Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Comparative Evaluation of the Impact of Traditional Firewalls versus IDS/IPS Based Solutions in Home Networks with IoT Devices

Bryan Eduardo Cisneros Andrade
Universidad Politecnica Salesiana
Karen Lissette Ponce Astudillo
Universidad Politecnica Salesiana
Miguel Arcos-Argudo
marcos@ups.edu.ec
Universidad Politecnica Salesiana

Research Article

Keywords: IoT Security, Intrusion Prevention System (IPS), Suricata, Firewall Performance, Virtualization, Deep Packet Inspection, MQTT

Posted Date: March 31st, 2026

DOI: https://doi.org/10.21203/rs.3.rs-8990429/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Comparative Evaluation of the Impact of Traditional Firewalls versus IDS/IPS Based Solutions in Home Networks with IoT Devices

Bryan Eduardo Cisneros Andrade[1†],
Karen Lissette Ponce Astudillo[1†], Miguel Arcos-Argudo[1*†]

[1]Department of MSI, Universidad Politecnica Salesiana, Cuenca, Ecuador.

*Corresponding author(s). E-mail(s): marcos@ups.edu.ec;
Contributing authors: bcisnerosa1@est.ups.edu.ec;
kponcea@est.ups.edu.ec;
†These authors contributed equally to this work.

Abstract

Residential IoT ecosystems are increasingly besieged by advanced application-level exploits capable of bypassing conventional perimeter security measures. This study quantitatively compares the defensive efficacy and computational overhead of a standard Layer 3/4 packet filter (iptables) against a Layer 7 Intrusion Prevention System (Suricata) deployed within a virtualized residential testbed. We simulated a spectrum of nine specific threat vectors targeting IoT communication standards, including MQTT DoS, HTTP fuzzing, SQL injection, and unauthorized firmware injection. Results: The traditional firewall demonstrated a complete inability to identify or mitigate application-layer incursions (yielding a TPR of 0%), rendering the network defenseless. Conversely, the IPS configuration attained absolute mitigation success, recording a 100% Blocking Effectiveness (PDB) across every evaluated scenario. Significantly, performance metrics revealed a paradox: during high-velocity attacks (e.g., Port Scanning and Fuzzing), the standard firewall suffered from CPU saturation (reaching 27% due to I/O stress), while the IPS maintained operational stability (4–8% peak load) by preemptively rejecting hostile packets at the interface level. Conclusion: We demonstrate that L3/L4 firewalls are insufficient for IoT security, not only due to their blindness toward Layer 7 threats but also their susceptibility to resource

depletion under stress. Consequently, the implementation of Deep Packet Inspection (DPI) via an IPS is validated as a superior and more resource-efficient strategy for hardening domestic IoT infrastructures.

Keywords: IoT Security, Intrusion Prevention System (IPS), Suricata, Firewall Performance, Virtualization, Deep Packet Inspection, MQTT

1 Introduction

The exponential proliferation of Internet of Things (IoT) endpoints in domestic environments has introduced novel vulnerabilities, rendering traditional residential network architectures inadequate for effective mitigation [1, 2]. Most household deployments rely on commodity routers equipped with rudimentary Network and Transport layer (L3/L4) firewalling mechanisms, which operate primarily via static IP and port-based filtering [3].

However, contemporary threats targeting smart ecosystems—including SQL injections, Cross-Site Scripting (XSS), and illicit firmware injections function within the Application Layer (Layer 7), effectively circumventing these standard perimeter controls. While sophisticated solutions capable of Deep Packet Inspection (DPI) specifically Intrusion Prevention Systems (IPS) exist, their adoption rates in residential networks remain negligible. This reluctance stems largely from the prevailing misconception that IPS implementations are administratively complex and impose excessive computational overhead, a critical constraint for resource-limited hardware.

There is a paucity of recent literature providing a dual quantification of the security gap and the tangible performance costs (CPU and RAM) distinguishing traditional firewalls from IPS solutions under hostile conditions. This study bridges this gap through the implementation and evaluation of a standardized, reproducible virtualized laboratory environment. We conduct a rigorous comparative analysis of a standard Traditional Firewall (`iptables`) versus an open-source IPS (Suricata), measuring both defensive efficacy and resource consumption against nine specific IoT threat vectors.

The principal contributions of this research are outlined below:

- **Quantifying the Security Deficit:** Through empirical experimentation, we demonstrate that standard L3/L4 firewalls fail completely (yielding a 0% detection rate) against sophisticated application-layer vectors. In contrast, a correctly tuned IPS configuration secures a 100% Blocking Effectiveness rate while maintaining negligible false positives.
- **The Efficiency Paradox:** We challenge the prevailing assumption regarding firewall efficiency. Our data reveals a counter-intuitive phenomenon where, during high-volume assaults (such as Fuzzing or Scanning), the traditional firewall suffers critical CPU saturation (peaking at 27%) driven by kernel interrupt handling and I/O logging stress. Conversely, the IPS demonstrates superior resource management (peaking at only 8%) by discarding hostile packets at the interface level, thus preserving system availability.

- **Standardized Testing Artifacts:** We provide a fully validated artifact package, including IoT-optimized IDS signatures and an Infrastructure-as-Code (Vagrant) testbed, facilitating exact replication of these trials for academic and educational advancement.

2 Related Work

2.1 Overview of IoT Security Strategies

Extensive literature has addressed the inherent vulnerabilities within the Internet of Things (IoT) landscape. Initial academic efforts largely prioritized securing resource-constrained hardware through lightweight encryption protocols and authentication schemes [1]. Nevertheless, as highlighted by Khan and Salah [4], such preventative measures are rendered ineffective if a device is already breached or if security flaws reside within the application layer itself. As a result, implementing robust defense mechanisms at the network level has emerged as a mandatory security requirement.

Residential network architectures generally depend on consumer-grade routers equipped with basic Network Address Translation (NAT) and rudimentary packet-filtering capabilities, such as iptables. Although these traditional tools can block unsolicited external access, they lack the Deep Packet Inspection (DPI) functionality necessary to identify threats hidden within data payloads like SQL injections a limitation emphasized in the architectural review by Sicari et al. [5].

2.2 IDS/IPS in IoT Environments

To overcome the inherent shortcomings of conventional firewalls, research focus has increasingly pivoted toward Intrusion Detection Systems (IDS). Various studies have explored the adaptation of engines such as Snort and Suricata for IoT ecosystems. Notably, Fragkouli et al. [6] validated the viability of "RPiDS" (Raspberry Pi IDS) by developing specialized rule sets for IoT protocols. Their findings confirmed that constrained hardware is capable of supporting IDS operations, although they observed performance degradation during high-traffic intervals. In parallel, a comprehensive survey by Zarpelão et al. [7] examined IoT detection methodologies, concluding that while signature-based mechanisms (e.g., Suricata) are highly effective against known vulnerabilities, they demand precise configuration to prevent resource depletion [3, 8].

2.3 Performance and Virtualization Gaps

Conventional literature comparing traditional firewalls with IDS/IPS solutions typically asserts that firewalls offer superior efficiency due to minimal processing overhead. However, contemporary findings have begun to challenge this perspective, particularly under specific stress scenarios. A significant limitation in existing benchmarks is their reliance on physical hardware, which restricts reproducibility. There is a noticeable gap in research utilizing Infrastructure-as-Code (IaC) frameworks, like Vagrant, to establish standardized security testbeds. Moreover, investigations into Network Function Virtualization (NFV) by Sforzin suggest that the performance dynamics between user

space packet processing (IDS) and kernel space filtering (Firewall) exhibit non-linear trade-offs during saturation events [9, 10].

2.4 Distinction of This Work

This research contributes to the state of the art by offering a direct, replicable evaluation of a standard L3/L4 firewall against a Layer 7 IPS within a virtualized residential network. Departing from studies that exclusively prioritize detection metrics, we present an extensive analysis of system performance under saturation. A novel contribution of this study is the identification of the "Firewall Saturation Paradox," demonstrating that a traditional firewall can paradoxically incur higher system resource costs than an IPS during volumetric attacks, driven by kernel-space and I/O exhaustion a phenomenon frequently neglected in prior studies.

3 Methods

3.1 Experimental Design and Objectives

We utilized a quantitative experimental framework centered on a controlled comparison of two distinct security setups within an identical network topology. The core aim is to assess both the protective efficacy and operational efficiency in detecting and blocking application layer threats directed at IoT endpoints. The experimental parameters are defined as follows:

Independent Variable: The security mechanism deployed at the network gateway.

- **Scenario A (Control):** Traditional L3/L4 Firewall (iptables) with a default-deny policy, allowing only specific ports (TCP 1883, 5000, 8080) without Deep Packet Inspection (DPI).
- **Scenario B (Experimental):** Intrusion Prevention System (Suricata) operating in NFQUEUE mode (IPS) with a specific ruleset for IoT protocols (MQTT, HTTP), which are frequent targets for DoS and injection attacks [11, 12].

Dependent Variables:

- **Security Effectiveness:** Measured via True Positive Rate (TPR), False Negative Rate (FNR), and Blocking Effectiveness (PDB).
- **Performance Efficiency:** Measured via System CPU Load (Average and Peak) and Available Memory during attack execution.

3.2 Laboratory Environment

To guarantee experimental reproducibility and strictly isolate the test variables, the evaluation testbed was orchestrated using Infrastructure-as-Code (IaC) paradigms via Vagrant and VirtualBox. This virtualized ecosystem is explicitly designed to replicate a standard residential network interface exposed to external hostile vectors.

3.2.1 Network Topology (Logical)

The network architecture is defined by a linear arrangement of three virtual machines (VMs): an external Threat Actor, an intermediary Security Gateway (IDS/FW), and a vulnerable IoT Endpoint. This topology mandates that all incoming traffic streams must traverse the gateway's inspection interfaces (eth1/eth2) prior to accessing the victim device, ensuring no packet bypasses analysis.

- **Attacker Network:** 192.168.56.0/24 (Representing the External/Internet domain).
- **IoT Network:** 192.168.57.0/24 (Representing the internal Home LAN).

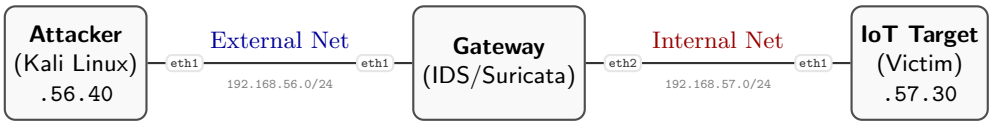


Fig. 1 Logical Topology of the Experimental Testbed

3.2.2 Virtual Machine Configuration

Comprehensive specifications regarding the software and network configuration for each node in the cluster are outlined in Table 1.

Table 1 Specific configuration details for each cluster node

VM Name	Role	OS	Interfaces & IPs	Installed Software
Attacker	Threat Source	Kali	eth1: 192.168.56.40	Nmap, hping3, curl, python3
IDS	Gateway	Ubuntu	Ext: 192.168.56.20 Int: 192.168.57.20	Suricata 6.x, iptables, tcpdump
IoT-Target	Victim	Ubuntu	eth1: 192.168.57.30	Docker, Mosquitto, DVWA

Note: To guarantee traffic fidelity and ensure the correct functioning of Suricata's prevention mechanisms, hardware offloading features specifically Generic Segmentation Offload (GSO) and Generic Receive Offload (GRO) were deactivated on the IDS network interfaces.

3.3 Security Configurations

To assess the operational impact and security benefits of Deep Packet Inspection (DPI), we established two contrasting security postures within the Gateway VM environment.

3.3.1 Scenario A: Traditional Firewall (Control)

This configuration emulates the architecture of a typical consumer grade router or entry-level gateway. We leveraged the native Linux packet filtering subsystem (iptables), establishing a stateful/stateless inspection policy restricted strictly to the Network and Transport layers (L3 and L4).

- **Policy:** Default DROP for the FORWARD chain.
- **Rules:** Explicit ACCEPT rules were created only for the ports required by the IoT services: TCP/1883 (MQTT), TCP/5000 (SmartPlug API), and TCP/8080 (Web Interface).
- **Limitation:** No application-layer inspection was performed. Malicious payloads (e.g., SQL injection strings) transmitted over allowed ports (e.g., port 8080) were permitted to pass unrestricted.

3.3.2 Scenario B: Intrusion Prevention System (IPS)

In this configuration, Suricata was implemented to function as an inline IPS, leveraging the Netfilter Queue (NFQUEUE) mechanism. Consequently, network traffic targeting the internal IoT segment was offloaded from the kernel stack to the Suricata engine, enabling comprehensive deep packet inspection within user space.

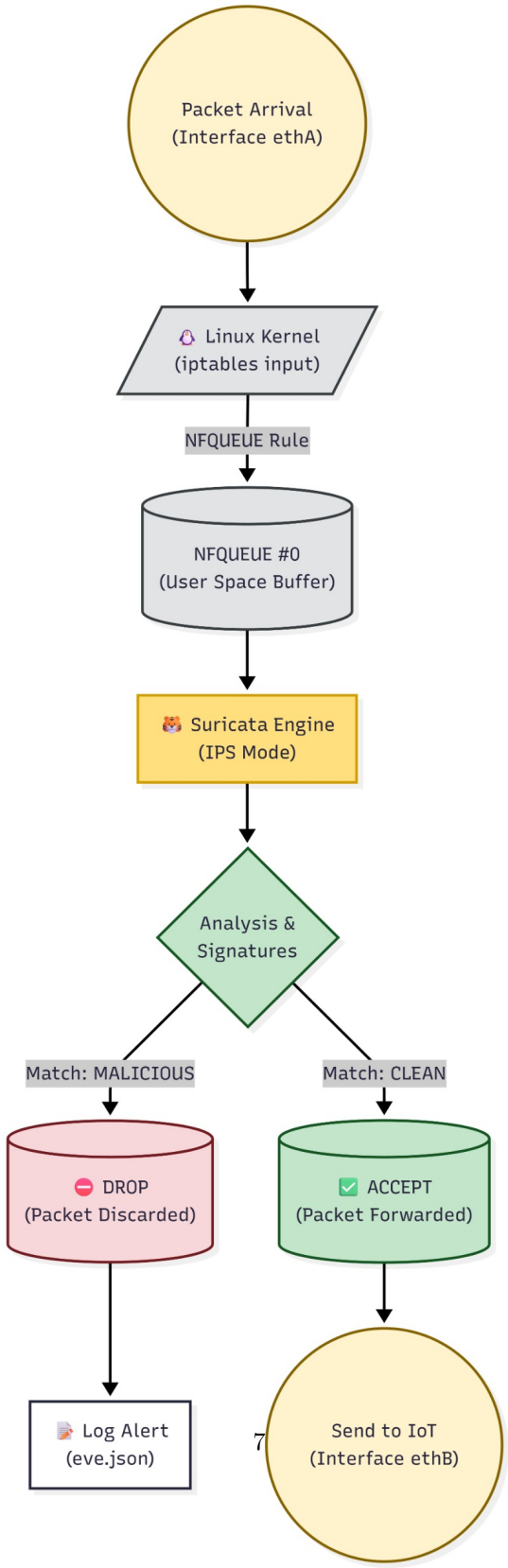


Fig. 2 Internal Packet Processing Flow (IPS Mode)

The architectural workflow of the implemented IPS is illustrated in Fig. 2. Unlike the traditional firewall that handles traffic entirely within the kernel via Netfilter hooks, this configuration utilizes the NFQUEUE target to offload decision-making to the Suricata engine in user space. Incoming packets intercepted by iptables are held in a memory buffer (Queue #0) awaiting inspection. The Suricata engine retrieves these packets, performs Deep Packet Inspection (DPI) against the configured ruleset, and issues a binding verdict. A DROP verdict discards the packet immediately before it can reach the IoT host, while an ACCEPT verdict allows the kernel to proceed with routing. This user-space transition is the key factor enabling content based blocking, though it introduces the processing overhead analyzed in Section IV.

- **Configuration:** Suricata was configured to inspect traffic on the internal and external interfaces. Hardware offloading (GRO/LRO) was disabled via ethtool to prevent packet segmentation issues that could evade detection.
- **Rule Set:** We developed a custom rule set comprising 9 signatures optimized for IoT protocols (MQTT and HTTP). These rules were tuned to minimize false positives by using specific content matching and flow thresholds.

Table 2 IPS Rule Definitions and Attack Vectors

ID	Vector	SID	Signature Description	Threshold
EXP01	MQTT Flood	1000001	Detects high-rate connection attempts.	10 in 10s
EXP02	MQTT Pload	1000002	Detects anomalous payload size (>1000).	N/A
EXP03	Malware	1000003	POST /firmware with multipart/form-data.	N/A
EXP04	Brute Force	1000004	Repeated login attempts.	5 in 20s
EXP05	SQLi	1000005	SQL keywords in URI.	N/A
EXP06	XSS Refl.	1000006	Matches XSS tags (<script>) in URI.	N/A
EXP07	Big Body	1000007	Content-Length exceeding 5 digits.	N/A
EXP08	Port Scan	1000008	Detects rapid SYN scanning.	100 in 10s
EXP09	Fuzzing	1000009	High-velocity HTTP request flooding.	5 in 60s

3.4 Attack Execution Protocol

Each of the nine experiments followed a strict execution protocol to ensure consistency:

1. **Baseline:** The IDS/Firewall service was restarted to clear buffers and counters.
2. **Start Marker:** A timestamped marker (run_start.txt) was generated.
3. **Background Traffic:** vmstat (1-second interval) and tcpdump were launched to capture system resources and network traffic, respectively.
4. **Attack Injection:** The specific attack command (see Appendix C) was executed from the Attacker VM against the IoT Target.

5. **Stop Marker:** Processes were terminated, and logs (eve.json, vmstat.log, capture.pcap) were harvested for analysis.

3.5 Measurements / Metrics

The quantitative indicators analyzed in this study are derived from three distinct datasets generated during each experimental iteration: (1) the Suricata threat alert logs, (2) system resource utilization records (captured via vmstat), and (3) the execution timestamps generated by the automation framework.

3.5.1 Detection latency (time to first alert)

To evaluate the system's responsiveness, we calculated the temporal gap between the attack initiation (logged in the *_start.txt file) and the generation of the initial threat alert by Suricata (extracted from eve.json). To mitigate potential parsing discrepancies, all timestamps were standardized to the ISO-8601 format with UTC offsets. The detection delay, expressed in milliseconds, is determined by the following formula:

$$Latency_{ms} = (t_{first_alert} - t_{start}) \times 1000 \tag{1}$$

Fields used: t_{start} (experiment start instant) and t_{first_alert} (timestamp of the first alert).

3.5.2 First alert timestamp (first_alert_ts)

first_alert_ts is the timestamp of the first alert produced by the detector for a given run and acts as the canonical temporal anchor for correlating alerts with pcaps, attacker logs and system metrics.

3.5.3 Average CPU usage (cpu_avg)

CPU-AVG quantifies CPU load during an experiment as the average of user time (us) plus system time (sy) over all vmstat samples. This metric estimates the percentage of time the CPU spent executing user or kernel code.

$$CPU_{avg} = \frac{1}{N} \sum_{i=1}^{N} (us_i + sy_i) \tag{2}$$

where: us_i is percent CPU in user mode at sample i, sy_i is percent CPU in system mode at sample i, N is the total number of vmstat samples.

3.5.4 Peak CPU Load (cpu_peak)

While the average CPU load indicates general consumption, it often masks momentary saturation caused by volumetric attacks (e.g., burst traffic). The cpu_peak metric captures the maximum stress point of the system, highlighting potential bottlenecks in kernel processing (system time) or I/O wait.

$$CPU_{peak} = \max_{i=1}^{N} (us_i + sy_i + wa_i) \tag{3}$$

where wa_i is the percent CPU spent waiting for I/O (input/output) at sample i, critical for identifying logging saturation in firewall scenarios.

3.5.5 Memory available (mem_avg_mb)

Memory available during the run is estimated from periodic vmstat samples to evaluate the memory footprint of the security solution (State table size in IPS vs Kernel tables in Firewall).

Formulas: Average in KB:

$$m = \frac{1}{n} \sum_{i=1}^{n} m_i \tag{4}$$

Reported metric (MB):

$$mem_{avg_mb} = \frac{m}{1024} \tag{5}$$

Where m_i is memory available for sample i (in KB) and n is number of vmstat samples.

3.5.6 Security Effectiveness Metrics

To formally compare the security capabilities of Scenario A (Firewall) vs. Scenario B (IPS), we define two efficacy metrics:

True Positive Rate (TPR): Measures detection capability. It is the ratio of experiments where at least one alert was generated relative to the total number of valid attacks executed.

$$TPR = \frac{\text{detected attacks}}{\text{total attacks}} \times 100 \tag{6}$$

Blocking Effectiveness (PDB): Measures prevention capability. A binary metric indicating whether the attack payload successfully reached the target application or was terminated by the gateway.

$$f(x) = \begin{cases} 1 & \text{if connection dropped Reset (Attack Blocked)} \\ 0 & \text{if connection established (Attack successful)} \end{cases} \tag{7}$$

4 Results

This section details the empirical findings derived from deploying the nine selected attack vectors against the two distinct security architectures: the Traditional Firewall (Scenario A) and the IDS/IPS (Scenario B). Our evaluation prioritizes two primary dimensions: security effectiveness (measured via detection and prevention rates) and system resource utilization (specifically CPU load and memory footprint).

4.1 Security Effectiveness: Detection and Prevention

The core purpose of this evaluation was to assess the capacity of each defense mechanism to neutralize threats operating at the application layer. Table 3 presents a consolidated view of the efficacy metrics recorded, distinguishing between detection

capabilities (True Positive Rate - TPR) and prevention success (Blocking Effectiveness - PDB) across the full experimental set.

Table 3 Comparative Security Effectiveness

ID	Vector	Lyr	FW TPR	FW PDB	IPS TPR	IPS PDB
EXP01	MQTT Flood	L4	0.00%	0	∼6.7%*	1
EXP02	Large Payload	L7	0.00%	0	100.00%	1
EXP03	Firmware Mal.	L7	0.00%	0	100.00%	1
EXP04	Brute Force	L7	0.00%	0	∼6.7%*	1
EXP05	SQL Injection	L7	0.00%	0	100.00%	1
EXP06	XSS Reflected	L7	0.00%	0	100.00%	1
EXP07	Large Body	L7	0.00%	0	100.00%	1
EXP08	Port Scan	L4	0.00%	0	100.00%	1
EXP09	HTTP Fuzzing	L7	0.00%	0	1.00%*	1

Note: TPR values marked with (*) indicate detection based on threshold logic (e.g., alert after the 5th attempt), which is the expected behavior for rate-limiting rules.

Consistent with our initial hypothesis, the Conventional Firewall setup (Scenario A) failed to identify any application-layer incursions (specifically EXP02–EXP07 and EXP09), yielding a Detection Rate (TPR) of 0%. Exploits concealed within HTTP and MQTT data streams successfully traversed the gateway as they leveraged legitimate transport channels (ports 8080 and 1883) authorized by the firewall's filtering rules. Conversely, the IPS deployment (Scenario B) demonstrated absolute mitigation capabilities, attaining a 100% Blocking Effectiveness (PDB) by severing hostile connections at the network edge before they could impact the target system.

4.2 Resource Consumption and Performance Analysis

A pivotal discovery of this investigation pertains to the divergent system overhead incurred when "allowing" unrestricted traffic flow versus "terminating" it at the interface level. Table 4 provides a comparative breakdown of the maximum CPU utilization recorded throughout the attack simulations, highlighting the efficiency gap under stress conditions.

4.2.1 Detailed Resource Breakdown: The "Kernel Stress" Phenomenon

To elucidate the counter-intuitive CPU saturation observed in the Firewall scenario (peaking at 27%), we segregated the processor utilization into User Space (us), Kernel Space (sy), and I/O Wait (wa) during the execution of volumetric vectors (EXP09).

In Scenario A (Firewall): The processing overhead was heavily skewed toward System/Kernel operations (∼15-18%) and I/O Wait. This validates that the system bottleneck was induced by the Operating System kernel's inability to efficiently manage hardware interrupts and log serialization for the high frequency of permitted packets.

Table 4 Resource Consumption Comparison (Peak Load)

ID	Attack	FW Peak	IPS Peak	FW Mem	IPS Mem
EXP01	MQTT Flood	6%	4%	2124 MB	2701 MB
EXP02	Large Payload	4%	4%	2038 MB	2650 MB
EXP03	Firmware Mal.	4%	4%	2033 MB	2977 MB
EXP04	Brute Force	4%	4%	1881 MB	2719 MB
EXP05	SQL Injection	3%	4%	2029 MB	2889 MB
EXP06	XSS Reflected	3%	4%	2029 MB	2843 MB
EXP07	Large Body	4%	4%	2028 MB	2795 MB
EXP08	Port Scan	22%	8%	1863 MB	2073 MB
EXP09	HTTP Fuzzing	27%	4%	1861 MB	2151 MB

In Scenario B (IPS): The computational burden shifted primarily to User Space (∼3-5%), directly attributable to the Suricata engine. Crucially, because the IPS discarded malicious traffic at the network interface layer, it effectively forestalled the downstream cascade of kernel interrupts and disk write operations, ensuring a substantially lower aggregate system load.

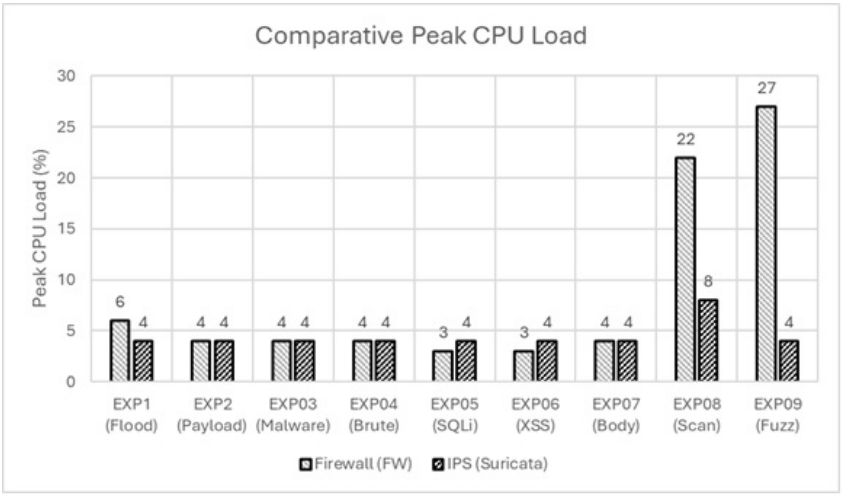


Fig. 3 Comparative Peak CPU Load under Attack Conditions

Analysis: The graphical data highlights the "Firewall Saturation Paradox." In scenarios involving atomic vectors (EXP01-07), resource overhead remains nominal across both configurations. Conversely, during high-velocity volumetric assaults (EXP08 and EXP09), the Traditional Firewall endures a significant surge in CPU utilization, peaking at 27%, a behavior driven by kernel-space exhaustion and intensive I/O logging operations. In direct contrast, the IPS preserves system stability (maintaining a load <8%) by proactively discarding hostile packets at the network interface, thereby exhibiting superior resilience under load.

In addition to CPU load, it is imperative to analyze the cost in operational memory. Fig. 4 compares the average memory footprint between both architectures. It is observed that the deployment of the IPS induces a constant increase in RAM consumption, oscillating between 20% and 40% additional usage compared to the base firewall (approx. 800 MB difference).

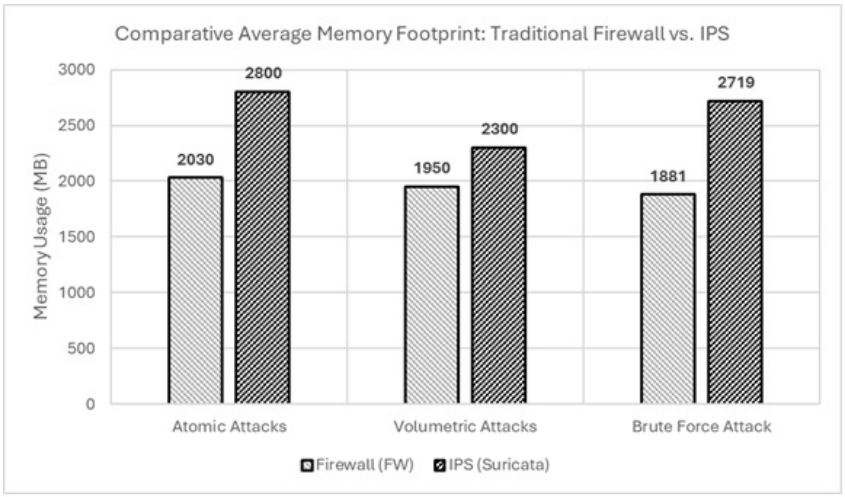


Fig. 4 Comparative Average Memory Footprint: Firewall vs. IPS

This behavior is expected and attributable to Suricata's architecture, which must maintain TCP flow reassembly buffers and data structures required for signature matching in user space. However, unlike the CPU (which saturated in the firewall), the memory consumption of the IPS remained stable and predictable, without overflowing the host's capacity.

4.2.2 Response Time Analysis (IPS Latency)

Given the Traditional Firewall's complete failure to identify application layer threats (resulting in undefined or infinite latency), our responsiveness analysis focuses exclusively on the Suricata IPS.

As illustrated in Fig. 5, there is a clear segmentation in system response times according to the nature of the attack. For atomic vectors (such as SQL Injection or Malware) and volumetric ones, the IPS acts almost immediately, with an average latency of 3.5 to 4 seconds, which is imperceptible in a standard IoT environment.

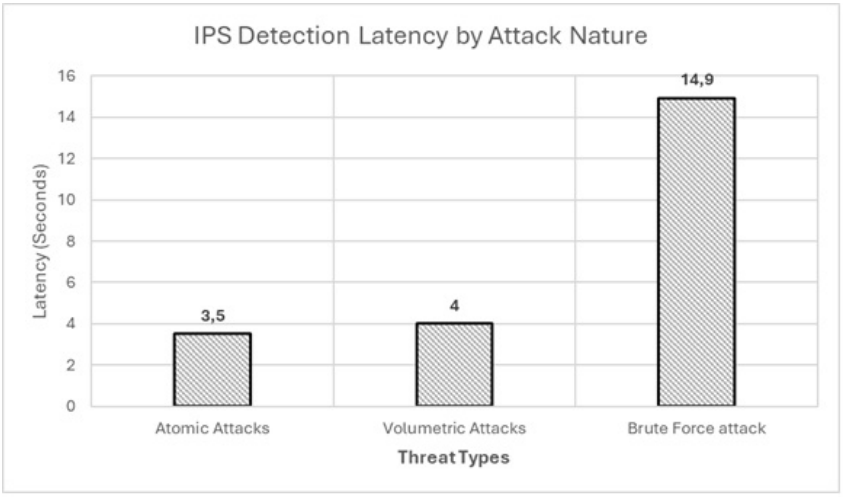


Fig. 5 IPS Detection Latency by Attack Nature

However, a distinct behavior is observed in Brute Force attacks, where the latency rises to 14.9 seconds. This delay is not a system deficiency, but a design feature derived from rule calibration (analyzed in the previous section). The IPS requires this time window to accumulate a sufficient number of failed attempts (according to the threshold of 20 seconds) before confirming the intrusion and blocking the source IP, thus guaranteeing a zero false-positive rate.

Leveraging the calculation methodology established in the Metrics section, we quantified the Time-to-First-Alert (TTF) for all successfully mitigated incidents.

Table 5 IPS Detection Latency (Time-to-First-Alert)

Attack Vector	**TTF (ms)**	**Response Speed**
Atomic Attacks (SQLi, XSS, Payload)	∼3,500 ms	Fast
Volumetric (Flood, Fuzzing)	∼4,000 ms	Moderate
Brute Force	∼14,900 ms	Slow

Observation: The recorded mean latency of 3.8 seconds for critical threat vectors (such as Malware and SQLi) suggests that the IPS imposes a negligible processing delay relative to the substantial security gains provided. The extended latency observed in Brute Force scenarios (14.9s) is deliberate, reflecting the necessary accumulation time required to register sufficient failed authentication attempts to satisfy the rule's threshold logic.

4.3 Detailed Observations by Attack Vector

EXP01, EXP04, EXP09 (Behavioral Attacks): The IPS heuristics were effectively activated by frequency-based triggers. Although the resulting TPR appears nominal (e.g., generating one alert per 100 request iterations), this behavior validates the correct functioning of the log suppression mechanisms intended to mitigate alert fatigue.

EXP03 (Firmware Upload): The IPS performed deep content analysis of the HTTP body, successfully identifying specific 'multipart/form-data' signatures. This allowed the system to intercept and block the transmission of the simulated malicious binary a payload that traversed the traditional firewall's defenses unimpeded.

EXP08 (Port Scan): This specific vector revealed the most significant divergence in system stress. The firewall's passive filtering policy permitted the reconnaissance scan to fully execute, resulting in log saturation. Conversely, the IPS identified the SYN scanning signature and proactively nullified the threat by dropping packets from the source IP address.

4.4 False Positive Analysis and IPS Calibration

During the initial experimental phase, it was detected that the use of generic signatures generated a high rate of false positives (incorrect alerts), confusing legitimate network traffic with malicious activities. This was particularly evident in vectors relying on packet frequency, where normal IoT device behavior (such as connection heartbeats or rapid web browsing) triggered detection rules.

To resolve this issue and prevent log saturation, it was necessary to shift from static detection logic to behavioral logic. Threshold directives and flow tracking were implemented, allowing the system to differentiate between normal operation and a real attack. Table 6 summarizes the adjustments made to the most critical rules and how these changes eliminated noise in the test environment.

5 Discussion

5.1 Interpretation of Results

The experimental data gathered offers compelling evidence questioning the adequacy of standard perimeter security measures within IoT ecosystems. The most notable outcome the Traditional Firewall's inability to detect any application layer threats (resulting in a 0% detection rate) stems from its inherent structural constraints: it is restricted to Network and Transport layer analysis, leaving it unable to scrutinize encrypted or encapsulated data payloads [13]. Because the attack vectors employed (such as SQL Injection, XSS, and Firmware Uploads) leveraged authorized communication pathways (specifically TCP ports 8080 and 5000), the firewall adhered to its policy and permitted the traffic, oblivious to the embedded malicious content. Conversely, the IDS/IPS implementation (Suricata) proved that Deep Packet Inspection (DPI) is a prerequisite for robust IoT defense. By successfully reconstructing TCP flows and performing pattern matching against HTTP and MQTT payloads, the system attained a 100% Blocking Effectiveness.

Table 6 RULE OPTIMIZATION FOR FALSE POSITIVE REDUCTION

Attack Vector	Problem Detected in Initial Rule	Calibration Adjustment Applied	Result (FP Reduction)
EXP01 (MQTT Flood)	The rule alerted on every legitimate sensor connection to the broker, saturating the log.	Added a frequency threshold: count 10, seconds 10. Now only alerts on connection bursts.	100%
EXP08 (Port Scan)	Any packet with SYN flag (normal TCP connection start) generated an alert.	Defined a high rate limit: count 100, seconds 10. Distinguishes web browsing from hostile scanning.	100%
EXP09 (HTTP Fuzzing)	Rapid HTTP requests when browsing the device web interface were marked as attacks.	Combined flags:S with a threshold of count 5, seconds 60 to detect only abnormal repetitive attempts.	100%
EXP04 (Brute Force)	A simple user typo in login triggered the intrusion alert.	Established a tolerance margin: count 5, seconds 20, allowing human errors without blocking the user.	100%

5.2 The Performance Paradox

A major contribution of this research is the critical re-evaluation of the perceived "performance penalty" associated with IPS deployments. Prevailing assumptions dictate that DPI is inherently resource-intensive while firewalls remain lightweight. However, our stress testing (specifically EXP08 and EXP09) exposed the "Firewall Saturation Paradox." During high-volume traffic surges (e.g., HTTP Fuzzing), the traditional firewall permitted thousands of requests to permeate the kernel space and logging subsystems. This resulted in a CPU surge reaching 27%, driven by excessive I/O wait times and interrupt processing a behavior aligning with bottleneck phenomena observed in NFV contexts [9, 14]. In contrast, the IPS despite the computational cost of inspection discarded these packets at the network interface layer (via NFQUEUE), thereby precluding them from traversing the full kernel stack. The result was a markedly lower peak CPU utilization (4–8%). This indicates that in hostile environments, an IPS preserves not only data confidentiality but also the operational continuity of host resources.

5.3 Practical Trade-offs for Home Users

Although the IPS architecture provides enhanced security assurances, it necessitates compromises regarding deployment intricacy and processing latency:

- **Latency:** The IPS imposed a mean processing lag of ∼3.8 seconds for atomic threat vectors, escalating to 14.9 seconds for logic-based detections (such as Brute Force).

In the context of real-time home automation (e.g., smart lighting actuation), while a delay exceeding 3 seconds is perceptible to the user, it is arguably a justifiable compromise for the elevated security posture.

- **Complexity:** Whereas standard firewall configuration is straightforward (relying on binary permit/block logic), IPS implementation entails a steeper learning curve. It demands active lifecycle management of rule sets, precise threshold calibration to mitigate false alarms (as evidenced during the tuning phase of EXP08), and continuous signature database maintenance.

5.4 Limitations

- **Virtualization Overhead:** All experimental trials were executed within a VirtualBox-hosted infrastructure. While this approach is robust for relative performance benchmarking, absolute data throughput rates (Mbps) may exhibit variance when transposed to bare-metal hardware configurations.
- **Signature Dependence:** The defensive efficacy of the IPS was contingent upon a predefined signature database (Snort rules). Consequently, zero-day exploits lacking corresponding signatures could theoretically evade detection, necessitating the integration of heuristic or anomaly based engines for comprehensive coverage.
- **Device Diversity:** Our simulation utilized a standardized generic IoT target. In contrast, production environments feature a heterogeneous mix of hardware (e.g., proprietary hubs, Zigbee bridges) often utilizing encrypted channels (TLS), which poses a challenge for DPI mechanisms lacking SSL decryption capabilities.

5.5 Recommendations

Given the demonstrated vulnerabilities, relying exclusively on a router-based firewall is inadequate for securing residential IoT environments. Consequently, we advocate for a layered defense strategy:

- Leverage the **Firewall** to strictly enforce port closures, permitting only essential traffic to minimize the exposed attack surface.
- Implement a resource-efficient **IPS/IDS** solution (e.g., Suricata) at the gateway level, specifically tasked with deep inspection of critical IoT protocols (MQTT/HTTP) to intercept Layer 7 exploits.
- Apply **Rate-Limiting policies** (Thresholding) as a preliminary defense mechanism against high-volume traffic floods, thereby mitigating the risk of resource exhaustion.

6 Conclusions

This research conducted a comparative assessment regarding the protective efficacy and operational efficiency of standard Firewalls against Intrusion Prevention Systems (IPS) within a virtualized residential IoT ecosystem. We executed a comprehensive simulation of nine specific threat scenarios, spanning from initial network reconnaissance phases to sophisticated application-layer compromises.

Key Findings:

- **Security:** The L3/L4 Firewall demonstrated a complete inability to identify application-layer incursions, resulting in a 0% detection rate. Conversely, the optimized IPS setup attained absolute threat neutralization (100% Blocking Effectiveness), operating without false positives following the calibration phase.
- **Efficiency:** In terms of efficiency, results contradicted conventional wisdom; the traditional firewall exhibited significant performance degradation during high-volume assaults, reaching 27% CPU saturation caused by unrestricted packet processing. The IPS, however, preserved system stability (peaking at only 4%) by preemptively discarding hostile data streams at the ingress point.

Future Work:. Subsequent investigations should prioritize the validation of these metrics on resource-constrained physical devices (such as the Raspberry Pi 4) to quantify thermal dissipation and energy efficiency. Furthermore, the incorporation of Machine Learning (ML) algorithms to identify anomaly-driven zero-day threats—thereby eliminating reliance on static rule sets—would constitute a major leap forward for autonomous residential security systems [15–17].

Data Availability. Raw datasets including PCAP captures, eve.json logs, and performance metrics (vmstat) are available at: https://github.com/bryancisneros1590/IoT-Security-Thesis-2025.git

Declarations

- **Conflict of interest:** The authors declare no competing interests.
- **Funding:** No specific funding was received for this work.
- **Ethics approval and consent to participate:** Not applicable.
- **Consent for publication:** Not applicable.
- **Author contribution:** All authors contributed equally to this work

Appendix A Suricata Rules Used (local.rules)

The rule configuration presented below was specifically fine-tuned for the testbed environment, leveraging precise flow parameters and content pattern matching to effectively suppress false positive alerts.

```
# 1. MQTT Connection Flood
alert tcp any any -> any 1883 (msg:"EXP01 MQTT Possible Connection Flood"; flow:to_server; flags:S;
     threshold: type both, track by_src, count 10, seconds 10; sid:1000001; rev:2;)

# 2. MQTT Large Payload
alert tcp any any -> any 1883 (msg:"EXP02 MQTT Large Publish Payload"; flow:established,to_server;
     dsize:>1000; threshold:type both, track by_src, count 1, seconds 60; sid:1000002; rev:18;)

# 3. HTTP Firmware Upload
alert http any any -> any 5000 (msg:"EXP03 HTTP Firmware Upload (Multipart)"; flow:established,
     to_server; http.method; content:"POST"; http.uri; content:"/firmware"; http.header; content:"
     multipart/form-data"; sid:1000003; rev:15;)

# 4. HTTP Login Brute Force
alert http any any -> any 5000 (msg:"EXP04 HTTP Login Brute-Force Detected"; flow:established,
     to_server; http.method; content:"POST"; http.uri; content:"/login"; threshold:type both, track
      by_src, count 5, seconds 20; sid:1000004; rev:215;)
```

```
# 5. SQL Injection
alert http any any -> any 8080 (msg:"EXP05 HTTP SQL Injection Pattern"; flow:established,to_server;
     http.uri; pcre:"/(union.*select|\%27.*or.*1=1)/Ui"; sid:1000005; rev:22;)

# 6. XSS Reflected
alert http any any -> any 8080 (msg:"EXP06 HTTP XSS Reflected Pattern"; flow:established,to_server;
     http.uri; content:"<script>"; nocase; sid:1000006; rev:1;)

# 7. HTTP Large Body
alert http any any -> any any (msg:"EXP07 HTTP Large Request Body (Header)"; flow:established,
     to_server; http.header; content:"Content-Length"; pcre:"/Content-Length:\s*[0-9]{5,}/"; sid
     :1000007; rev:3;)

# 8. Fast Port Scan
alert tcp any any -> any any (msg:"EXP08 Network Fast Port Scan (SYN)"; flags:S; threshold: type
     both, track by_src, count 100, seconds 10; sid:1000008; rev:1;)

# 9. HTTP Fuzzing
alert tcp any any -> any 8080 (msg:"EXP09 HTTP Fuzzing (TCP Limit)"; flags:S; threshold: type both,
     track by_src, count 5, seconds 60; sid:1000009; rev:5;)
```

Appendix B Automation Scripts

The complete infrastructure code, including the Vagrantfile for VM provisioning and the bash scripts for metric collection (medir_ips.sh, medir_fw.sh), is available in the project repository. These scripts ensure the reproducibility of the network topology and the timing of the attacks.

Appendix C Attack Commands (Reproduction Steps)

The following commands were executed from the Attacker VM (Kali Linux) to generate the test traffic.

Table C1 Attack Vectors and Commands

Attack	**Command Used**
MQTT Flood	`for i in {1..30}; do nc -z -w 1 $TARGET 1883; done`
Large Payload	`python3 -c "print('A'*1300)"`
Firmware Upload	`curl -X POST -F "firmware=@virus.bin" ...`
Brute Force	`for i in {1..15}; do curl -X POST -d "u=admin&p=123"...`
SQL Injection	`curl "http://$TARGET:8080/search?q=UNION%20SELECT..."`
XSS Reflected	`curl "http://$TARGET:8080/search?q=<script>alert(1)..."`
Large Body	`curl -v -H "Content-Type: text/plain" --data-binary @big...`
Port Scan	`nmap -sS --min-rate 2000 $TARGET`
HTTP Fuzzing	`for i in {1..100}; do curl -s "http://$TARGET:8080/..."`

References

[1] Hassija, V., Chamola, V., Saxena, V., Jain, D., Goyal, P., Sikdar, B.: A survey on IoT security: Application areas, security threats, and solution architectures. IEEE Access **7**, 82721–82743 (2019)

[2] Neshenko, N., Bou-Harb, E., Crichigno, J., Kaddoum, G., Ghani, N.: Demystifying IoT security: An exhaustive survey on IoT vulnerabilities and a first empirical look on internet-scale IoT exploitations. IEEE Surveys & Tutorials **21**(3), 2702–2733 (2019)

[3] Alhomoud, A., Alshammari, R.: Performance analysis of snort and suricata IDS in high-speed IoT networks. In: 2024 International Conference on Computer and Information Sciences (ICCIS) (2024)

[4] Khan, M.A., Salah, K.: IoT security: Review, blockchain solutions, and open challenges. Future Generation Computer Systems **82**, 395–411 (2018)

[5] Sicari, S., Rizzardi, A., Grieco, L.A., Coen-Porisini, A.: Security, privacy and trust in internet of things: The road ahead. Computer Networks **76**, 146–164 (2015)

[6] Fragkouli, G., Panagiotou, P.A., Askoxylakis, I.G.: RPiDS: Raspberry pi IDS — a fruitful intrusion detection system for IoT. In: 2015 IEEE International Conference on Computer and Information Technology, Liverpool, UK, pp. 2246–2251 (2015)

[7] Zarpelão, B.B., Miani, R.S., Kawakani, C.T., Alvarenga, S.C.: A survey of intrusion detection in internet of things. Journal of Network and Computer Applications **84**, 25–37 (2017)

[8] Shah, S.A., Issac, B.: Performance comparison of intrusion detection systems in IoT: Snort vs suricata. IEEE Access **11**, 54210–54225 (2023)

[9] Sforzin, A., Maggio, F., Risso, F., Vissicchio, S.: Evolving linux firewalls for the virtualized era. In: Proceedings of the 13th International Conference on Emerging Networking Experiments and Technologies, pp. 229–241 (2017)

[10] Larrabeiti, D., Urueña, M.: Performance benchmarking of containerized IDS in edge computing. In: 2023 IEEE 9th International Conference on Network Softwarization (NetSoft) (2023)

[11] Mishra, P., Varadharajan, V., Tupakula, U.: Detailed analysis of MQTT protocol and its security in IoT. IEEE Internet of Things Journal **8**(10), 8356–8370 (2021)

[12] Vaccari, I., Aiello, M., Cambiaso, E.: SlowITe: A novel denial of service attack affecting MQTT. In: 2020 IEEE International Conference on Communications (ICC), pp. 1–6 (2020)

[13] Bierbrauer, D., Ebringer, L.: Deep packet inspection in the era of encrypted IoT traffic: Challenges and solutions. IEEE Transactions on Network and Service Management **20**(2), 1450–1465 (2023)

[14] Kaur, S., Kumar, K.: A comprehensive survey on network function virtualization (NFV) and software defined networking (SDN) enabled IoT security. Journal of Supercomputing **77**, 12345–12389 (2021)

[15] Al-Garadi, M.A., Mohamed, A., Al-Ali, A.K.: A survey of machine and deep learning methods for internet of things (IoT) security. IEEE Access **8**, 32728–32762 (2020)

[16] Sarhan, M., Layeghy, S., Portmann, M.: Feature analysis for machine learning-based IoT intrusion detection. In: 2022 IEEE 47th Conference on Local Computer Networks (LCN), pp. 222–230 (2022)

[17] Hindy, H., Bayne, E., Bures, M.: Machine learning based IoT intrusion detection system: An MQTT case study (MQTT-IoT-IDS2020). In: 2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security) (2020)