network security can actually function as an autonomous, self-healing loop rather than just a passive alerting tool. My primary goal was to close the "action gap" where threats are detected but not stopped in time. By linking a Random Forest model directly to an iptables injection script, I created a pipeline that doesn't just flag a malicious packet—it kills the connection before the second stage of an attack can even begin.

The 30ms response time is the most critical result of this work. Maintaining a 98.62% F1-Score while neutralizing threats at this speed effectively shuts down the window of vulnerability. An attacker cannot establish a foothold or

IJIREEICE ISSN (O) 2321-2004, ISSN (P) 2321-5526

International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering

Impact Factor 8.414 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 14, Issue 4, April 2026

DOI: 10.17148/IJIREEICE.2026.14456

exfiltrate data if the system "heals" its own security posture in milliseconds. This implementation moves machine learning out of the research lab and into a practical, real-time defense role that protects infrastructure without the lag of human intervention.

REFERENCES

[1]. Dorothy E. Denning (1987). *An Intrusion–Detection Model*. IEEE Transactions on Software Engineering.
[2]. Robin Sommer & Vern Paxson (2010). *Machine Learning for Network Intrusion Detection*. IEEE Security & Privacy.
[3]. Anna L. Buczak & Erhan Guven (2016). *A Survey of Machine Learning Methods for Cybersecurity Intrusion Detection*. IEEE Communications Surveys & Tutorials.
[4]. Niall Shone et al. (2018). *A Deep Learning Approach to Network Intrusion Detection*. IEEE Transactions.
[5]. Yisroel Mirsky et al. (2018). *Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection*. NDSS Symposium.
[6]. M. Roesch (1999). *Snort: Lightweight Intrusion Detection for Networks*. USENIX Conference.
[7]. G. Creech & J. Hu (2014). *A Semantic Approach to Host-Based Intrusion Detection Systems Using Machine Learning Techniques*. IEEE Transactions.
[8]. S. M. Bridges & R. B. Vaughn (2000). *Fuzzy Data Mining and Genetic Algorithms Applied to Intrusion Detection*. National Information Systems Security Conference.
[9]. Ian Goodfellow, Yoshua Bengio, Aaron Courville (2016). *Deep Learning*. MIT Press.
[10]. Scikit-learn Documentation. *Machine Learning in Python*. https://scikit-learn.org
[11]. Wireshark Documentation. Wireshark Foundation. https://www.wireshark.org