Raspberry-Pi based IDS for IoT

TOGHRUL GARALOV, University of Twente, The Netherlands
SUPERVISOR: DR.ING. M. ELHAJJ (MOHAMMED)

Abstract - As the number of IoT devices increases, protecting them from cyber attacks has become increasingly challenging. Traditional IDS solutions may not be practical for IoT devices due to their limited resources. Therefore to address this, in this research, we aim to propose and evaluate the effectiveness of a Raspberry Pi-based IDS for securing IoT devices. The study will analyze the Pi-based IDS for detecting various network attacks, aim to identify configuration changes for performance improvement and investigate any potential vulnerabilities in the IDS. The methodology involves carrying out attacks on the IoT environment, consisting of a Raspberry Pi(RPi) with IDS, a temperature sensor sending data to a web server and a background traffic generator. Subsequently, the metrics data from log files are analyzed and presented. The expected outcome of this research is to demonstrate the efficacy and offer insights into the possible advantages of utilizing an RPi-based IDS in detecting network attacks for securing IoT devices. The study's findings will contribute to IoT security literature and provide areas of needed improvement to optimize Snort IDS on the RPi.

Additional Key Words and Phrases: Intrusion Detection System, IDS, IoT, Snort, Raspberry Pi, DoS, Nmap, Brute Force, Spoofing

1 INTRODUCTION

In recent years, the proliferation of Internet of Things(IoT) devices has been remarkable, with their integration into various aspects of daily life, such as smart homes, smart lamps, smartwatches, industrial automation, healthcare, and more [2]. This widespread adoption of IoT has provided organizations with opportunities to collect vast amounts of data and automate numerous processes, leading to increased efficiency, productivity, and innovation. However, this expansion has also brought forth new security challenges, primarily due to the inherent vulnerabilities of IoT devices, their limited resources, and frequent connections to the internet [3]. To address these security concerns, this research evaluates a cost-effective and efficient security system: an RPi-based Intrusion Detection System(IDS). The IDS aims to provide robust protection for IoT devices against various factors and attacks, ensuring their security and integrity.

Intrusion Detection Systems are capable of analyzing network traffic in real-time to identify suspicious and potentially malicious activity through predefined rules or machine learning algorithms. For this study, we employ Snort, an open-source IDS known for its high customization capabilities, reliability, and active community support. The research will specifically apply the IDS to an RPi, a low-cost, compact-sized single-board computer popularly used in many IoT applications. By leveraging the capabilities of the RPi, IDS's effectiveness and suitability in protecting IoT devices will be assessed by conducting comprehensive evaluations and testing across different attack scenarios.

TScIT 39, July 7, 2023, Enschede, The Netherlands

Recently, the security of IoT devices has become a significant concern due to the sensitive data they carry and their vulnerability to cyber-attacks. Traditional IDSs are not practical or efficient solutions for IoT devices due to their limited computing power, memory, and battery life. Several surveys and research studies have highlighted the limitations of traditional security solutions for IoT devices, such as cryptographic protocols, secure routing policies, anti-malware solutions, and trust management systems, which consume excessive resources, energy, and bandwidth [4, 6]. There have also been concerns that cloud security solutions are not enough for protecting the IoT as well, specifically against attacks utilizing various levels of communication protocols[7]. Additionally, Blockchain, which is often proposed as a modern security solution, may not be suitable for IoT security due to issues with anonymity, scalability, and performance rates [10].

To address these challenges, IDSs have shown promise in securing IoT devices by analyzing network traffic and identifying malicious activities. However, existing research on IDS performance analysis for IoT devices, particularly using RPi as a platform, is limited. Previous studies have primarily focused on the feasibility and resource usage of RPi-based IDSs, with little emphasis on other performance metrics such as detection rate, false positive rate, and reaction time. Moreover, there is a lack of research exploring the security aspects of RPi IDS, including its susceptibility to exploitation, penetration, and other types of attacks.

Motivated by these research gaps, this study aims to contribute to cyber defense strategies by investigating the security challenges in the IoT technology space. Specifically, we focus on evaluating the effectiveness of a lightweight RPi-based IDS in detecting and logging cyber attacks. Additionally, we analyze the resource usage of RPi 4 as an IDS, providing up-to-date insights compared to previous studies that focused on earlier versions of the device. By sharing our findings, we aim to enhance the understanding of the capabilities and limitations of using RPi devices as IDS solutions in the modern cybersecurity landscape.

The objective of this research is to assess the efficacy of using an RPi 4-based Snort IDS as a security solution for IoT devices across various industries. To achieve this goal, the study addresses the following research questions:

(1) How does the performance of the RPi-based IDS compare to other IDS solutions in terms of detecting rate, false positive rate, and response time for different types of attacks (e.g., DoS, Authentication, MITM, Network scanning) on IoT devices?
(2) What specific configuration modifications can be implemented to improve the effectiveness and security of the proposed RPi-based IDS in identifying and logging cyber-attacks, particularly DoS attacks, on IoT devices?
(3) What are the particular vulnerabilities and exploits that could potentially impact the performance of the proposed RPi-based IDS, and what measures can be taken to mitigate them?

The rest of this paper will be structured as follows. First, In section 2 we detail the setup of our experiments, Attacker machine, and IoT environment to assess the effectiveness of RPi-based IDS in securing IoT devices. Moreover, in Section 3 we present the results achieved from our experiment. Section 4 goes into a critical analysis of our results, address the limitations, and proposes potential solutions to address the vulnerabilities discovered. Additionally, Section 4 performs a comparative analysis with the related works in the field. Finally, section 5 concludes the research, and presents areas of interest for further research.

2 RESEARCH METHODOLOGY

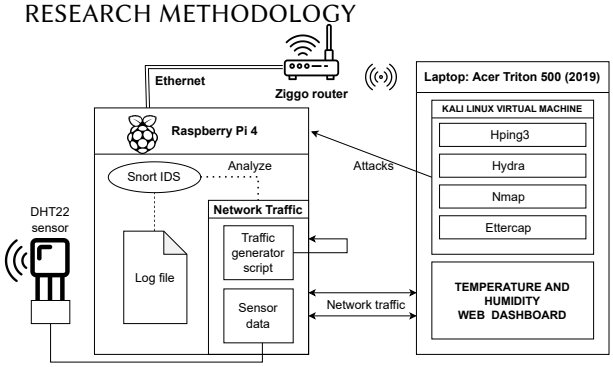

Fig. 1. RPi IDS SCHEMA

To address the research questions, our study encompassed several stages. Firstly, we conducted a comprehensive literature review to gather relevant papers and research on RPI IDS performance evaluations and comparisons. This served as the foundation for our bibliography and provided valuable insights into the existing body of knowledge. Next, we focused on creating an IoT environment that could replicate authentic IoT traffic. To accomplish this, we employed an RPi as our IoT device and utilized a traffic generator script to simulate traffic. Additionally, we incorporated our own temperature and humidity sensor data, which was transmitted to a web dashboard. This allowed us to establish a robust IoT device ecosystem that was well-suited for testing and evaluation. Subsequently, we configured and deployed Snort on RPi to monitor the generated traffic. Snort served as our intrusion detection system, enabling us to analyze and detect potential security threats within the IoT network. To further evaluate the performance and security aspects, we utilized various penetration testing tools and leveraged a Kali Linux Virtual Machine. Through rigorous testing, we subjected the IoT network traffic to different stress scenarios, generating data for multiple performance and security evaluations. The insights gained from these evaluations contributed to a comprehensive understanding of the system's capabilities and vulnerabilities.

An overview of the architecture we constructed for this research can be observed in Figure 1. This visual representation provides a clear depiction of the components and their interconnections within our research environment.

Lastly, we concluded the study by presenting our findings and results, which directly addressed the research questions we had initially formulated. Through a systematic approach and a combination of literature review, experimentation, and analysis, we were able to contribute valuable insights to the field of RPI IDS and IoT security.

2.1 Measurement Environment

2.1.1 Hardware and Software Configuration. The hardware and software components are demonstrated in Tables 1 and 2. In this setup, the RPi 4 functions as both an IoT device and an Intrusion Detection System (IDS) to safeguard itself. To measure temperature and humidity, an RPi is equipped with a DHT22 sensor. The IDS functionality is achieved through the utilization of Snort software, which operates as a signature-based IDS running on the RPi. For data collection and traffic generation, Python3 programming language is employed, along with the Scapy library. Additionally, Python3 is utilized in certain scripts for executing timed attack commands. The primary research codes can be accessed on the RPi IDS Github page[1]. In order to simulate an attacker, Kali Linux is employed, which encompasses a range of attack tools such as Hping3 for Denial of Service (DoS) attacks, Hydra for Brute-Force attacks, Nmap for scanning, and Ettercap for ARP spoof attacks.

In our study, we made certain modifications to the Snort configuration file to tailor it to our requirements. Firstly, we edited the *"HOME-NET"* parameter, which defines the IP address range of our internal network. In our specific case, it was set to "192.168.178.0/24" as we were utilizing a Ziggo router in our home network. Additionally, we included port 5000 in the list of HTTP ports to effectively monitor the Flask web dashboard traffic. To enable the detection of ARP spoofing attacks, we activated the ARP preprocessor in the configuration file and added a list of IP-MAC address pairs. We also incorporated preprocessor rules[2] related to ARP and other preprocessors and enabled the PATH and rule inclusion in the configuration file. The ARP preprocessor in Snort continuously monitors and analyzes ARP traffic, generating alerts when it detects any inconsistencies within the IP-MAC list, indicating ARP spoofing attacks. Furthermore, we customized the snort rules inclusion to align with our research objectives. For several tests, we created our own rules in the "Local.rules" file, specifically tailored to simulate and evaluate the attacks relevant to our research. These configuration procedures were crucial in ensuring that Snort was appropriately configured for our research environment, enabling it to effectively detect and log the specific network threats we were examining.

Table 1. Hardware specifications

Hardware Component	Specifications
Raspberry Pi 4 Model B	64-bit quad-core Cortex-A72 processor, 8GB LPDDR4 RAM, 32 GB microSD storage, 802.11b/g/n/ac wireless, Gigabit Ethernet port, 300 - 400 Mbit/s download and 30 - 50 Mbit/s upload internet speed
DHT22 AM2302 Sensor	Temperature and Humidity Sensor
Acer Triton 500 (2019)	Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz, 16GB DDR4, Killer DoubleShot Pro wireless, 512 GB SSD storage
Kali Linux VM	2GB RAM, 2 Processor cores, vdi SATA virtual 80.09 GB storage (actual 39.03GB)
Router	Ziggo TG2492LG-ZG SmartWifi Modem

[1] https://github.com/Toghrul000/Raspberry-Pi-IDS
[2] https://github.com/eldondev/Snort/blob/master/preproc_rules/preprocessor.rules

Table 2. Main Software Components and Descriptions

Software	Explanation
Raspbian OS	Raspbian GNU/Linux 10 (buster), the OS running inside the RPi
Kali Linux	v2023.2, Attacker VM machine
Snort	Snort v2.9.20 IDS in the RPi for detecting attacks
Flask	v2.3.2, used for running the web dashboard for temperature and humidity data
trafficpy.py	A background traffic generator script
Scapy	v2.5.0, a Python library for crafting your own packets, used in background traffic generation and some custom attacks tested
datetime.sh	A shell script that prints start and end timestamps of the command it is run with
run_for_duration.py	A Python script used for making commands run for a specific duration of seconds
sensor.py	A program for reading data from a sensor and sending it to the dashboard
Hping3	v3.0.0-alpha-2, a tool used for DoS attacks
Nmap	v7.93, a tool used for various types of scan attacks
Hydra	v9.4, a tool used for SSH brute force attacks
Ettercap	v0.8.3.1, a tool used for ARP spoof attacks
Wireshark/tcpdump	Tools used for analyzing network traffic
Metasploit/ExploitDB	Tools used for testing the system against exploits

2.2 Measurement Metrics

To evaluate the effectiveness of the RPi-based IDS, we analyzed several metrics to address our research questions. These metrics included response time, detection rate (Alerts Logged), false positives, packets processed by Snort, Snort's packets analyzed per second (Pkts/sec), percentage of dropped and analyzed packets, CPU and memory usage, and power usage.

We collected some of these metrics directly from the output of the Snort command, while others were manually obtained. For evaluating response time, we utilized our custom script *"datetime.sh"* along with attack commands to capture the start and end timestamps. By comparing these timestamps and the timestamp of the first non-false positive alert, we could measure the response time of the IDS.

To assess the detection rate, we considered both internal and external aspects. The internal detection rate was determined from the Snort command output and represented the ratio of alerts of targeted attacks to the packets processed by Snort, specifically excluding the total number of attacks directed at the Pi. However, this metric often included numerous true negatives. To address this, we examined the external detection rate, particularly for attacks like DoS where the exact number of sent packets was known. We estimated the external detection rate by dividing the number of alerts logged by the number of attacks executed.

To identify false positives, we utilized tools such as *"tcpdump"* and leveraged Snort's built-in analysis capabilities to examine and filter log files and alerts.

For monitoring Snort's CPU and memory usage, we employed the *"top"* command in Linux along with *"grep"*. By periodically executing the command *"top | grep snort"* while Snort was running, we obtained outputs showing the CPU and memory usage of Snort. We plotted these data points on a graph along with the duration Snort was active, as extracted from the Snort output.

To measure the power usage of Snort, we installed and utilized the *"powertop"* command. This command provided insights into metrics such as *Usage* and *Wakeups/s*. The *"Usage"* metric represented the duration for which a specific component actively consumed system resources, indicating the frequency and duration of resource utilization. The *"Wakeups/s"* metric indicated the frequency at which a component triggered wake-up events, potentially impacting power consumption and system performance.

Other metrics such as Packets processed, Pkts/s and Packet I/O were obtained from the Snort output. These metrics were also tested for RQ2, where we made modifications to the Snort configuration file and compared the results with the metrics collected for RQ1. Details of these modifications are provided in the section 3.

In terms of security and vulnerability assessment, we focused on several main metrics. These included determining if a specific attack triggered an alert, testing Snort's resilience against DoS attacks to divert attention from hidden attacks, and evaluating Snort's effectiveness against custom-crafted packets and IPv6 DoS attacks. To conduct these assessments, we utilized tools such as Metasploit and ExploitDB, conducted buffer-overflow attacks, tested fragmented attack packets, and examined Snort's performance against custom-crafted packets and IPv6 DoS attacks.

3 RESULTS

3.1 Answering RQ1

In this section, we present the results of our performance evaluation of RPi IDS against the ICMP, TCP SYN, HTTP floods, SSH Brute Force, ARP spoofing, and Nmap scans. We ran our background traffic (sensor data and traffic script) and each attack for a duration sufficient to observe measurable results. For ICMP, SYN floods and ARP spoofs this was around 5 minutes. For HTTP Flood, Brute Force, and Nmap we let them run until the attack was over. All the attacks were run with our custom script *"datetime.sh"* to print the timestamps of the Start and End of an attack. Thus in the next, section, when which attack commands were used are shown, assume the full commands are executed as follows: *"sudo datetime.sh <COMMAND>"*

There were several general observations and findings we found during the experiment. In Tables 5 and 6, the benchmark results for the main metrics we examined for this study are presented. One of the main findings was related to the packet processing of Snort. we noticed that in all the attacks the Packet processing speed (Pkts/s) is relative to the number of Packets processed. When the processed packets are higher such as more than 2.6 million, the speed it processes can reach as high as around 10000 Pkts/s, sometimes even more. This is especially related to DoS attacks such as ICMP and SYN flood where high volumes of traffic are targeted at Snort. However, for other attacks, that don't generate many instant alert messages and logs, their packet processing speed was much lower. We can also see this in Figure 8, where it is obvious which attacks took the most out of Snort. In addition, according to Table 4 ICMP and SYN floods had the most power usage. Another observation we found was about the Packet drop rates. we noticed that RPi combined with the IDS was not powerful enough to handle all DoS network traffic and always had around 40% packet drop rates.

First, we tested ICMP and SYN Flood attacks against Snort. We ran these attacks in Kali Linux with the Hping3 tool. The commands for these attacks were: *"hping3 <–icmp/-S -p 80> –flood <host>"*. For detecting these attacks we also utilized our own custom snort rules: *"alert icmp any any -> $HOME_NET any (msg:"ICMP flood"; sid:1000001; rev:1; classtype:icmp-event; detection_filter:track by_dst, count 500, seconds 3;)"* and *" alert tcp any any -> $HOME_NET 80 (flags: S; msg:"Possible SYN flood"; flow:stateless; sid:1000002; detection_-filter:track by_dst, count 20, seconds 10;)"*. The first rule is designed to generate an alert when counting 500 ICMP packets within a 3-second timeframe. The second rule monitors TCP traffic from any source to port 80, detecting SYN packets (S flag), and generating an alert when the destination address receives more than 20 SYN packets in 10 seconds. Both rules track by destination (our host) since DDoS is attacked from various and too many source addresses to count. Based on these rules, the response time for the detection of these attacks was very fast by ICMP DoS being detected almost instantly, and SYN flooding in approximately 1 second.

Next, we attempted the slow HTTP flood test. For the purpose of this attack, we needed a better target. Thus, in this attack, we switched the Temperature Web dashboard to be hosted on the Pi instead of the laptop. After that, we performed the attack with the following command: *"slowhttptest -c 1000 -H -g -o slowhttp -i 10 -r 200 -t GET -u http://<host>:5000/ -x 24 -p 3"*. This command initiates a Slowris-style HTTP attack to the host with 1000 connections, using GET requests maximum length of 24 bytes at a rate of 200 requests per second, with 10 seconds waiting for data and 3 seconds probe connection timeouts. Since there wasn't any specific rule against this attack, we employed our rule to monitor port 5000 (Flask web port). The rule *"alert tcp any any -> $HOME_NET 5000 (msg:"FLASK HTTP Flood"; threshold: type threshold, track by_src, count 100, seconds 10; sid:40000056; rev:1;) "* tracks the source and generates an alert when there are more than 100 packets to port 5000 in 10 seconds. This was effective in detecting such attacks since it was able to detect the attack in less than a second.

Afterward, we performed an SSH brute-force attack with the help of Hydra. We set up a wordlist with the help of a *crunch* tool. The list contained 1000 entries, one of which was the password for our Pi. We ran the command: *"hydra -l pi -P wordlist.txt ssh://<host>"*. To project SSH port 22 against this attack we utilized our custom rule: *"alert tcp any any -> $HOME_NET 22 (msg:"SSH Brute Force Attempt"; flow: established,to_server; content:"SSH"; nocase; offset:0; depth:4; detection_filter:track by_src, count 5, seconds 60; sid:10000015; rev:1;)"*. This rule monitors port 22 traffic, looks for the string case-insensitive "SSH" and by tracking the source generates an alert if it counts 5 attempts in 60 seconds. This rule responded to the attack in a second and regularly provided alerts at periodic times, indicating if the brute force is ongoing.

The next attack on our list was the Nmap scans. The performance and resource specifications of the Nmap can be found in Table 6 and Figures 6, 7. In our system we had 4 services running, they were VNC, OpenSSH, FTP and UPnP (Temp. dashboard running on p5000). For this test, we performed numerous port and service scan techniques. The test involved running 12 different scans with *"nmap (-s<S/T/A/W/M/V/C/U>/-PR/-A/-O) <host>"*. Since Nmap involves various types of attacks, we relied on Snort's own default rules for this attack. The response times for each attack were around 18 seconds, with only the UDP ports scan taking 88 seconds. The alerts generated in the attack accurately classified scans as *"Attempted Information Leak"*.

The Last Attack we tried was one of the variations of Man in the Middle Attack, an ARP spoofing. The attacks involve sending forged ARP packets to poison the ARP cache of the host and router into linking the attacker's MAC address with the IP of the other device, resulting in traffic redirection and interception. To perform this Attack we first enabled the network traffic forwarding with *"sysctl net.ipv4.ip_forward=1"* and employed *"ettercap -G"* to open the GUI of Ettercap. After that, we performed the attack by selecting the router gateway and RPi IP as targets. Meanwhile, in Snort all the ARP spoof protection setups were already enabled as described in the section 2.1.1. With a response time of 1 second, IDS was effectively able to identify the attack.

In Tables 5 and 3 you can also see the Detection rates and False positives. Most of the False positives in the attacks were classified as "BAD TRAFFIC" or "Consequetive TCP small segments" resulting from traffic to ports 22 and 5000. These were most likely resulting from our traffic generator script and attacks such as HTTP flood and Brute Force to ports 22 and 5000. Due to some services running on our Laptop, there were also occasional false alerts indicating UPnP service discovery attempts. The most notable False positive was in the SYN flood, where it alerted several incorrect TCP port scans, miscellaneous activity and bad traffic alerts. Furthermore, in terms of detection rates, the results were not as high as expected. The internal detection rate in ICMP flood was good. However, SYN Flood showed around 50% detection. Even without the packet drops in some attacks such as HTTP and SSH BF, external detection rates were low. Since Nmap scans didn't generate as many alerts, for this experiment they showed sufficient detection.

Table 3. False Positive Test Statistics

Attack Type with duration	Num Attacks	Alerts	FP
ICMP Flood (340s)	19264990	3111810	21
TCP Flood (330s)	17545878	2039951	253
HTTP Flood (240s)	1000 connections	1919	372
SSH Brute Force (613s)	90.86 tries/min, 636 tries	393	221
ARP spoof (340s)	unk	36	2
(nmap, -sSTAWM, -PR) (30-50s)	unk	2 per each	0
(nmap -sV) (154s)	unk	2	0
(nmap -A) (177s)	unk	4	1
(nmap -O) (50s)	unk	3	0
(nmap -sC) (62s)	unk	3	1
(nmap -sU) (1076s)	unk	21	6

Table 4. Power Consumption Statistics

Attack Type	Duration	Usage	Wakeups/s
ICMP Flood	320s	9.1 ms/s	0.12
TCP SYN Flood	320s	31.6 ms/s	5.0
HTTP Flood	260s	409.2 us/s	0.04
SSH Brute Force	490s	57.4 us/s	0.07
NMAP Scans	340s	360.3 us/s	0.13
ARP spoof	340s	48.8 us/s	0.09

Table 5. Attack Table V1 RQ1

Attack Type with duration	NumOfAttackPkts	Response time	Alerts	Packets processed	External DR	Pkts/sec	Packets Dropped
ICMP Flood(352s)	10128751	0.077s	2686664(92.724%)	2897481	26.525%	8231	16136076 (45.881%)
TCP SYN Flood(333s)	8973979	1.075s	2971030(46.786%)	6350191	33.107%	19069	11923351 (39.485%)
Slow HTTP Flood(240s)	1000 connections	0.3s	120(0.268%)	44702	12%	160	0%
SSH Brute Force(673s)	601 tries, 85.86 tries/min	1s	177(0.504%)	34974	29%	51	0%
ARP spoof(340s)	unk	1s	36(0.154%)	23358	unk	68	0%

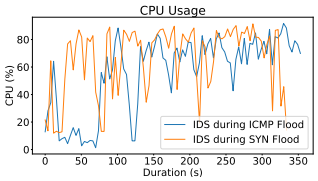

Fig. 2. CPU-ICMP, SYN Flood

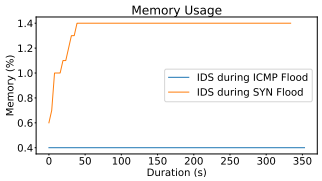

Fig. 3. MEM-ICMP, SYN Flood

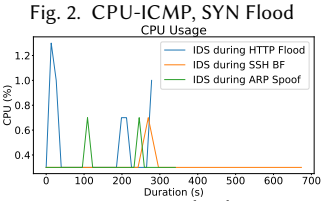

Fig. 4. CPU-HTTP Flood, BF, ARP

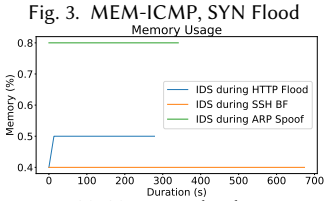

Fig. 5. MEM-HTTP Flood,BF, ARP

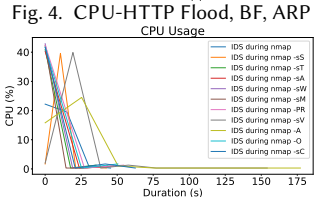

Fig. 6. CPU-Nmap Scans

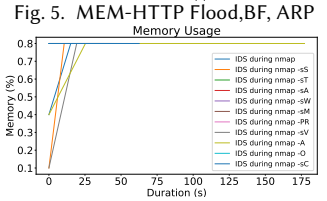

Fig. 7. MEM-Nmap Scans

Fig. 8. CPU and MEM Resource Usages of Snort across different attacks

Table 6. Attack Table V2(Nmaps) RQ1

Attack Type	Response time	Packets processed	Pkts/sec
(nmap)(45s)	18s	3712	82
(nmap -sSTAWM)(35-43s)	(17.8-17.9s)	(3223-3537)	(78-92)
(nmap -PR)(54s)	17.77s	3865	71
(nmap -sV)(154s)	18.18s	7996	51
(nmap -A)(177s)	18.53s	8933	50
(nmap -O)(50s)	17.92s	3802	76
(nmap -sC)(62s)	17.96s	4541	73
(nmap -sU)(1076s)	88.32s	40832	37

3.2 Answering RQ2

In this section, we did multiple tests and modifications on Snort configuration files to try to increase the effectiveness of the IDS in detecting and logging the attacks.

For this experiment, we chose to test Snort against DoS attacks. The main reason why we selected DoS attacks is they generate the most alerts and better Packet I/O data. Thus, we believe they would be the ones that would show if there is any potential improvement in Snort.

In an attempt to improve Snort performance, we have tried several modifications to the Snort config file. Our main goal was to examine if we could make the IDS process and analyze more packets. If more packets were to be analyzed, more alerts could have been generated, thus better detection rate could have been achieved. Initially, we started changing configs in the base detection engine section. The First modification we did was increasing *pcre_match_limit* and *pcre_match_limit_recursion* from 3500 to 5000 and from 1500 to 2000 respectively. These settings specify how many times Snort will try to match a packet to a rule using Perl Compatible Regular Expressions (PCRE) before giving up. By raising these parameters, Snort's capacity to detect complex attacks could get improved. Then, we increased the *max-pattern-len* from 20 to 30 in the *detection* config where *ac-split* search method was defined. The *max-pattern-len* controls the maximum length of the patterns Snort searches. After that, we raised the number of events Snort can handle simultaneously, by increasing *max_queue* in *event_queue* config from 8 to 16. Next, we doubled the maximum amount of data that Snort buffers before discarding it, by changing the value of *paf_max* to 32000. We then moved to the preprocessor configuration section. In this section, the first thing we did was disable all the normalization preprocessors. Since they are more useful in IPS mode, we didn't require them, as our focus was only on IDS mode. Next, we increased 8 times the Snort's ability to handle large amounts of fragmented traffic. By modifying *max_frags* in *frag3_global* preprocessor, we increased the number of fragments Snort can maintain at any given time. In the *stream5_global* preprocessor, we raised the IDS's ability to handle large amounts of TCP and UDP traffic by 4 times, by increasing values of *max_tcp* and *max_udp*. In *http_inspect_server* preprocessor by increasing the values of *chunk_length* by 4 times, and values of *oversize_dir_length, max_header_length, max_headers, max_spaces* by doubling, we enhanced the capability of Snort to handle larger amounts of HTTP traffic. These were the main configuration changes we tried to improve the effectiveness of Snort against our attacks.

We decided the test the IDS performance in 3 ways with the following commands: *"sudo snort -A fast -c /path/to/snort.conf -i eth0 -l /path/to/logfile"*, *"sudo snort -A fast -c /path/to/snort_Modified.conf -i eth0 -l /path/to/logfile, sudo nice -20 snort -A fast -c /path/to/snort_-Modified.conf -i eth0 -l /path/to/logfile"*. First was running the attacks against regular configuration, second was with modified and third was with the highest priority. As a result, surprisingly, none of the modifications or the priority changes we tried had any major effects on the packet processing and performance of Snort. All the performance test results can be seen in Tables 7, 8 and in Figure 15. The notable resource usage difference in test versions was: The second and third commands had more memory usage and the second command had higher power usage.

3.3 Answering RQ3

In this section, we describe the vulnerability assessments we took to find security issues affecting Snort IDS and tested a few attacks against Snort's detection. In Table 9 you can find the attacks that we tested.

First, we explored different kinds of Scan attacks to try to test the IDS. One scanning tool we found to be effective as nuclei. Nuclei is a customizable vulnerability scanning tool that was created in 2020 by *projectDiscovery* an open-source cyber security company.

Table 7. Attack Table RQ2

Attack Type with duration	NumOfAttackPkts	Response time	Alerts	Packets processed	Pkts/sec	Packet Drop	FP
ICMP_FLOOD_CV0(329s)	15417059	1.442s	2425932(99.528%)	2437413	7408	10161861(44.646%)	4
ICMP_FLOOD_CV1(331s)	17111445	1.212s	2674129(99.725%)	2681474	8101	9870959(44.021%)	6
ICMP_FLOOD_CV2(331s)	15522480	0.938s	2574306(99.639%)	2583595	7805	9941963(44.250%)	20
SYN_FLOOD_CV0(327s)	15501579	0.898s	1729377(50.172%)	3446874	10540	8589316(41.644%)	206
SYN_FLOOD_CV1(330s)	16142825	0.672s	1716858(50.188%)	3420864	10366	8610579(41.714%)	159
SYN_FLOOD_CV2(328s)	15870443	0.454s	1760632(51.036%)	3449769	10517	8502459(41.567%)	191
HTTP_FLOOD_CV0(280s)	1000 conn	1.695s	1907(119 relevant)	49481	176	0%	399
HTTP_FLOOD_CV1(273s)	1000 conn	1.411s	2102(120 relevant)	49473	181	0%	397
HTTP_FLOOD_CV2(276s)	1000 conn	1.195s	2011(122 relevant)	49108	177	0%	394

Table 8. Power Statistics for RQ2

Attack Type	Duration	Usage	Wakeups/s
ICMP_FLOOD_CV0	329s	74.1 ms/s	0.08
ICMP_FLOOD_CV1	331s	94.3 ms/s	0.07
ICMP_FLOOD_CV2	331s	74.5 ms/s	0.06
SYN_FLOOD_CV0	327s	66.9 ms/s	0.05
SYN_FLOOD_CV1	330s	82.1 ms/s	0.07
SYN_FLOOD_CV2	328s	69.2 ms/s	0.05
HTTP_FLOOD_CV0	280s	671.6 us/s	0.06
HTTP_FLOOD_CV1	273s	1.1 ms/s	0.07
HTTP_FLOOD_CV2	276s	1.0 ms/s	0.08

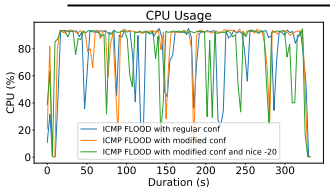

Fig. 9. IDS CPU-ICMP Floods

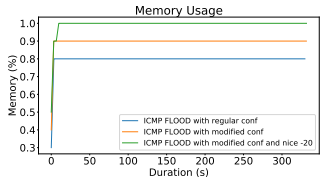

Fig. 10. IDS MEM-ICMP Floods

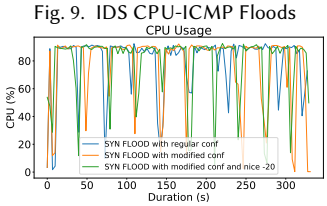

Fig. 11. IDS CPU-SYN Floods

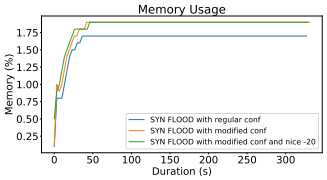

Fig. 12. IDS MEM-SYN Floods

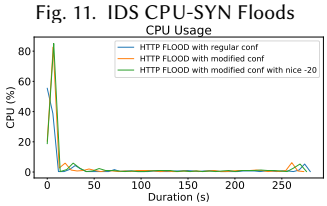

Fig. 13. IDS CPU-HTTP Floods

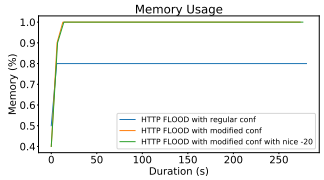

Fig. 14. IDS MEM-HTTP Floods

Fig. 15. IDS CPU and MEM Resource Usages across DoS with diff. Confs

Table 9. Effectiveness of Security Metrics

Security Metric	Detected	Snort Affected
Nuclei Scan	No	No
IPv6 DoS Echo	Yes	No
DoS Attack Overwhelm	DoS Yes, Hidden Attack No	No
Buffer Overflow	Yes	No
Fragmented Buffer Overflow	Yes	No
Exploits	-	No
Special-Crafted Packet	No	No

Its customizability allows you to utilize and create YAML files according to your specific needs. For our security testing, we decided to download and use one of the predefined templates created by their community. The command we ran for testing is as follows: *"nuclei -u <host> -t <path to temp folder/network>"*. Surprisingly, when combined with the network template, this command was able to discover services such as VNC and SSH and their versions in our system. The IDS failed to detect these scans, which suggests a potential vulnerability where attackers could apply nuclei instead of Nmap for covert reconnaissance. It is important to note that nuclei were not as comprehensive as Nmap, as it missed detecting the FTP service. Nevertheless, considering nuclei is new and in ongoing development, it has the potential to match Nmap's effectiveness in service scans in future.

Next, We tested the IDS against IPv6 attacks, to assess its ability to detect IPv6. It is important that Snort detects IPv6 since every year IPv4 addresses are depleting[12] and more companies are starting to transition to IPv6. Thus, we changed the Snort HOME_NET to detect IPv6 as well. The new HOME_NET in our Snort is as follows:

Listing 1. Snort HOME_NET

```
ipvar HOME_NET [2001:1c06:1b0e:2600::/64]
ipvar HOME_NET 192.168.178.0/24
```

To test IPv6 we used *ICMPv6EchoRequest* and created a small custom DoS script targeting the IPv6 address of the Pi with Scapy. we also had to make small count modifications in our ICMP rule for it to detect Python loop DoS. Since Hping3 DoS is much faster than a regular *while true* loop, we modified the ICMP rule as follows:*"alert icmp any any -> any any (msg:"ICMPv6 flood"; sid:1000001; rev:1; classtype:icmp-event; detection_filter:track by_dst, count 50, seconds 3;)"*. In the end, Snort was able to detect this, and thus it confirmed that Snort can be used to protect IoT from regular IPv6 attacks.

Afterwards, we tested the IDS against DoS Overwhelming. We found that when snort is busy processing DoS attack packets, any other attacks made to the same system are undetected by the RPi IDS. While the DoS attack was in progress, we noticed that attacks such as Brute Force and Nmap are being undetected by the IDS. This raised another potential vulnerability for an attacker to perform covert various types of attacks while Snort's attention is diverted.

Later we assessed the RPi IDS against Buffer Overflow and Fragmented Buffer Overflow. Snort with its own default rules was not able to detect the attack packet. However, after we enabled the preprocessor rule set(footnote 2) we added earlier for the ARP spoof, It was able to detect both attacks accurately as some irregular Data on the SYN packet. Additionally, it is worth noting that Snort produced alerts for some small fragments as well, albeit it incorrectly identified them as DoS attempts. Nonetheless, the IDS's ability to detect fragments was a notable milestone in its capacity to identify potential signs of fragmented attacks.

Lastly, we evaluated the IDS against exploits and one specially crafted packet. We utilized *ExploitDB* and *MetaSploit* for testing

exploits and shellcodes targeted at Snort. Since Snort 2.9 is written in C, most of the exploits were attempting to cause segfaults in Snort, which is why we didn't have any entry in the Detection section of Table 9 for Exploits. All the exploits and shellcodes we tested targeted at Snort were useless against the latest Snort version (2.9.20). Out of all the exploits, the exploits for CVE-2009-3641 took our attention the most, since they were using specially crafted ICMPv6 packets. With a little modification, we replicated the custom packet. Originally, this exploit was causing Snort Application to crash. It was fixed in later versions of the Snort (after version 2.8.5). However, for our security assessment, we tested Snort's ability to detect the DoS attack with the crafted packet. As a result, Snort was not able to detect the DoS attack. This discovery introduced another potential vulnerability that could be exploited by attackers to overwhelm the target system undetected.

4 DISCUSSION

This section will discuss limitations, and reasons for some of the main results, address if possible solutions for the vulnerabilities found, and perform a comparative analysis with the other related works in the field.

In this research, we conducted all the experiments using only a single instance of Snort IDS rather than running separate instances for different types of attacks. This choice was driven by the lightweight nature of most real-world IoT applications. We believe usually most IoT manufacturers utilize lesser, more cost-effective boards, such as RPi Zero or Pico in the IoT appliances. These boards have lesser CPU power and thus, it would not be ideal for IDS to hog most CPU power to itself since extra CPU usually goes into other services running inside the IoT devices.

The main significant limitation in this research was the single-threaded nature of the Snort 2 application. We believe this had a substantial impact on our performance results since Snort 2 utilizes one packet thread per process[9]. We believe this combined with other limitations of Snort 2 itself, could also be the reason why our configuration change attempts at improving the performance failed. After all, in the end, we found that the main factors that affect Snort's performance are Hardware resources, Network traffic, Rule sets, and Snort's own internal architecture. Moreover, detection rates for DoS and Brute force attacks were also influenced by these limitations. DoS detection rates were especially affected by this since they had 40% packet drop rates due to being unable to handle all the traffic. While it is essential to aim for accurate and consistent detection rates, reaching 100% detection is normally unlikely. Another main factor was the alert rule's complexity, which we believe affected the performance of HTTP and Brute force attempts. In the end, in a lightweight IoT environment, where limited resources and constrained processing capabilities may exist, the ability to detect that a particular kind of attack happened, even without precise specifics or accurate attack counts, might still be important. It could help in identifying attack patterns and help in making security and incident response strategies. In addition, one of the vulnerabilities we found, a DoS overwhelm, was the result of this limitation as well. Usually, this kind of vulnerability is solved by utilizing multi-threaded IDS, however, for IoT devices employing rate limiting and traffic shaping techniques could also control the influx of incoming traffic, and prevent DoS from overwhelming the IDS. As regards to other vulnerabilities found, the DoS with crafted packets and nuclei scan, the most we can do is regularly update Snort and the rule sets and utilize new weekly community rules to the IDS. Snort's current inability to recognize detection patterns for nuclei scans and crafted packets can be due to several kinds of factors. Firstly, nuclei being a relatively new technology requires security specialists more time to study their patterns and develop precise detection rules. Additionally, the unpredictable nature of specially crafted packets presents challenges in anticipating all the potential variations and devising effective rules for them. Thus, as the threat landscape shifts, IDSs like Snort are constantly updated to handle new attack methods. However, developing and integrating complete detection rules might require quite some time.

Throughout the literature, there have been many papers evaluating IDS performance and resource utilization on various platforms. In Table 10 you can find the overview of attacks and IDSs by other researchers.

Kyaw et al.[5] have tested their RPi 2 running Snort and Bro(now Zeek) IDSs against SYN flood, Spoofing, and port scanning. In our research, we also examined these attacks. However, we obtained slightly different findings. For SYN flooding of 437369 and 357650 packets with 30% background traffic(BT), Kyaw et al. found that Snort experienced an 11.36%, and Bro experienced 40.76% packet drop rates. In comparison, our system experienced packet drops of 39.48% for Snort during 8973979 attack packets. They also analyzed the packet processing speed during ARP spoofing with 60% BT and observed Snort and Bro achieving 400 and 394 Pkts/sec, respectively, while our system processed 68 Pkts/sec when Snort processed 23358 packets during spoofing. Next, during port Scanning attacks with 90% BT, they reported Snort had 9.6% and 23.5% and Bro had 56.3% and 4.6% CPU and MEM utilization respectively. In contrast, during Nmap scans, we experienced initial CPU usage spikes up to 40% and MEM rising up to only 0.8%.

Furthermore, Sforzin et al.[8] evaluated their RPi 2 Snort against various types of malicious recorded Pcap traffic with different speeds. We believe their traffic was mainly DDoS/DoS. They have found that, when the data rates surpassed 60 Mbit/s, their system experienced CPU usage ranging from 20-100% with packet losses ranging from 40-80%. In comparison, during ICMP and SYN floods, our system encountered CPU usage ranging from 10-95% with packet drop rates of up to 45%.

Moreover, Visoottiviseth et al.[11] developed and evaluated the "PITI" system, which utilized an RPi 3 and Snort IDS for the detection of DoS, password, SQLi, and Evil Twin attacks. While our research did not directly target SQLi and Evil Twin attacks, we carried out more in-depth testing focused on DoS and password attacks. Visoottiviseth et al. reported that their system generated few alerts for DoS and password attacks and had a low amount of false negatives. Our system, on the other hand, produced a higher amount of false positives and issued significantly more alerts during ICMP, SYN DoS, and brute force attacks. Their paper also shows CPU usages of 14-53%, while our system had 10-95% CPU usages during DoS and less than 1% CPU usages during Brute Force.

Additionally, Coşar et al. [1] assessed the performance of Snort and Suricata IDSs on RPi 3 against SYN flood, UDP flood, and Smurf

Table 10. Comparison with other IDS Measurement Papers

References	Detection Method	Pi Model	IDS	Attacks	Environment
Kyaw et al. [5]	Misuse-based	Pi 2-B	Snort, Bro	SYN flood, ARP spoofing, port scanning	Conventional
Sforzin et al. [8]	Misuse-based	Pi 2-B	Snort	-	IoT
Visoottiviseth et al. [11]	Misuse-based, Anomaly-based, Hybrid	Pi 3-B	Snort	SQLi, Password, DoS/DDoS, Evil Twin	IoT
Coşar et al. [1]	Misuse-based	Pi 3-B	Snort, Suricata	SYN flood, Smurf, UDP flood	Conventional
Our Proposed Solution	Misuse-based	Pi 4-B	Snort	ICMP, SYN, HTTP Floods, Password, ARP spoof, port and service scans	IoT

attacks. Among these, we focused solely on the SYN flood. During which, Coşar et al. observed packet drop rates of 6.9% for Snort and 34.9% for Suricata, while in our system we encountered drop rates of around 40%.

In summary, our findings vary from the aforementioned papers, showing differences in various metrics. In the end, based on the literature review and to our knowledge, our proposed solution gained an advantage over the solutions presented in the literature, since we did a more in-depth analysis of many of the attacks experienced by the IDS systems.

5 CONCLUSION

In this research paper, we proposed a Snort IDS system based on RPi as a lightweight security solution for IoT devices. We conducted a series of attack scenarios, made configuration modifications to evaluate the IDS performance, and assessed the vulnerability of the IDS itself to test its effectiveness in IoT environments. Our findings demonstrated that the RPi-based Snort IDS successfully detected various types of attacks, including ICMP, SYN, HTTP floods, Brute Force attempts, Nmap, and Spoofing. The IDS showed prompt responses to most of the attacks we tested, although there was a slight delay in responding to Nmap, likely due to the nature of Nmap's probing mechanism. Regarding false positives, we observed that when the IDS protected a specific IoT device, it did not generate alarming false positives. Instead, it primarily provided alerts related to the state of network traffic. The detection rates did not meet our initial expectations when encountering high volumes of DoS traffic, which led to elevated CPU usage and packet drops in the IDS. The detection rates also depended on the complexity of the alert rules. Despite these findings, the IDS demonstrated a reasonable level of accuracy in recognizing attacks, allowing for estimates of their potential severity. Additionally, the IDS efficiently utilized system resources, consuming less than one-fifth of the available RAM on the RPi, making it a suitable choice for low-powered IoT devices with 1 GB RAM.

Moreover, based on our observations, it was evident that modification of configuration, as well as adjustments to high-priority levels, had minimal effect on Snort's packet processing and overall performance. Instead, we identified that the key factors influencing the performance of the IDS are its internal architecture, the complexity of rule sets, the hardware environment, and the network traffic itself. These findings emphasize the significance of taking these key parameters into account when enhancing Snort's performance in real-world deployment environments.

Furthermore, during our research, we identified three vulnerabilities in the IDS, a DoS overwhelming, undetected nuclei scan and undetected crafted packet attacks. While we identified a remedy for the first vulnerability, the remaining two are currently without a definitive resolution. Nonetheless, we have provided potential recommendations for addressing them in Section 4

For future work, we plan to test the performance of multi-threaded IDS solutions like Snort 3 and Suricata on the RPi. Additionally, we intend to evaluate the performance of multiple running instances of Snort and explore more complex IDS rule signatures.

REFERENCES

[1] Mustafa COŞAR and Harun Emre KIRAN. 2018. Performance Comparison of Open Source IDSs via Raspberry Pi. In *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*. 1–5.

[2] Mohammed El-Hajj, Ahmad Fadlallah, Maroun Chamoun, and Ahmed Serhrouchni. 2019. A survey of internet of things (IoT) authentication schemes. *Sensors* 19, 5 (2019), 1141.

[3] Mohammed El-Hajj, Hussien Mousawi, and Ahmad Fadlallah. 2023. Analysis of Lightweight Cryptographic Algorithms on IoT Hardware Platform. *Future Internet* 15, 2 (2023), 54.

[4] Mario Frustaci, Pasquale Pace, Gianluca Aloi, and Giancarlo Fortino. 2018. Evaluating Critical Security Issues of the IoT World: Present and Future Challenges. *IEEE Internet of Things Journal* 5, 4 (2018), 2483–2495.

[5] Ar Kar Kyaw, Yuzhu Chen, and Justin Joseph. 2015. Pi-IDS: evaluation of open-source intrusion detection systems on Raspberry Pi 2. In *2015 Second International Conference on Information Security and Cyber Forensics (InfoSec)*. 165–170.

[6] Mardiana binti Mohamad Noor and Wan Haslina Hassan. 2019. Current research on Internet of Things (IoT) security: A survey. *Computer Networks* 148 (Jan. 2019), 283–294. https://www.sciencedirect.com/science/article/pii/S1389128618307035

[7] Juan Ruiz Lagunas, Anastacio Hernández, Mauricio R. Reyes-Gutiérrez, Ferreira-Medina Heberto, Torres-Millarez Cristhian, and Paniagua-Villagómez Omar. 2019. How to Improve the IoT Security Implementing IDS/IPS Tool using Raspberry Pi 3B+. *International Journal of Advanced Computer Science and Applications* 10 (Jan. 2019).

[8] Alessandro Sforzin, Félix Gómez Mármol, Mauro Conti, and Jens-Matthias Bohli. 2016. RPiDS: Raspberry Pi IDS — A Fruitful Intrusion Detection System for IoT. In *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. 440–448.

[9] Snort.org. 2023. *Why Snort 3?* https://www.snort.org/snort3

[10] Abid Sultan, Muhammad Azhar Mushtaq, and Muhammad Abubakar. 2019. IOT Security Issues Via Blockchain: A Review Paper. In *Proceedings of the 2019 International Conference on Blockchain Technology (ICBCT 2019)*. Association for Computing Machinery, New York, NY, USA, 60–65. https://dl.acm.org/doi/10.1145/3320154.3320163

[11] Vasaka Visoottiviseth, Gannasut Chutaporn, Sorakrit Kungvanruttana, and Jirapas Paisarnduangjan. 2020. PITI: Protecting Internet of Things via Intrusion Detection System on Raspberry Pi. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*. 75–80. ISSN: 2162-1233.

[12] Abubakar Zakari, Maryam Musa, Girish Bekaroo, Surayya Ado Bala, Ibrahim Abaker Targio Hashem, and Saqib Hakak. 2019. IPv4 and IPv6 Protocols: A Comparative Performance Study. In *2019 IEEE 10th Control and System Graduate Research Colloquium (ICSGRC)*. 1–4.