Exploring Open-Source Intrusion Detection Systems for Smart Home Security

Matthew Richard
School of Computing
Queen's University
Kingston, Ontario
23brlv@queensu.ca

Nicholas Dionne
School of Computing
Queen's University
Kingston, Ontario
24kg1@queensu.ca

Yannick Abouem
School of Computing
Queen's University
Kingston, Ontario
24hd7@queensu.ca

***Abstract*—Smart homes, homes that implement interconnected IoT devices, greatly enhance convenience and security in a home, resulting in them becoming more popular throughout the last decade. However, the increased use of smart homes also introduces new security vulnerabilities. This project evaluates the effectiveness of Suricata, an open-source intrusion detection system (IDS), as a lightweight and accessible security solution for smart home environments. We use Home Assistant to simulate a realistic smart home setting and develop custom Suricata rules tailored to detect common IoT attacks, including code injection, brute force login attempts, and denial-of-service (DoS) attacks. Our results demonstrate that Suricata operates with minimal resource consumption while successfully identifying malicious activity. By prioritizing usability, this project showcases the potential of open-source IDS tools to provide effective security for smart homes without requiring expert knowledge. The findings contribute to the advancement of user-friendly smart home security solutions, emphasizing practicality and accessibility for homeowners.**

***Index Terms*—cyber-physical system, smart home security, intrusion detection system, Home Assistant, Suricata.**

1. Introduction

Throughout the years, cyber-physical systems (CPS) have been slowly integrating more and more into our daily lives [1]. This can be seen with the rise in popularity of autonomous vehicles from companies like Tesla, smart grids used by IBM, and medical devices like pacemakers [1]. CPS are systems that leverage the functionality of both physical and computational components [1]. One of the most prominent CPS seen in daily life are smart homes [2].

Smart homes are traditional homes that contain numerous devices connected through the Internet of Things (IoT) [2]. The IoT devices in the home allow for the automation and control of common household aspects like lighting, temperature, and security [2]. For example, smart lights that turn on when motion is detected or controlled remotely through a phone, and smart security systems that can remotely lock doors and monitor for suspicious activity around a home. The innovation of smart homes has greatly enhanced convenience, safety, and the overall quality of life for homeowners [2].

The increasing use of smart homes and their interconnected IoT devices, while overall beneficial, has also introduced many new security vulnerabilities [3]. The many vulnerabilities unique to these devices highlight the pressing need for smart home security solutions.

This project explores the viability of Suricata, an open-source intrusion detection system (IDS), as a lightweight and practical security solution for smart homes. Rather than focusing on complex or pre-existing rule sets, we developed our own easy-to-understand and highly customizable rules tailored specifically to common IoT threats. These rules were tested within a simulated smart home environment using Home Assistant, an open-source platform for home automation, prioritizing ease of use and deployability for all users.

By designing accessible and effective IDS rules, this project aims to contribute to smart home security research with a focus on usability. Our goal is to demonstrate that smart home security does not need to be limited to experts; customizable IDS solutions like Suricata can offer security that is both approachable and adaptable for everyday users. Furthermore, by publishing our ruleset and methodology, we aim to show that smart home security can be both accessible and understandable to homeowners.

The following sections provide context and detail for our work. We begin by providing the motivation behind our work, and discussing related work. Next, we provide important background information relevant to they key topics in our report; smart home security, intrusion detection systems, and Suricata. We then outline our implementation methodology, including the simulation setup and custom rule development. This is followed by an evaluation of our results, a discussion of potential future work, and concluding thoughts.

2. Motivation

Smart homes, and their increasing widespread use, have significantly benefited the lives of homeowners, offering convenience, efficiency, and enhanced security. As a result, the homeowner experience has improved in all regards.

Despite the overwhelming benefit of smart homes and IoT devices, the use of the advanced technology also introduces many new security risks [3]. More specifically, the use of multiple devices communicating over a wireless network makes a smart home vulnerable to cyberattacks [3]. A compromised smart home system has the possibility to result in privacy breaches, data leaks, and even the manipulation of the physical devices. Therefore, maintaining the security of smart homes is of the upmost importance for all homeowners.

The most used and well tested cyber-physical system security solutions are intrusion detection systems [4]. IDS monitor the activity of IoT devices for possible malicious attacks [4]. While IDS tools like Snort and Suricata have been widely researched and applied in traditional network environments, their effectiveness in smart homes has not been greatly explored. With the unique vulnerabilities smart homes have, the lack of IDS research in the domain is alarming.

This project addresses that gap by evaluating the viability of Suricata as a smart home security solution. Rather than relying on pre-existing configurations, we develop customized detection rules designed specifically for common IoT threats and test them in a realistic, simulated smart home setup. Our goal is to contribute to the advancement of smart home security research by promoting lightweight, accessible, and user-friendly IDS strategies designed for this unique environment.

3. Related Work

As smart homes have increased in use, research in smart home security has also increased. While the interest in this research field is still new, there has been papers that have goals similar to our own in this paper. Such prior research has explored the performance of open-source IDS tools in general IoT networks, compared IDS tools in a smart home setting, and even propose new security methods [5], [6], [7]. In this section, we outline they key findings from these studies, and compare them to our own work.

In their 2023 paper ”A Comparative Analysis of Snort 3 and Suricata”, Boukebous et al. conduct a comparative analysis of Snort 2, Snort 3, and Suricata and their performance as network intrusion detection systems [5]. The three IDS systems were tested in a virtualized environment in which they measured memory and processor usage, packet processing rates, and packet drop rates [5]. The results indicated that Snort 3 greatly improved the performance of Snort in comparison to Snort 2 [5]. More specifically, Snort 3 improved upon processor and memory usage as well as decreased packet drop rate [5]. Comparing Snort 3 to Suricata, Suricata on average has greater processor and memory usage, however, it has a significantly lower drop rate for packets and alerts [5]. The authors conclude that while Suricata is the best performing NIDS, future advancements to Snort 3 will increase its competition [5]. Unlike their work that uses default rulesets in a generic IoT environment, our work specifically adapts Suricata for smart home environments by developing lightweight, and customizable rules targeting threats specific to smart homes. Furthermore, their evaluation prioritizes throughput metrics, whereas we emphasize practical usability for homeowners. Ultimately, our work is highly focused in the smart home domain, and not a general overview of IDS performance.

Alsakran et al. perform a comparative analysis of different IDS systems in a smart home setting in their 2020 paper ”Intrusion Detection Systems for Smart Home IoT Devices: Experimental Comparison Study” [6]. Their work compared three open-source IDSs, Snort, Suricata, and Bro IDS [6]. The authors set up a testing environment using Docker containers [6]. Using a TCPreplay, the authors replayed PCAP files containing malicious traffic from different attacks [6]. Their results show that both Suricata and Bro IDS use a similar amount of processor and memory resources when identifying the different attacks [6]. On the other hand, they found that Snort uses significantly more resources than both Suricata and Bro IDS [6]. They conclude their work with noting that both Suricata and Bro IDS are effective open-source intrusion detection systems for a smart home [6]. This paper focused on comparing multiple IDS. Comparatively, our work focuses solely on Suricata, as we noted in our foundational research that is known to perform best. Additionally, while Alsakran et al. provided a strong comparison of IDS performance through the use of malware PCAPs, their study did not evaluate threats specific to smart home IoT devices. In contrast, our work tests attacks directly against the simulated IoT devices and their MQTT communication in Home Assistant, properly testing the effectiveness of Suricata in a realistic smart home domain.

Sookavatana proposes a lightweight security handshake protocol in their 2022 paper ”Detecting security violations on IoT devices in home automation systems using a lightweight security handshake protocol” [7]. The proposed protocol aims to detect security violations affecting devices connected to a smart home system gateway [7]. The handshake monitors events and requests authorization confirmation from any device attempting to communication [7]. The protocol was implemented into Home Assistant, offering a different protection approach [7]. The protocol was tested and compared to Home Assistant with no IDS, Home Assistant running Snort, and Home Assistant running Suricata [7]. The results show that in comparison to Home Assistant with Snort or Suricata, their lightweight protocol significantly reduces processor and memory usage [7]. In the paper, that author proposes a new smart home security method that differs from traditional IDS options. As a result of the protocols design, threats are only detected after the security violation happens. In contrast, our work uses and IDS, which can detect attacks before a security violation happens. Overall, our work relies on tested and trusted smart home security solutions, whereas this paper focuses on introducing new solutions.

While prior research has explored IDS performance in general, compared IDS solutions in a smart home environment, and proposed new security solutions, our work takes a more focused and practical approach. Through the creation of Suricata rules that are specific to smart home threats and validation of them in a simulation using Home Assistant, we fill in the gap of research about highly accessible, customizable, and usable smart home security solutions. Our focus on the user experience for smart home security makes this work a valuable contribution to the ongoing research in the smart home security field.

4. Background

In this section, we will go over the background information relevant to the key concepts in our project. It introduces intrusion detection systems, the specific tool we are implementing, Suricata, smart home simulation, and IoT attacks.

4.1. Intrusion Detection Systems

An intrusion detection system (IDS) is a software that monitors a network or a specific host to detect potentially malicious traffic [20]. To detect malicious traffic, an IDS captures and analyzes each packet traveling over the network and performs some sort of analysis. In the case the packet were to be deemed malicious by the IDS an alert would be produced to notify the network administrators of a potential intrusion occurring. Some IDS posses more active countermeasures and can block packets and connections that are considered malicious [20].

IDS are a crucial aspect of cybersecurity, and although IoT may be limited in performance, IDS is a solid security measure for IoT-based smart home devices. IDS are commonly classified in the following categories:

- Network Intrusion Detection System, which provide security for a whole network composed of multiple hosts
- Host Intrusion Detection System, which are installed onto a specific host system and detects threats coming from the network towards the host or from within the host [20]
- Stack-based Intrusion Detection System, which operate at the transport layer and can detect packets before it can be received by an application [20]

Modern IDS are usually a hybrid combinations of the previously mentioned categories to better determine if a specific action is malicious.

IDS can also be classified into three detection types: signature-based, anomaly-based and hybrid. A signature-based IDS identifies malicious activities by using predefined attack signatures or a database of known threats, making it ineffective against zero-day attacks [20]. Despite this limitation signature-based IDS are effective against known attacks and have an overall low false positive alarm rates [21]. An anomaly-based IDS, on the other hand, utilize machine learning or a statistical model based on the regular network traffic of the implementation environment to determine if activities are anomalous and therefore malicious or outside of the norm [20]. Hybrid IDS combine signature-based and anomaly-based IDS in an attempt to gain the best of both implementations and improve the overall accuracy of the IDS.

Following is an overview of some commonly used IDS, including Suricata, the IDS we chose to work with for this project.

4.1.1. Snort. Snort is a popular open-source anomaly-based Network IDS that is widely used in the industry. Snort is composed of three modes that offer different functionalities:

- *Sniffer mode*, captures and reads packets on the network for display [8]
- *Paket Logger mode*, which operates similarly to sniffer mode but stores packet data onto the disk [8]
- *Network Intrusion Detection System mode*, the mode of interest to us, this mode performs detection and analysis of the network [8]

In Network Intrusion Detection System mode, snort applies a set of user-defined rules to each packet on the network to determine whether action should be taken [8]. Actions typically include producing an alert and logging or showing a message to administrators containing a message and the packet header that triggered the alert [8].

Snort is a widely popular and actively developed IDS. It was originally published in 1998, and has since been in active development [9]. Its popularity and wide usage have made it a staple in the network security industry. Moreover, later versions of Snort support multi-threading for faster detections [9].

4.1.2. Suricata. Suricata, similar to Snort, has multiple functionalities, including network IDS, Intrusion Protection System, and Network Security Monitoring Engine [10]. Surcata, detects malicious packets by matching signatures, often called rules, on packages to detect malicious activities [10]. This program supports multi-threading and is considered a newer alternative to Snort [9]. Past research also suggest Suricata to be better performing than other IDS [9]. For these reasons we decided to use Suircata in this project, as well as aligning to our goal of user-friendliness and ease of use.

4.1.3. Other Intrusion Detection Systems. During our research, we also found two more open-source IDS. These are OSSEC, developed by the OSSEC project team, and Zeek, developed by the Zeek Project. However, we decided to not use these IDS in our project for the following reasons. OSSEC, unlike the other systems we considered, is a host-based intrusion detection system, which means that it only detects intrusion in a specific system rather than the entire network [11]. Although this could help in detecting attacks on the server that manages the Smart Home system, it is not in line with our goal.

As for Zeek, we decided not to use this program due to its smaller rule set that will restrict its efficacy [9]. Another reason is that Zeek first functionality is not to be an IDS but rather a Network Security Monitoring tool and has different functionality from Snort and Suricata making a comparison unfair. Zeek does offer a IDS functionality but it uses multiple criteria to determine if a packet is malicious or not, which include signature-based and anomaly-based approaches [12]. This program is a powerful tool for network security but it does not fit in the scope of this project.

Other IDS solutions include BluVector Cortex, a proprietary anomaly-based IDS used in many commercial settings, Cynet, another proprietary anomaly-based IDS, and Fail2Ban, which only protects against brute force login attacks and unauthorized logins.

4.2. Cyber-physical System Simulation

The simulation of a realistic smart home environment is a major component of our project, allowing us to test and evaluate the performance of the IDS. It is vital that our simulated CPS accurately reflects real-world smart home interactions, as this ensures the validity of our research and findings.

4.2.1. Home Assistant. To simulate a realistic smart home environment, we utilize Home Assistant for our project. Home Assistant is an open-source smart home automation software designed to manage all the IoT devices within a smart home [13]. The platform offers many important features that are essential to monitoring and automating a smart home [13]. Such features include customizable automation, connectivity support between IoT devices made from different manufacturers, and real-time monitoring [13]. Additionally, Home Assistant allows for the simulation of smart home devices, creating the opportunity for users to test and develop systems without requiring any physical hardware [13]. To accompany the software and its many features, Home Assistant provides a user-friendly interface in the layout of a dashboard that displays all the important information about a smart homes IoT devices.

Home Assistant, with its simulation functionality, will be the foundation of our project. We will simulate common IoT devices such as locks, temperature controls, and lighting systems, and have them communicate to Home Assistant through a communication protocol such as the Message Queuing Telemetry Transport (MQTT) or the Hypertext Transfer Protocol (HTTP) [14]. The simulated devices will be configured to reflect a real-world smart home IoT device, like allowing for the remote control of lights and temperature. As previously mentioned, these devices will be subjected to numerous simulated cyberattacks, all while the connected an IDS like Suricata aims to detect the threats. Home Assistant's monitoring features will allow us to oversee the entire process and measure the behaviour and effectiveness of the IDS.

4.2.2. Other Smart Home Networks. While Home Assistant is our chosen software for simulating a smart home environment, there are many other tools that have similar features.

The four most popular non-proprietary smart home platforms are Home Assistant, openHAB, Domoticz, and ioBroker [15]. Unfortunately, the three latter platforms mentioned do not meet all of our requirements when considering this project. openHAB is an open-source home automation software that is similar to Home Assistant in many ways [15]. It offers excellent automation and rule-based configurations, however, it's setup is more complex and it's user interface is less friendly than Home Assistant's [15]. The Domoticz home automation platform excels at automation and prioritizes this functionality [15]. As a result, compared to Home Assistant or openHAB, Domoticz does not have many features that are required for our project, such as IoT device simulation [15]. Similar to Home Assistant and openHAB, ioBroker is an open-source platform that supports hundreds of IoT devices [15]. The platform not only specializes in large-scale automation, which is not necessary for our project, but also does not allow for the simulation we need [15].

Beyond open-source solutions, there are also many popular proprietary smart home ecosystems available on the market. Proprietary platforms are developed and controlled by specific companies, meaning their source code is not publicly available and their customization options are often limited [16]. Some such examples of these proprietary smart home platforms include Apple HomeKit, Amazon Alexa, and Hubitat [17], [18], [19]. However, these platforms were unsuitable for our research needs. Apple HomeKit prioritizes privacy and security, but its closed ecosystem limits customization and traffic monitoring for our security testing [17]. Amazon Alexa offers voice-controlled automation using Amazon Echo devices which does not meet our goals of the project with the requirement of owning an Echo [18]. Lastly, Hubitat requires a Hubitat Hub to be purchased for their smart home automation [19]. Additionally, Hubitat does not offer the simulation capabilities that we require for our work [19]. For our goal of reproducible IDS testing, open-source platforms are ultimately more effective and accessible in comparison to proprietary platforms.

After comparing each of these smart home platforms, it is clear that Home Assistant aligns with our requirements the most. The primary goal of this project is to create a realistic smart home environment. Using Home Assistant's monitoring and simulation features, we can not only replicate IoT devices, but also assess the effectiveness of an IDS in securing a smart home network.

4.3. IoT Attacks

IoT devices provide enhanced convenience and automation, but often at the cost of security due to limited computational power and poor standardization in the area [22]. Even with governing bodies such as NIST on the case, adding pressure to IoT manufactures to increase security measures, IoT has remained an appealing target for attackers due to the aforementioned issues [22]. Some common attacks against IoT devices are brute-forcing credentials, code injections, denial-of-service, and man-in-the-middle attacks [22]. Due to the diverse applicability of IoT systems to a multitude of environments with many different communication protocols, it makes securing IoT systems that much more challenging, not to mention that many of these devices typically lack more basic yet effective security measures such as rate limiting, encryption, and in some cases even input validation [22]. All this together can make IoT systems, in some cases, easily susceptible to even rather simple attacks, such as the simple yet effective ones utilized in our project.

5. Methodology

In this section we will present the methodology for our experiment. We will discuss the experimental setup, including Home Assistant set up and the Suricata installation. We will then discuss the rules we have written as part of our experiment and the attacks we performed. We will conclude this section by presenting each steps taken during the experiment as well as the metrics we recorded.

We conducted our experiments on a Lenovo ThinkPad X1 Carbon with an Intel i5-8365U CPU with 8 physical and 16 logical cores and 4.1 GHz, and 16 GB of RAM, running NixOs 25.05 pre-release. Home Assistant was installed in a virtual machine running Home Assistant OS (HAOS), while Suricata was installed on Ubuntu server 24.04. Both virtual machines where run with QEMU. To each virtual machine we assigned 4 cores and 8 GB of RAM.

5.1. Home Assistant Setup

Home Assistant was chosen as our platform for simulating the smart home setting due to its strong documentation, open-source nature, and extensive support for simulation.

Home Assistant can be implemented into many different operating systems such as Windows, Mac, Linux, and even Raspberry Pi OS. We ran Home Assistant on a virtual machine by downloading Home Assistant's own operating system, Home Assistant OS, and running the OS in our VM. Through this virtual machine, we created a self-contained testing environment for our project.

All Home Assistant applications come with a configuration file, `config.yaml`, where users can modify settings, simulate devices, and even automate their devices. This file is where we simulated all the IoT devices in our smart home environment. These automated devices include a light, fan, security camera, thermostat, and smoke detector. Through this configuration file, we were able to configure each device to have settings unique to their real life counterpart. For example, the thermostat has heating, cooling, and off modes, while also being able to set to maximum temperature, minimum temperature, and rate of temperature change. In contrast, the smoke detector simply has options for being on or off. Below is a code snippet from our `config.yaml` file showing the configuration of the aforementioned smoke detector:

```
binary_sensor:
- name: "Kitchen Smoke Detector"
state_topic: "home/kitchen/smoke/state"
device_class: smoke
payload_on: "ON"
payload_off: "OFF"
qos: 0
```

To enable communication between simulated devices in the smart home network, we used the Mosquitto MQTT broker add-on, which was installed directly within Home Assistant. This broker controlled all publish and subscribe messages between the simulation devices using the MQTT protocol. Through this add-on, we were able to replicate IoT device communication seen in real smart homes.

Once all the devices were configured and communication was enabled, Home Assistant provided a visual interface for the environment. Each device is represented by a graphic, allowing for constant monitoring of the device. Users can even interact with the smart home devices through controls on the dashboard. The dashboard for our simulated smart home environment is shown in Figure 1.

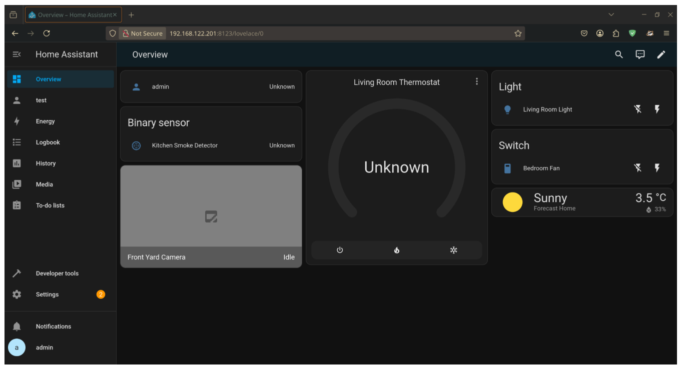

Figure 1. Screenshot of our simulated smart home environment dashboard in Home Assistant.

5.2. Suricata Installation

As a free and open-source software, Suricata is free of charge to use and quite easy to install, especially on GNU/Linux systems such as Ubuntu. Suricata is distributed by the developer's own repository, thus, we first import the necessary repository. Then, installing Suricata is merely a matter of installing the `suricata` package. After installation, to run Suricata it is only necessary to enable the `suricata` service via `systemd`. It should by default be already enabled, however there are changes that are needed in the Suricata's configuration file in order for the IDS to work properly. Suricata, by default, includes rules written by the community that can offer a basic lavel of security. However, for our purposes, we decided to not use these rules and write our own.

5.3. Attacks

To assess Suricata's capabilities in detecting malicious activity, we implemented three attacks we considered representative of common attacks targeting IoT systems similar to our Home Assistant environment.

- Code Injection Attack
  - Target: Home Assistant automation scripts.
  - Method: We crafted a malicious MQTT payload containing a semicolon as a proof of concept for piggybacking code or commands in MQTT packets past Suricata. Such payloads could alter device states or trigger unintended behaviors, allowing for further exploitation.
  - Outcome: The attack was successful in manipulating the device using atypical inputs. The detection of the attack utilizing Suricata required us to create custom rules to inspect MQTT message contents for those malicious payload patterns. It is important to note that many more rules than we implemented would be required to detect all possible malicious MQTT packet message contents.
- Brute Force Login Attack
  - Target: MQTT Authentication.
  - Method: A bash script was written to repeatedly attempt MQTT logins with various username and password combinations from a file. Files such as the top one million IoT passwords can be utilized to find commonly used and often simple IoT passwords with little to no effort, with similar files for usernames as well, although these are typically set as a manufacturer name or as admin.
  - Outcome: This attack showed that weak and default IoT credentials are rather easily compromised. Suricata was in this case able to easily detect the brute-force attempt once the packets surpassed a certain threshold within a short time frame; however, if brute force attempts are kept beneath this threshold, this attack may go unnoticed for a period of time.
- Denial-of-service (DoS) Attack
  - Target: MQTT Broker.
  - Method: By publishing a large number of MQTT packets within a small time frame, we simulated a flooding attack.
  - Outcome: The performance of the broker and Suricata was noticeably impacted, simulating the exhaustion of resources under a DoS attack; however, in our case, this impact wasn't significant as we had much more in the way of resources than actual IoT devices in our simulated environment, and our flooding method was rather simple in design. Suricata was again able to detect this attack rather easily as it's similar to a brute force in design; however, a DoS is unlikely to hide beneath the threshold while impacting resources, so this should not be an issue as before.

Although our attacks were simple in design, they were effective in some cases and a valid method of assessing Suricata's capabilities. These attacks are also relatively common among IoT devices, thus ensuring we remain grounded in a realistic study of open-source intrusion detection systems for smart home security.

5.4. Rules

To detect the attacks we have shown in the previous section, we wrote three rules. Each of these rules were written to detect MQTT packets that match a certain signature. Each rule will, in the case of a positive match, produce an alert that contains information on the packet that matched with this rule and a user created message. The following is a in depth look at the meaning and functioning of each rule.

The first rule we are going to explain is the payload injection rule. This rule is as follows:

```
alert mqtt any any -> $HOME_NET 1883 (
    msg:"Suspicious Payload Detected";
    mqtt.publish.topic;
    content:"home/bedroom/fan/set";
    mqtt.publish.message;
    content:"|3b|";
    nocase;
    sid: 1000123;
    rev:1;
)
```

As the name implies, this rule attempts to detect payload injection attacks by looking in the contents of an MQTT packet and checking if a specific character is present. This character, in our case is a semicolon (';'), which is commonly used to separate commands in bash. What this rule does in practice is, it alerts the administrators if it detects an MQTT packet that was sent by any host on any port and has as its destination the Home Assistant server (our 'home network') on the port 1883, which is used by MQTT. In addition this packet must contain all the following characteristics:

- The packet is publishing a message to a particular topic
- The packet contains in the message it tries to push the hex value for the semicolon (`3B`)

If a packet meets all the requirements then an alert is created and logged into Suricata's log file.

The second rule is designed to detect possible DoS attacks to the Home Assistant through the MQTT protocol. To do so this rule will detect if too many connection requests are made. The number we settled upon is of 100 packets detected in one second, which is a fair metric to simulate a DoS attack. The rule, also considers the source of the packet when counting the number of packets in order to avoid detecting different MQTT connections as being an attack. Following is the second rule used in the project:

```
alert mqtt any any -> any any (
    msg:"MQTT Multiple connection
    requests";
    mqtt.type:CONNECT;
    threshold:type both, track by_src,
    count 5, seconds 1;
    sid: 0001;
    rev:1;
)
```

The third and last rule we designed is the brute force login rule, which detects multiple failed attempts at creating a connection due to incorrect credentials. Below is the rule we have written:

```
alert mqtt $HOME_NET any -> any any (
    msg:"MQTT multiple login failures.
    Possible brute force attack!";
    mqtt.type:CONNACK;
    mqtt.reason_code:5;
    threshold: type both, track
    by_dst, count 5, seconds 1;
    sid: 999999;
    rev:1;
)
```

Unlike the other rules presented above, this rule detects packets that are coming from the Home Assistant server towards the attacker, because the way we can detect such attack is by counting the number of responses sent by the server to the client. In particular, these packets must contain a MQTT reason code of 5, which corresponds to a failed login attempt. This rule will display an alert if 5 such packets are sent on the network towards a specific host in one second.

5.5. The Experiment

To measure the performance of Suricata, we decided to run an experiment and collect data on the overall impact of the IDS on the system. The metrics we have chosen are the following:

- **Average CPU usage** the percentage of CPU used by the system while the IDS is running
- **Average Memory Usage** the amount of memory used by the system while the IDS is running
- **Average Network Usage** the amount of network traffic recorded while the IDS is running

We decided to chose these metrics because they offer a good representation of the burden the IDS has on the system. To offer insights on the impact, we compare our results with the same metrics recorded for the system while idling.

We carried out our experiment following this procedure and recording all our data using `collectl`, a open-source utility to collect usage data of a GNU/Linux system:

1) Record the data used for the baseline by letting the system run in idle with the `suricata` service stopped for 30 seconds
2) Enable the `suricata` service through `systemd`
3) Record performance on the system while Suricata is idle, meaning no attack is being carried out
4) For each attack we record the performance while the attack is running

One issue we ran into while collecting data was that the program we used to collect data returned unreliable data for the network usage. Instead of returning correct values for the network usage, `collectl` produced an output consisting of only zeroes. These results cannot be possible as the virtual machine we used to run Suricata was connected to a network and was functioning properly. Thus, we decided to discard these results and the metric entirely.

6. Results

State	Avg. CPU Usage	Avg. Mem. Usage
Idle	$< 1\%$	716280
Idle with Suricata	$< 1\%$	1192060
Code Injection Attack	$< 1\%$	1194447
DoS Attack	15%	1240472
Brute Force Attack	9%	1208835

TABLE 1. RESULTS OBTAINED IN THE EXPERIMENT

In this section we will provide the results obtained in our experiment as well as an analysis of our findings.

As we can see from Table 1, the CPU usage remains relatively low across all of the five states tested. The only two cases in which the system used more CPU than in the idle state was during the DoS attack, in which it used 15% in average, and the brute force login attack, in which the usage was at 9% on average. The reason for this behavior can be linked to the higher number of packets being captured and processed by Suricata compared to the code injection attack. A similar situation is visible in the memory usage. The system used more memory while the DoS and the brute force attack were ongoing. The memory increase over the ‘Idle with Suricata’ state was of 4.06% for the DoS attack and of 1.4% for the brute force attack, as opposed to a 0.2% for the Code Injection Attack. Perhaps the most interesting result is the fact that Suricata has a minimal impact on the CPU usage of the system and a marginal impact on the used memory, with a 39.9% increase in memory usage over the idle system.

7. Challenges

All research projects face challenges. Despite seeming like a roadblock in research and development, challenges actually provide valuable opportunities for further learning and innovation. In our project, several challenges arose from the complex nature of smart home devices and the implementation of an IDS. Overcoming these challenges and adapting our research has been crucial to the success of this project.

One major challenge that we faced comes from the IDS evaluation process, as this required us to perform an adequate amount of real-world attacks on Home Assistant to some success, as well as adopt the use of tricky performance metric tools such as collectl. The difficulty with regard to performing attacks can be found in how Home Assistant is a smart home environment, and so it consolidates a unique set of IoT devices and can, in some cases, have complex and unique network activity. So, when attacking this environment, we had to carefully consider how to design our attacks to be effective even given their simplicity (DoS, brute-force, code injections, etc). To potentially mitigate this obstacle, we considered taking the approach of utilizing IoT-specific attack datasets as either inspiration for our own attacks or as a direct means of evaluating our IDS; however, we did not end up pursuing this avenue other than making use of it for inspiration in regards to potential attacks.

To further expand on the above notion, it can be seen that we have two options as to attack methods, the first being that we construct our own attacks. Constructing our own attacks for the simulated environment would ensure that our attacks are fully adapted to its structure and are effective as an evaluator for Suricata. However, the amount and diversity of our attacks, as well as their sophistication, would likely be inferior to any specialized attack datasets made for IoT. Our Second option would be to introduce attack datasets such as TON-IoT, BoT-IoT, and IoT-23 utilizing these .pcap datasets We could potentially introduce a wide birth of sophisticated attacks in some cases mixed with normal traffic into our simulated environment by using a tool such as tcpreplay which would inject the traffic as if it where normal. There are pros and cons to either of these approaches, and in fact, because of that we decided to form our own attacks regardless of the advantages of utilizing attack datasets in our simulated environment such as the sheer amount of differing attacks especially for when it comes to more large scale or intensive attacks that we would find difficult to effectively employ ourselves such as bot nets in the BoT-IoT dataset. The creation of our own attacks allowed us to expand our exploration of Suricata beyond the surface level and into the formulation of rules to cleanly detect our own attacks against the system. Not to mention that with our setup, it was more practical to apply our own formulated attacks.

Another challenge to our project implementation was Snort; we had originally had the intention of also deeply exploring Snort on top of Suricata. Although we were able to successfully explore Snort and its capabilities to an extent allowing us to make some comparisons, we ultimately decided that a deeper and more thorough analysis of Suricata would be more beneficial to our project and goals rather than splitting our efforts across both IDSs. Some of the challenges we ran into with Snort include the rather poor documentation, both official and unofficial. The official documentation is okay, but it is lacking examples and explanations of the syntax when it comes to Snort3, at least when compared to Suricata. Moreover, in the case of unofficial Snort3 documentation, the IDS community seems to be generally disinterested in the matter, as most of what can be found is for Snort2, which, annoyingly enough, has a very slight difference to Snort3 in both commands and syntax, so previous documentation, both official and unofficial, becomes mostly useless. Snort also gave us other troubles in both performance and detection of attacks. Performance wise, there is an interesting difference between Snort and Suricata, as Suricata once activated runs as a relatively un-intrusive background operation that's barely noticeable unless a large amount of traffic is detected, while Snort runs at the fore front with a consistent strain on resources. However, this may in part be due to how Suricata has multi-threading active by default and Snort does not [9].

Rule writing was another challenging aspect of our project. In both cases, this was mostly because of syntax and the lack of a proper development environment for rules, which may have provided better information when encountering errors. However, another aspect was understanding the unique nature of the packets that have to be detected and how they can be exploited, in our case, these were MQTT packets. The only difficulties we ran into here was a slow learning process with little helpful documentation, although, Suricata thankfully does provide prime examples of there rules on Github and even some rules for MQTT packets which was helpful as an initial guide. Also, transferring rules between Snort and Suricata has nearly no support, both documentation-wise and in terms of converters. So, for any organization or individual looking to make a switch from one IDS to another, or our group in this case, things can get difficult, especially when you are new to the syntax.

8. Future Work

Our project rather successfully helps to demonstrate the feasibility of using Suricata and potentially other IDS for detecting common IoT attacks by utilizing a simulated smart home environment. This, however, does not mean that our project can not be further expanded or improved in some areas.

In fact, expanding upon our attacks would be a good start to further assess the performance capabilities of Suricata; by implementing additional attack types, our evaluation can be more thorough. For example, Man-in-the-Middle (MITM) attacks, which are commonly used in the realm of IoT, could target MQTT traffic to get more insights into how Suricata can detect more complex and stealth attacks.

IDSs such as Suricata often offer other, more advanced detection services, such as anomaly-based detection and Intrusion Prevention Systems (IPS). In future work, we could explore some of these options to test in our environment and further our detection capabilities.

Another potential avenue for future work is integrating and testing Suricata with our attacks on a real smart home network with the actual physical IoT devices. This would allow for more thorough research and a better understanding of how a real-world application of Suricata and IDSs in smart home networks would behave and interact under stress.

Although in our project we dabbled in Snort3 and not just Suricata, it would be best to return to Snort3 for another round of analysis once given the opportunity. Furthermore, there are plenty of other open-source IDS that we could test, some lighter and simpler than Suricata or Snort; examples of other open-source IDS we could test are BluVector Cortex, Cynet, Fail2Ban, OSSEC, Zeek, and OpenWIPS-ng.

9. Conclusion

This paper explored the viability of using Suricata as an accessible and lightweight intrusion detection system for smart home environments. To ensure that all smart home owners can have a security system that is free and can work will all devices, we investigated the accessibility of the intrusion detection system Suricata in a simulated smart home environment.

Using Home Assistant, we simulated a realistic smart home environment with numerous IoT devices that communicate over MQTT. A set of specialized detection rules targeting common smart home vulnerabilities, including payload injection, brute force attacks, and denial-of-service attempts.

The results show that Suricata performs reliably with minimal impact on system resources. Our custom rules successfully identified malicious activity. With this, we showed that effective smart home security can be achieved without requirement proprietary solutions or expert knowledge.

Our work demonstrates that high level smart home security can be achieved through accessible open-source solutions. By creating simple and easy to understand rules tailored towards common IoT attacks, we showcase that any homeowner can easily implement smart home security without expert knowledge. The results we recorded prove that these simple rules are highly effective and that security can be lightweight and practical. These findings offer a strong foundation for the continued development of user-friendly smart home security solutions.

To conclude, in a world were smart homes are becoming increasingly popular, our work demonstrates that security can be both accessible and effective without the need for complexity or compromise.

References

[1] Wankhade, M. & Kottur, S. Security Facets of Cyber Physical System. *2020 Third International Conference On Smart Systems And Inventive Technology (ICSSIT)*. pp. 359-363 (2020)

[2] Ahmad, H., Gupta, M. & Shivam A Study on Implementation of Cyber Physical Systems in Smart Home. *2023 IEEE 11th Region 10 Humanitarian Technology Conference (R10-HTC)*. pp. 58-63 (2023)

[3] Davis, B., Mason, J. & Anwar, M. Vulnerability Studies and Security Postures of IoT Devices: A Smart Home Case Study. *IEEE Internet Of Things Journal*. **7**, 10102-10110 (2020)

[4] Dotty, F. & Asnar, Y. Intrusion Detection System Architecture for Cyber-Physical System. *2023 International Conference On Electrical Engineering And Informatics (ICEEI)*. pp. 1-6 (2023)

[5] Boukebous, A., Fettache, M., Bendiab, G. & Shiaeles, S. A Comparative Analysis of Snort 3 and Suricata. *2023 IEEE IAS Global Conference On Emerging Technologies (GlobConET)*. pp. 1-6 (2023)

[6] Alsakran, F., Bendiab, G., Shiaeles, S. & Kolokotronis, N. Intrusion Detection Systems for Smart Home IoT Devices: Experimental Comparison Study. *Security In Computing And Communications*. pp. 87-98 (2020), http://dx.doi.org/10.1007/978-981-15-4825-37

[7] Sookavatana, P. Detecting security violations on IoT devices in home automation systems using a lightweight security handshake protocol. *2022 6th International Conference On Information Technology (InCIT)*. pp. 23-28 (2022)

[8] Cisco Systems, Inc., San Jose, CA, USA. *SNORT User Manual*, v2.9.16 (2020). Accessed: Mar. 3, 2025. [Online]. Available: https://www.snort.org/documents/1

[9] A. Waleed, A. Fareed, and A. Masood, "Which open-source IDS? Snort, Suricata or Zeek," Computer Networks, vol. 213, pp. 109116, Aug., 2022, doi: https://doi.org/10.1016/j.comnet.2022.109116.

[10] Open Information Security Foundation, Lafayette, IN, USA. *Suricata*, v7.0.8 (2025). Accessed: Mar. 3, 2025. [Online]. Available: https://docs.suricata.io/en/latest/index.html

[11] OSSEC Project Team. "*OSSEC - Open Source HIDS - FIM, Rootkit Detection, Malware Detection*." ossec.net. Accessed: Mar. 4 2025. [Online]. Available: https://www.ossec.net/about/

[12] Zeek Project, San Francisco, CA, USA. *Zeek Documentation*, v7.0.5 (2024). Accessed: Mar. 4, 2025. [Online]. Available: https://docs.zeek.org/en/lts/

[13] Open Home Foundation, Switzerland. *Home Assistant Documentation*, 2025.2.5 (2025). Accessed: Mar. 2, 2025. [Online]. Available: https://www.home-assistant.io/docs/

[14] Cujilema Paguay, J., Hidalgo Brito, G., Hernandez Rojas, D. & Cartuche Calva, J. Secure home automation system based on ESP-NOW mesh network, MQTT and Home Assistant platform. *IEEE Latin America Transactions*. **21**, pp. 829-838 (2023)

[15] Setz, B., Graef, S., Ivanova, D., Tiessen, A. & Aiello, M. A Comparison of Open-Source Home Automation Systems. *IEEE Access*. **9** pp. 167332-167352 (2021)

[16] Anand, A., Krishna, A., Tiwari, R. & Sharma, R. Comparative Analysis between Proprietary Software VS. Open-Source Software VS. Free Software. *2018 Fifth International Conference On Parallel, Distributed And Grid Computing (PDGC)*. pp. 144-147 (2018)

[17] Apple. "Home App" apple.com. April 10, 2025. [Online]. Available: https://www.apple.com/ca/home-app/

[18] Amazon. "Alexa Smart Home" amazon.ca. April 10, 2025. [Online]. Available: https://www.amazon.ca/b?ie=UTF8&node=21497225011

[19] Hubitat. "Home Automation" hubitat.com. April 10, 2025. [Online]. Available: https://hubitat.com/home-automation/overview

[20] Fortinet. "What Is An Intrusion Detection System (IDS)? " fortinet.com. April 15, 2025. [Online]. Available: https://www.fortinet.com/resources/cyberglossary/intrusion-detection-system

[21] J. Díaz-Verdejo, J. Muñoz-Calle, A. Estepa Alonso, R. Estepa Alonso, G. Madinabeitia, "On the Detection Capabilities of Signature-Based Intrusion Detection Systems in the Context of Web Attacks," Appl. Sci., 2022, 12, 852. [Online]. Available: https://doi.org/10.3390/app12020852

[22] Mehari Msgna. "Anatomy of attacks on IoT systems: review of attacks, impacts and countermeasures." Journal of Surveillance, Security and Safety, vol. 3, pp. 150–173, Dec. 2022. [Online]. Available: https://www.oaepublish.com/articles/jsss.2022.07