UFPE

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Davino Mauro Tenório da Silva Júnior

A Flexible Approach For Creating and Enforcing Intrusion Detection Rules On Internet of Things Networks

Recife
2020

Davino Mauro Tenório da Silva Júnior

A Flexible Approach For Creating and Enforcing Intrusion Detection Rules On Internet of Things Networks

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação na Universidade Federal de Pernambuco como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de Concentração: Engenharia de Software
Orientador(a): Kiev Gama

Recife
2020

Catalogação na fonte
Bibliotecário Cristiano Cosme S. dos Anjos, CRB4-2290

S586f Silva Júnior, Davino Mauro Tenório da
A Flexible Approach For Creating and Enforcing Intrusion Detection Rules On Internet of Things Networks/ Davino Mauro Tenório da Silva Júnior. – 2020.
61 f.: il., fig., tab.

Orientador: Kiev Gama.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2020.
Inclui referências.

1. Engenharia de Software. 2. Internet das coisas. 3. Segurança. 4. Sistemas de detecção de intrusão de rede. I. Gama, Kiev (orientador). II. Título

005.1 CDD (23. ed.) UFPE - CCEN 2021 – 141

Davino Mauro Tenório da Silva Júnior

"A Flexible Approach For Creating and Enforcing Intrusion Detection Rules On Internet of Things Networks"

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação na Universidade Federal de Pernambuco como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 06/02/2020.

BANCA EXAMINADORA

Prof. Dr. Divanilson Campelo
Centro de Informática / UFPE

Prof. Dr. Everton Ranielly
Departamento de Informática e Matemática Aplicada / UFRN

Prof. Dr. Kiev Gama
Centro de Informática / UFPE
(Orientador)

I dedicate this work to God, my family, friends, and professors who supported me.

ACKNOWLEDGEMENTS

I am extremely grateful for all the support that was given to me to get this far.

First, I am grateful for the one and only God that walked and continues to walk with me, care for me, and help me every day of my life.

I am forever grateful for the support of my family. My grandmother, Áurea Lopes, which passed away in 2014 (before I began my Masters) but was, and still is, my greatest example and inspiration leading me through most of my time of pursuing my Bachelor's degree. My aunts, Eliane Rute and Jane Albuquerque, and my uncle, André Rui, for being with me on the bad and good moments and remaining my pillars in this world, landing me the support that I could never repay for.

I am also very grateful for the friends I encountered during my academic life, Luís Melo, Jeanderson Cândido, Igor Simões, and Sotero Junior. For all the laughs and funny moments at the INES (CIn-UFPE) laboratory, and also the cries (of joy) with the adversities, I thank you for being present and helping me get here. You deserve all the success in this world, count with me!

I am also grateful and blessed in finding a professor that later became an advisor but more importantly, a friend: Kiev Gama. Thank you for the unconditional support and understanding, especially on the personal problems that I faced during these two and a half years. I also thank all the professors I encountered during my journey: Marcelo d'Amorim, José Suruagy, Paulo Gonçalves, Darko Marinov and Atul Prakash; for all the work that we did together and support on the travels. I thank FADE, RNP and NSF for funding the project and my work.

Finally, I am forever grateful for the people not directly named here but which also supported me, you all know who you are! My personal, childhood and work friends, people that walked with me through this whole process. Thank you!

"Success is the sum of small efforts - repeated day in and day out." (COLLIER, 2013, p.188)

ABSTRACT

Securing IoT devices is not an easy task, but crucial due to the rapid growth of the IoT market in the recent past. For that, Network Intrusion Detection Systems (NIDS, or IDS for short) can be used to employ defenses on IoT environments by making use of rules to detect anomalies on the network traffic. Due to the nature of this market, usability must be treated as a crucial feature of these systems, especially on the process of creating the aforementioned rules. In this work, we present IoT-Flows: a platform built on traditional IDS's concepts such as network monitoring and generation of alerts once an anomaly is detected, but that focus on enabling users to create rules in an intuitive way with a user-interface (UI). To evaluate the proposed platform focusing specifically on usability, we compared it with Suricata, the most popular open-source IDS. We focused the evaluation on the process of creating the rules with a usability test of both systems where the participants were assigned the task of creating a rule to detect a popular distributed denial-of-service attack (DDoS) attack. After the test, we applied a System Usability Scale (SUS) questionnaire, which is tool to evaluate usability of a given system combined with open-ended questions and general observations throughout the test. After analyzing the results of both quantitative and qualitative feedback, we found the consensus among the participants was that Suricata, albeit providing a complete documentation, lacks flexibility for creating the rules due to its complex syntax and non-existing user-interface (UI), being a negative point particularly for non-experienced users. For the proposed system, IoT-Flows, the participants highlighted its UI and flexibility as its strongest points, providing an intuitive way of creating the rules. However, they also noted that creating the rules was slower if compared to Suricata. During this work, we observed that usability is indeed a crucial point that needs to be taken into consideration when developing security systems, especially if the systems target IoT contexts, where the presence of non-IT users is a common thing.

Keywords: Internet of things; Security; Network intrusion detection systems.

RESUMO

Garantir a segurança de dispositivos da Internet das Coisas (IoT) não é uma tarefa fácil, mas é crucial devido ao grande crescimento do mercado IoT nos últimos tempos. Para isso, Sistemas de Detecção de Intrusão (NIDS, ou IDS) podem ser usados como defesa em ambientes IoT fazendo uso de regras que detectam anomalias no tráfego de rede. Devido a natureza desse mercado, a usabilidade deve ser tratada como uma funcionalidade crucial nesses sistemas, especificamente no processo de criação destas regras. Neste trabalho, nós apresentamos IoT-Flows: uma plataforma construída a partir de conceitos tradicionais de IDS como monitoramento de rede e geração de alertas quando uma anomalia é detectada, mas que foca em permitir que usuários criem regras numa forma intuitiva por meio de uma interface de usuário (UI). Para avaliar a plataforma proposta focando especificamente em usabilidade, nós a comparamos com Suricata, a mais popular IDS de código aberto. Nós focamos especificamente no processo de criação de regras com um teste de usabilidade onde aos participantes foi dada a tarefa de criar regras para detectar um popular ataque distribuído de negação de serviço (DDoS). Depois do teste, nós aplicamos o questionário SUS, que é uma ferramenta para avaliar usabilidade de um dado sistema combinado com perguntas abertas e observações gerais feitas durante o teste. Após analisar os resultados do feedback quantitativo e qualitativo, o consenso entre os participantes foi de que Suricata, apesar de fornecer uma documentação completa, tem menor flexibilidade para criação de regras devido à complexidade de sua sintaxe e a não existência de uma interface para o usuário, sendo este um ponto negativo particularmente para usuários inexperientes. Para o sistema proposto, IoT-Flows, os participantes destacaram sua UI e flexibilidade como um dos pontos principais, provendo uma forma intuitiva de criar as regras. Apesar disso, eles também pontuaram que a criação das regras foi mais lenta se comparada ao Suricata. Durante este trabalho, nós observamos que a usabilidade é um ponto crucial que deve ser levado em consideração no desenvolvimento de sistemas de segurança, especialmente sistemas que focam em contextos IoT onde a presença de usuários fora da área de tecnologia é comum.

Palavras-chave: Internet das coisas; Segurança; Sistemas de detecção de intrusão de rede.

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

IoT	Internet of Things
IDS	Intrusion Detection System
SUS	System Usability Scale
OEQ1	Open-ended Question 1
OEQ2	Open-ended Question 2
OEQ3	Open-ended Question 3

CONTENTS

1 INTRODUCTION

The Internet of Things (IoT) is built on the concept of connecting physical devices capable of communicating with each other [2]. These devices, or "things", have the ability to use data from the surroundings, effectively exchanging this data through the Internet with other devices (or things).

The IoT paradigm comes with opportunities to built more secure environments (e.g., using cameras), improved medical care (e.g., using health devices such as real-time monitors) and industry efficiency (e.g., using sensors to monitor excluded areas without human intervention). These devices are often heterogeneous on its nature and contexts, and distributed on multiple areas. From (Smart) Homes, to (Smart) Industries and (Smart) Cities, devices that were once disconnected are now connected with embedded sensors capable of collecting real-time data.

Data collected by IoT devices is often sensitive enough to raise concerns involving security and privacy. This led to security on IoT to be a highly trendy topic in the last decade. Among possible reasons we can enumerate two. First, the IoT market has been increasing rapidly, with latest predictions on 24 billion connected devices by 2020 [3]. Second, devices that are part of day-to-day life of consumers have been actively targeted by attackers due to sensible information carried by those devices [4].

IoT devices are essentially network devices, therefore susceptible to traditional network attacks. This is demonstrated by the increasing number of incidents involving IoT devices for distributed denial-of-service (DDoS) attacks in the last year, with the number of IoT devices involved increasing at an astounding rate of 300% [5].

Although different approaches and tools exists to address IoT threats, not all of them are effective enough. Because IoT devices effectively work with data on multiple network layers, it is important for security solutions to cover all layers. However, solutions often act on a single network layer, forcing the user to have multiple tools in place as to prevent the attacks. Also, these solutions do not provide any means to extend the tool in question for new attacks, therefore lacking flexibility. Even if some of them do provide means to extend them, e.g., add new rules to an Intrusion Detection System to enable the tool to detect new attacks, this process is overly complicated even to IT domain's users [6; 7].

One type of solution addressing IoT threats is Network Intrusion Detection Systems (NIDS, or IDS, for short). Essentially, IDS are tools (either hardware, software, or both) used to monitor network traffic by looking for suspicious behavior [8]. Due to the rapid growth of the IoT market and with its market share being represented mostly by consumers on Smart Home contexts [9], it is important for these systems to be easy to use and extend. For example, in

a real world scenario where multiple IoT devices exist in a Smart Home, these systems make use of rules which can be created by the user and serve the purpose of detecting anomalies on the network traffic involving the devices. Because of that, **usability** stands as one of the crucial features of these systems, specifically on the process of creating the aforementioned rules. However, it is often neglected, as demonstrated by previous studies, which showed that usability was a difficult challenge for traditional Intrusion Detection Systems (IDS)s as the rules created and enforced on these systems proved to be non-intuitive and difficult to understand [7; 6].

In this work, we propose a platform that enables users to create rules in an intuitive way via a user-interface (UI), while keeping the core features of traditional Intrusion Detection Systems (IDS)s with network monitoring and detection of anomalies using a pattern-matching method, commonly used on Signature-based systems. Although the main contribution of this work (i.e., the platform) does not limit itself to a specific IoT context (e.g., Enterprise IoT, Industrial IoT and Consumer IoT), it is important to notice that the platform enables even non-IT users to be able to create rules capable of detecting IoT threats in a seamless manner, which is a common scenario that exists on Consumer (or Smart Home) IoT context.

To evaluate the proposed platform focusing specifically on usability, we compare it with a traditional and the most popular [10] open-source IDS, Suricata [11]. We focus specifically on the process of creating the rules with a usability test of these systems where the participants were assigned the task of creating a rule to detect a popular DDoS attack. After the test, we applied a System Usability Scale (SUS) questionnaire (which is a tool to evaluate usability of a given system) together with open-ended questions as to gather both quantitative and qualitative feedback.

The remainder of this work is structured as follows:

- Chapter 2 presents a background of the solutions targeting IoT network threats, focusing specifically on Intrusion Detection Systems (IDS);
- Chapter 3 introduces the target platforms and discuss their core process for creating the rules;
- Chapter 4 describes the usability test used for comparing IoT-Flows and Suricata focusing specifically on the process of creating the rules;
- Chapter 5 discuss the results of the usability test in terms of both qualitative and quantitative feedback;
- Chapter 6 presents relevant work related to IoT security in general, focusing on IDS and and Usability;

- Chapter 7 draws our conclusions and presents future work.

2 BACKGROUND

The IoT has come from promise to reality and forms the state-of-the practice for constrained networks today. In simple terms, an IoT network is built by multiple interconnected embedded devices which can be controlled either locally or remotely using mechanisms such as a mobile app [12]. Devices range from smart lamps to washing machines and effectively replace traditional commodities of the everyday house, also appearing in critical areas such as hospital's health monitors and autonomous car's sensors.

The concept of IoT cannot be separated from network, as IoT devices are essentially network devices on their core. An IoT network and the smart environment it creates is built around sensors and actuators that work together to gather and operate with different types of information. Although the nature of this information varies, it is, more often than not, sensitive in such a way that it raises concern to the adoption of the IoT paradigm [13]. Consider for example a scenario where an attacker gains access to simple information such as a smart light schedule, e.g., the exact time when it turns on/off regularly. With this information in hand, a robber can infer the time when the house will probably be empty. By hacking the house's smart lock, for example, the attacker essentially gains access to a empty house. This scenario is not far from reality as IoT devices such as smart lights and locks are often plagued with several vulnerability issues, as reported in previous studies [14; 4]. This demonstrates that Security and Privacy has not been taken into consideration by manufacturers of IoT devices when releasing products to the general public [4].

IoT devices, being network appliances, are susceptible to various security threats that plagued traditional computers over the years such as distributed denial-of-service (DDoS) attacks. The combination of constrained resources (which leads to less security mechanisms to avoid hacking those IoT devices) and also its greater numbers if compared to computers is dangerous. Both new and traditional attacks plague IoT devices, as shown by Nawir et al. [1]. This was also made public by the (in)famous Mirai botnet attack that occurred on 2016 (the largest of its kind at the time), where a series of DDoS attacks were launched across the US using common vulnerabilities found on IoT smart cameras and DVR players [15].

In the next sections, we will focus on the security aspect of IoT, first presenting the different type of network attacks, and then, the tools and solutions that exists to mitigate those attacks.

2.1 IOT SECURITY

Different approaches and tools addressing security of IoT networks were presented in the recent past. Due to a large number of attacks, it is important to categorize them to have a better understanding and appropriately provide solutions to detect and neutralize such threats. Nawir et al. [1] proposed a categorization for the different types of IoT attacks using a fine-grained classification based on TCP/IP network layers. We extended the work using a non-exhaustive search for different solutions and tools proposed to tackle each one of these type of attacks. In the following subsections, we first revisit Nawir et al. categorization on the different network attacks targeting IoT devices. Then, we present the state-of-the-art solutions for each attack found in our non-exhaustive search.

2.1.1 Vulnerabilities

The three pillars of Security for any type of system (including IoT) are the cornerstone to understand the different types of vulnerabilities on these systems. The CIA Triad, represented by *Confidentiality*, *Integrity* and *Availability* have been widely used on the academic world and shaped how security is implemented [16].

Confidentiality, as first introduced by the Bell-La Padula Model [17], established rules to protect and limit access to information, focusing on granting access only to those who "need to know". Integrity, introduced by the Biba Model [18], focused on preventing data modification by unauthorized parties and maintain data consistency. Availability, introduced by the Denning intrusion-detection model [19], was based the basics for a real-time intrusion detection system, or IDS (which is later discussed on this Chapter). The main goal of this model was to show how suspicious use-patterns could be detected by closely monitoring the system.

The vulnerabilities described below are centered on IoT systems and goes against one or more of the security pillars. Table 1 shows the attacks distributed on the different TCP/IP layers: Physical, Data Link, Network, Transport and Application; as described by Nawir et al. [1].

In the **Physical** layer, *Jamming* consists on constantly, deceptively or randomly compromising the communication channel with meaningful data, whereas *Tampering* is physically attacking the device [20].

On the **Data Link** layer, attacks are classified as collision, resource exhaustion, and unfairness. *Collision* consists of sending packets at the same time of legitimate data packets, harming specific packets instead of the whole channel [20]. Resource *Exhaustion* is a type of attack that force the device to consume its resources deliberately, for example, sending multiple requests to devices that use batteries until the battery dies. Some MAC protocols give priority

to devices that are very low on battery, the *Unfairness* attack consists in creating a low battery device to have priority when sending packets denying real traffic [20].

It is, however, on the **Network** layer that the majority of attacks are reported. The attacks based on *Spoofed, Altered or Replayed routing information* are based on unprotected ad-hoc networks, where the routing can be compromised. *Selective forwarding* is related to a denial of service on a specific node with packets being dropped. *Sinkhole* consists of a network node pretending to have the shortest path to other node in order to drop packets, modify data or interfere in clustering algorithms. *Sybil* attacks are based on a node obtaining multiple fake identities and misleading other nodes on the network. *Wormholes* attacks are based on different devices on distinct connected networks, where the data from one network is sent to another in order to create real, but misleading information. *HELLO* messages are used by some protocols to establish connection or neighborhood relation and are used with a high power transmitter to create fake proximity relations. *Acknowledgment spoofing* uses ack messages to pretend that a disabled node is alive or the connection between two nodes is strong than it apparently is.

Regarding the **Transport** layer, *Flooding* consists of exploring the natural vulnerabilities of TCP and UDP protocols as the UDP have no flow control and the TCP protocol is vulnerable to SYN (new connection request) flood. De-synchronization is the interference of an attacker in order to interrupt an active connection between two nodes, using fake packets containing error or specific control flags [21].

On the **Application** layer, *Reliability* issues are often related to execution problems like a buffer overflow. *Cloning* is the capability of attackers to steal information from devices or steal the device credentials.

2.1.2 Solutions

Building on the taxonomy of the IoT vulnerabilities previously described, we did a non-exhaustive search with the main goal of distributing the state-of-the-art solutions on the different network layers presented [22]. Table 1 shows the results.

Starting on the **Physical** layer, Namvar et al. presented a novel anti *Jamming* strategy which promotes an IoT controller to protect the IoT devices against malicious radio jammers [23]. For *Tampering* attacks, a team at the NEC Corporation developed a lightweight-architecture for tampering detection on IoT devices, using real-time inspection with no impact on the device normal operations [24].

On the **Data Link** layer, *Collision* attacks are similar to jamming, thus inheriting the same solution. Ruckebusch et al. presented a new architecture designed with IoT in mind that mitigates *Exhaustion* attacks, an after-effect of the previous attacks [25]. Regarding *Unfairness* attacks, Djedjig et al. presented a trust-based defense model to detect malicious behavior,

calculating trust levels for participating nodes [26].

On the **Network** layer, solutions often tackle more than one type of attack. SVELTE is one such case, applying real-time intrusion detection to detect *Spoofing* and *Sinkhole* attacks [27]. Huijuan et al. describe a system that uses watermarked packets to identify whether a network node is doing a *Selective Forwarding* attack, using a trust value to identify how many marked packets are dropped related to normal packet loss, skipping nodes with low trust [28]. Due to the fact of *Sybil* attacks are based on creating fake nodes in networks, Sohail et al. developed a system based on device mobility that is capable of differentiating if a node is fake or not by reading the RSSI (Received Signal Strength Indication) pattern [29]. For Pongle et al., the *Wormhole attack* is very location related, so the developed system periodically broadcasts the nearby RSSI. Then, this information is used by other devices to infer if a node is nearby or not, classifying it as compromised [30]. Singh et al. also used the RSSI but to mitigate *HELLO* attacks, considering that devices have a homogeneous signal strength, any other value too different is considered strange. If the value is close to the standard, the node will be asked to solve a puzzle that increases exponentially in difficulty per HELLO message. If the node fails to answer in an assigned time, the node is labeled as strange [31].

The *Flooding* attack is commonly used on the **Transport** layer for distributed denial-of-service (DDoS) in IoT environments. Dao et al. presented how attack behavior learning can be used to detect flooding attacks with smart filters distributed on the network [32]. *Desynchronization* attacks can be mitigated using authentication protocols like the one proposed by Fan et al. using an RFID protection scheme [33].

In the **Application** layer, common defenses include access policies to control information flows between applications and IoT devices. One such solution was presented by Demetriou et al. with HanGuard, applying SDNs to enforce policies on Smart Home networks [34]. Another approach involves increasing data security on IoT applications. Fernandes et al. developed FlowFence, a framework that enables developers to secure function executions involving sensitive data in Android-OS's created processes [35].

2.2 INTRUSION DETECTION SYSTEMS

The state-of-the-art solutions presented on Table 1 are essentially ad-hoc solutions tailored specifically for each type of attack. There is one type of solution, however, that stands out as it usually focus on different network layers and cover different type of attacks: **Network Intrusion Detection Systems**, or (N)IDS for short.

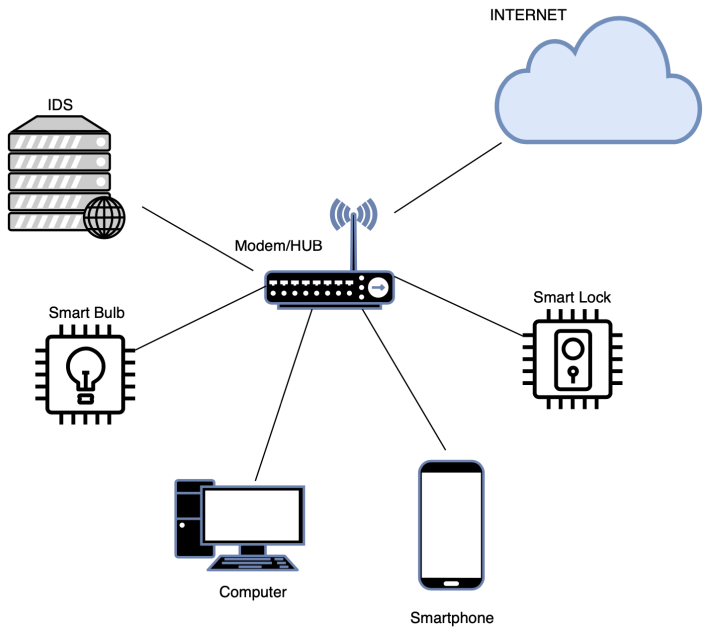
An IDS is a tool that analyze network traffic by monitoring network nodes (e.g., the IoT devices) and its data packets with the goal of detecting suspicious behavior [8]. This analysis is usually done by identifying signature of well-known attacks. Once detected, the IDS triggers

Layer	Attacks	Methods/Strategies	State-of-the-art solution
Physical	Jamming	Creates radio interference and exhaustion on IoT devices	Namvar et al. [23]
	Tampering	Creates compromised nodes	NEC Corporation [24]
Data Link	Collision	Simultaneous transmission of two nodes on the same frequency	Namvar et al. [23]
	Exhaustion	By repetetively colliding the nodes	Ruckebusch et al. [25]
	Unfairness	Using above link layer attacks	Nabil et al. [26]
Network	Spoofed, altered or replayed routing information	Creates routing loops, extend or shortening sources routes, attracting or repelling network from select nodes.	Raza et al. [27]
	Selective forwarding	Choose what information to gather before transmitting.	Deng et al. [28]
	Sinkhole	Compromised node tries to attract network traffic by fake advertising its fake routing update	Raza et al. [27]
	Sybil	Single node duplicates its node to be in multiple locations.	Abbas et al. [29]
	Wormholes	Selectively tunneling or retransmit information to the IoT devices.	Pongle et al. [30]
	HELLO flood	Uses HELLO packets as weapon to launch the attack on IoT system	Singh et al. [31]
	Acknowledgement spoofing	Spoof the link layer acknowledgments for overhead packets.	Raza et al. [27]
Transport	Flooding	Repeat the request of a new connection until the IoT system reaches maximum level.	Dao et al. [32]
	De-synchronization	Disruption of an existing connection.	Fan et al. [33]
Application	Clock skewing, Selective message forwarding, Data aggregation distortion	The adversaries usually masquerade like normal behavior in IoT system. Attackers also can still choose a message that he/she intend in the IoT system and launched their own malicious activities.	Demetriou et al. [34] Fernandes et al. [35]

Table 1 State-of-the-art solutions found in our non-exhaustive search, categorized by the IoT Attacks taxonomy defined by Nawir et al. [1]

an alert to the users or even an automated response such as disconnecting a suspicious device from the network. Figure 1 shows a simple IDS setup, with devices such as IoT's smart bulbs, smart door locks, a computer and smartphone connected to a wireless modem/HUB. An IDS system is also connected to the HUB, monitoring the network traffic coming from the devices.

Figure 1 A simple IDS setup



There are mainly three types of IDS: Anomaly-based, Specification-based, and Signature-based, each focusing on different characteristics while monitoring the network traffic to detect threats [36]. In this work, we focus on the Signature-based IDS type due to its popularity on the IoT context [8].

Anomaly-based IDS. An anomaly represents a deviation of an expected behavior, which on network contexts comes from monitoring regular traffic, devices, users, etc. Events can be either static or dynamic, e.g., failed login attempts and count of emails sent. An Anomaly-based IDS compares normal traffic and looks for these type of events to recognize possible threats [37].

Specification-based. Specification-based IDS are similar to Anomaly-based IDS in the sense that they also detect a deviation from the expected behavior. However, instead of using a predefined set of events to detect an anomaly, this type of IDS uses manually developed specifications that capture legitimate system behaviors [38].

Signature-based IDS. A signature-based IDS employs a pattern matching strategy where the system uses a predetermined set of well-known patterns to detect whether the incoming monitored network packets are malicious or not [36]. Tools such as Zeek [39], Snort [40] and Suricata [11] are examples of these type of systems. They enable users to create their own rules and share them through the community, amplifying the capability to detect network threats as users can download the shared *rules*.

Signature-based, similarly to other types of IDS in general, have some limitations however. Due to the rapid growth of network traffic and complexity of network attacks, it becomes increasingly difficult for a signature-based IDS to keep up with current threats [41]. Recall that signature-based IDS are based on patterns (or rules), which are applied in real-time to the network traffic being monitored as to detect suspect activity, generating alerts to do so. Failing to generate an alert during an attack (false negative) or generating alerts to benign network traffic (false positive) can be both critical to the devices being monitored, possibly compromising the whole network.

These systems need to be extensible as to keep up with current threats. With the rules (or patterns) created representing the core of an IDS, they should be as intuitive as possible, which forms the base concept of usability of an IDS [6; 7]. Measuring usability of an IDS (or any system for that matter) presents a challenge as usability itself is a subjective concept and not easily measurable like variables such as processing time, false positive/negatives and memory consumption, which previous works covered in detail [36; 42; 37; 43].

3 TARGET PLATFORMS

In this chapter, we describe two IDS platforms, Suricata, and IoT-Flows. The rationale for focusing on these specific systems is that the former is the most popular traditional IDS [10] not targeting IoT devices specifically, with the latter focusing on the IoT context, both of them being signature-based IDSs. We give a general overview of both platforms while focusing on the mechanism to create the rules, which is the focus of this work.

3.1 SURICATA

Suricata [11] is an open-source Signature-based IDS originally conceived in 2009 by the US Department of Homeland Security along with a consortium of private companies forming the Open Information Security Foundation (OISF), with its first version coming in 2010.

Suricata was based on Snort [40] in such a way that the rules written on Snort can also be used on Suricata interchangeably. One improvement over its predecessor, however, is that Suricata incorporates a new Hyper-Text Transfer Protocol (HTTP) parser capable of examining HTTP traffic for traditional attack-threats that were known for circumventing Snort along with older IDSs.

3.1.1 Architecture

Suricata is built around a multi-thread architecture, which enables smarter decisions on how to split processing and coordinate pattern matching between the threads while monitoring network traffic. For that reason, Suricata is able to take advantage of multiple operating CPUs while executing its pattern-matching on multiple threads but on the same detection engine [44].

Figure 2 shows the multithread architecture of Suricata. Once acquiring the data packets from the network, Suricata decode the packets and begins a pattern-matching process using its rules in multiple threads, trying to identify signatures of well-known threats. The output are the alerts generated once a pattern is matched, or none at all for benign network traffic.

It is important to clarify that, although being the most popular open-source IDS (together with Snort) [10], there is a clear lack of documentation available regarding the platform's architecture. Therefore a perspective about the components that comprise Suricata will not be presented.

3.1.2 Rules Mechanism

A rule on Suricata is the defacto method for detecting threats using this platform. Being a Signature-based IDS, Suricata uses these rules (or signatures) to match them against the network traffic [45].

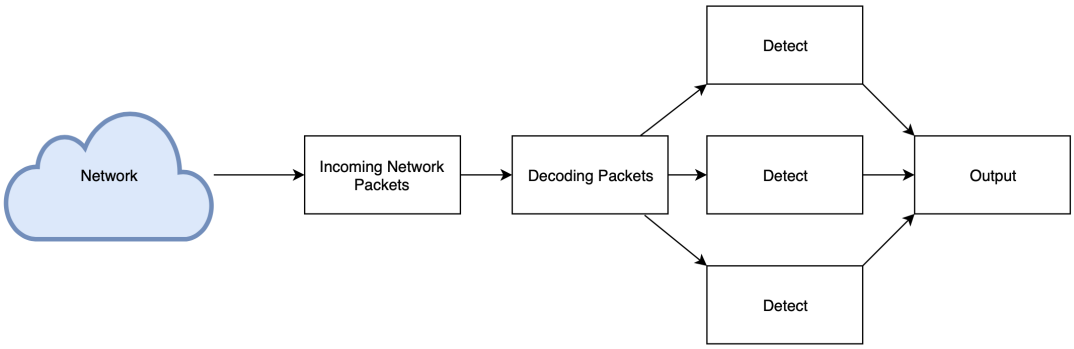


Figure 2 Suricata Multithread Architecture

```
alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"ET DOS ICMP Path MTU lowered below
    ↪ acceptable threshold"; itype: 3; icode: 4; bytes_test:2,<,576,6; byte_test:2,!=,0,7,
    ↪ sid:2001882; rev:10;)
```

Listing 3.1 Suricata's Rule to detect an ICMP Flood Attack

A rule consists of three components:

- Action: Determines what happens when the signature is matched;
- Header: Defines the protocol, IP addresses, ports and direction of the rule;
- Rule Options: Define the specifics of the rule.

Consider the rule shown at Listing 3.1, which serves the purpose of generating an alert in case an ICMP Flood Attack is detected. In this example, the part "alert" is the **action**, "icmp $EXTERNAL_NET any -> $HOME_NET any" is the **header** and the remaining text are the **options** of the rule. For the **action**, you have four different values (pass, drop, reject and alert), all self explanatory and reflecting actions to be taken involving the network packets. For example, the *alert* word means that it would trigger an alert when a signature matches a network pattern. On Suricata, the alert means a message on the console, with additional actions being possible using external solutions such as Firewalls that capture Suricata's events.

The part concerning the header options contains three main points: (i) protocol; (ii) source and destination addresses; and (iii) port addresses. The **protocol** tells Suricata which network protocol it looks while trying to match the signature to the network packets. Currently Suricata accepts four basic protocols TCP, UDP, ICMP and IP network packets together with seven different applications protocols such as HTTP and SMTP. The **addresses and ports** tell Suricata which direction to consider when trying to match a signature to a certain network flow. Still considering Listing 3.1, the "$EXTERNAL_NET" part represents the source of the traffic whereas "$HOME_NET" represents the destination of the traffic (notice the direction of the

directional arrow between them). The variables "HOME_NET" and "EXTERNAL_NET" tells Suricata to consider all addresses on the local and external network, respectively. Notice that, instead of variables, one can also choose to specify IP addresses (both IPV4 and IPV6) as well as IP ranges. The user also can specify which specific port to consider, both on source and destination. The word *any* can be used interchangeably here, meaning **any** port.

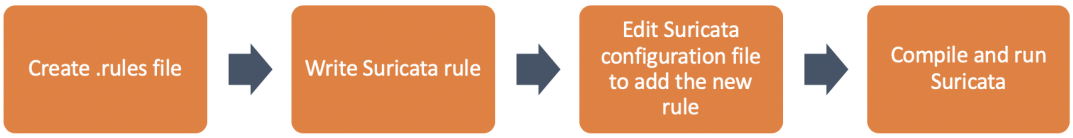


Figure 3 Suricata's Process for Creating a Rule

The process of creating a rule for Suricata can be seen in Figure 3. First, the user should create a file adding the name of the newly created file with the extension ".rules", that denotes a rule to be loaded by the Suricata platform. Then, the user writes the signature on this file as in Listing 3.1. Notice this assumes that the user had either analyzed the network traffic as to obtain sufficient information for identifying a signature of an attack or obtained this information elsewhere. After creating the rule file, the user edits Suricata's main configuration file (usually located on */etc/suricata/suricata.yaml*) adding the name of the newly created file (as seen in Figure 4). Finally, the user (re)compile Suricata by restarting it so it can reload the configuration file and the newly created rule.

3.2 IOT-FLOWS

The IoT-Flows platform was originally conceived on a project of the same name in 2019 as a means to ensure secure communication flows in IoT networks. It naturally evolved to fill a gap on IDSs, as these systems, more often than not, do not cover all network layers or provide an easy way to extend them for new attacks as discussed at chapter 2. The project [46] is part of a collaboration between USA and Brazil with the National Science Foundation (NSF) and Rede Nacional de Ensino e Pesquisa (RNP) as its funding agencies, respectively. The system was designed and built on the Centro de Informática (CIn) of the Federal University of Pernambuco, with support of researchers from the University of Michigan (UMICH) and the University of Illinois at Urbana-Champaign (UIUC).

The IoT-Flows platform is built on the principle of autonomous computing [47]. The importance of autonomic systems for IoT security was already pointed out in a review that gathers different solutions emphasizing on autonomic computing to mitigate IoT threats [48], although with a limitation of typically addressing just one or sometimes two layers from TCP/IP and

```
1808  ##
1809  ## Configure Suricata to load Suricata-Update managed rules.
1810  ##
1811  ## If this section is completely commented out move down to the "Advanced rule
1812  ## file configuration".
1813  ##
1814
1815  default-rule-path: @e_defaultruledir@
1816
1817  rule-files:
1818    - suricata.rules
1819    - new_rule.rules
1820
1821  ##
1822  ## Auxiliary configuration files.
1823  ##
1824
1825  classification-file: @e_sysconfdir@classification.config
1826  reference-config-file: @e_sysconfdir@reference.config
1827  # threshold-file: @e_sysconfdir@threshold.config
1828
1829  ##
1830  ## Include other configs
1831  ##
1832
1833  # Includes.  Files included here will be handled as if they were
1834  # inlined in this configuration file.
1835  #include: include1.yaml
1836  #include: include2.yaml
1837
```

Figure 4 Suricata's Main Configuration File

providing no means to extend to new threats. Besides the architectural approach, the concepts of self-healing and self-protection are the main aspects taken from the *self-** principle.

IoT-Flows is a system that employs this autonomic approach in a distributed architecture with multiple components, each one possessing a specific and unique responsibility. Following an old design principle–Separation of Concerns–one can see each component as a module addressing a different concern, i.e., a problem [49]. A component can, for example, deal with the problem of monitoring the network in a distributed manner and filter the network traffic, while another component analyzes this data and look for signs of suspicious behavior. This separation of concerns alleviates the complexity of a security enforcement system monitoring different networks with heterogeneous devices. The architecture of the system is based on the MAPE-K, a traditional architecture blueprint originally designed for self-adaptive systems [47]. This architecture consists on five main components (seen in Figure 5): *Monitor*,*Analyze*,*Plan*,*Execute* and *Knowledge*. The *Monitor* component collects data from the managed resource, which is then passed to the *Analyze* component that performs complex data analysis and is influenced by the *Knowledge* component. The *Plan* component holds the actions needed to achieve a predefined goal. Finally, the *Execute* component changes the behavior of the resource that is managed based on the actions received from the *Plan* component [47].

The IoT-Flows platform focuses on surveilling communication between IoT devices. It

acts on the different network layers, providing a multilayer defense for IoT environments. The system is able to monitor the traffic on the different WiFi networks that the smart devices are connected to. It also provides extensibility, allowing the user of the system to incorporate new attacks into the defense model, in the form of Complex Event Processing (CEP) rules, which consists of an approach that allows the system to analyze streams of data in real-time. Currently, we have developed patterns against attacks in three TCP/IP layers: Network, Transport, and Application layers. For instance, while monitoring the network, the system is able to detect that an IoT device is being targeted for Acknowledgement Spoofing with a fake device trying to masquerade the official device. At the same time, on the transport layer, the attacker would be flooding the IoT device with multiple requests, also acting on the application layer, trying to masquerade normal behavior requests, like turning the device on or off. The system is able to detect any of these behaviors while monitoring the network traffic and applying pre-configured rules that analyze the packets being sniffed. Once a suspicious behavior is detected, the system can alert the user or block all requests directed to the IoT device in question, therefore stopping the attack.

IoT-Flows allows the user to download new security patches that provides detection of new attacks while also providing manual configuration if needed. We have tested the approach of having an extensible mechanism based on Complex Event Processing rules that allows to easily include the identification of new attacks. Some drawbacks are the need to understand the rule language of the CEP Engine, [1] Esper, and understanding the metadata of the packet structure in order to write the rules. The process of creating the rules is explained in detail on subsection 3.2.2.

3.2.1 Architecture

Based on the architectural parts of the MAPE-K blueprint [47], IoT-Flows is formed by the components shown in Figure 6. We describe each component below according to their matching role on the MAPE-K architecture (Figure 5). Notice that the Knowledge (K) MAPE-K component is absent as we currently do not make use of any approach (e.g., machine learning) to analyze historical data on IoT-Flowsś architecture, although this is envisioned to be explored in future work.

Flow Monitors. The flow monitor relates to the *Monitor* component on the MAPE-K architecture. Its main responsibility is to aggregate and filter network traffic data, i.e., the resource that is monitored, generating “events” to be analyzed by the Analyzer component. All flow monitors are connected to the Analyzer component and, after decoding the network packets, periodically send the aggregated traffic data to the Analyzer component in a platform-

[1] www.espertech.com/esper

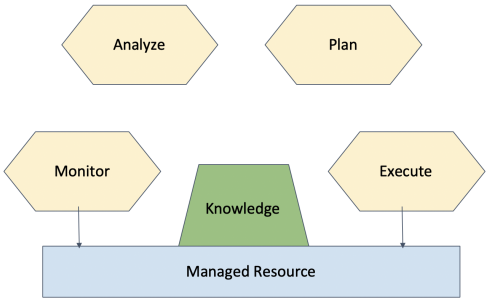


Figure 5 Original MAPE-K Architecture.

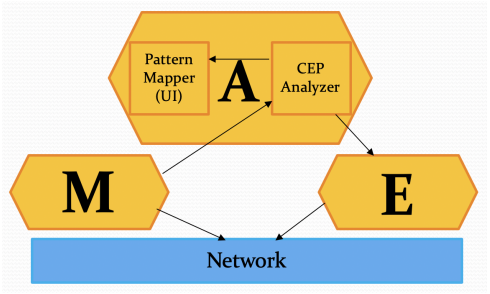


Figure 6 Architecture of the IoT-Flows platform.

specific format containing the possible network packet fields. Figure 7 shows the JSON message that is sent to the analyzer by the monitors containing the possible network packet fields that can be used on the creation of rules.

```
{
  “srcAddrMac”:  “MAC address”,
  “dstAddrMac”:  “MAC address”,
  “srcAddrIP”:  “IP address”,
  “dstAddrIP”:  “IP address”,
  “srcPort”:  string,
  “dstPort”:  string,
  “SYNFlag”: string
  “ACKFlag” : string
  “ICMPType”:  “string”,
  “ICMPCode”:  “string”,
  “Timestamp”:  “yyyy-mm-ddThh:mm:ss.ms UTC”,
  “PacketSize”:  int,
  "Content": string,
   ...
}
```

Figure 7 JSON representing the fields of a network packet.

IoT-Flows has 2 types of monitors, one targeting the first and second TCP/IP layers (Physical and Data Link), and the other focusing its efforts on the remaining layers (Network, Transport, Application). The rationale for that is because monitoring the top TCP/IP layers often involves analyzing encrypted network packets. Thus, it is imperative to the top-layer monitors to be connected directly to the router that has the encryption key, enabling the monitors to relay the unencrypted information to the Analyzer. For that reason, the top-layer monitors are connected

```
select * from NetworkPackets where 'ICMPType' = 3 and 'ICMPCode' = 3 group by 'dstAddrMac'
    ↪ having count(*) > 250
```

Listing 3.2 Translated query on CEP Analyzer to detect an ICMP Flood Attack.

directly to the router. Monitoring the physical and data link layer, however, does not involve any encrypted information from the network packets. This enables these type of monitors to be distributed on the network without being directly connected to the router, monitoring the wireless network and its devices.

Pattern Mapper. This is a core component of the platform and the focus of this work with the largest contribution from the authors (together with the Analyzer). This component relates to the Analysis part of the MAPE-K architecture and has the goal of maintaining the rules of the IoT-Flows platform.

A *Pattern* maps a rule to a certain action. For example, one can create a rule to block flows from a certain IoT device to an unwanted address. The system detects this behavior trying to match the pre-defined rules to the network data received from the *Monitors*. Then, once matched, the system performs a pre-configured action, e.g., alert the user or block the network request. A support application allows IoT-Flows's users to write these rules and deploy them directly into the system in real time. This component also has an API which is consumed directly by the Analyzer and has the responsibility of storing and managing all rules created on the system.

CEP Analyzer. This component also refers to the Analysis part of the MAPE-K architecture. Its main responsibility is to receive the aggregated traffic data from the monitors and apply Complex Event Processing (CEP) to match these data against pre-defined patterns. These patterns are gathered from the Pattern API every X interval (the "X" here being configurable) and translated to the rule language of the CEP Esper's Engine [2]. For instance, Listing 3.2 shows the final pattern to be applied to the traffic data. After gathering the rules listed at Figure 10 for the ICMP Flood attack, the analyzer combines them in a pattern that is applied to every network packet (translated into events by the Esper engine). The resemblance with an SQL-like query is expected due to the nature of the CEP Engine, however it is important to note that the Esper language contains several additional operators that could be used, please refer to the documentation for a full list [50].

The main process of the IoT-Flows uses the CEP Analyzer at its core. First, the analyzer gather all rules stored on the Pattern API at startup, and at a timely interval after that. Then, the analyzer receives the network traffic data from the flow monitors in a JSON message format (as described earlier). These messages are essentially an array of network packets in that JSON

[2] http://www.espertech.com/

structure, which are loaded into the CEP Engine and pattern-matched in real time with the configured rules. If the engine finds a match, it raises an alert (seen at Figure 8).

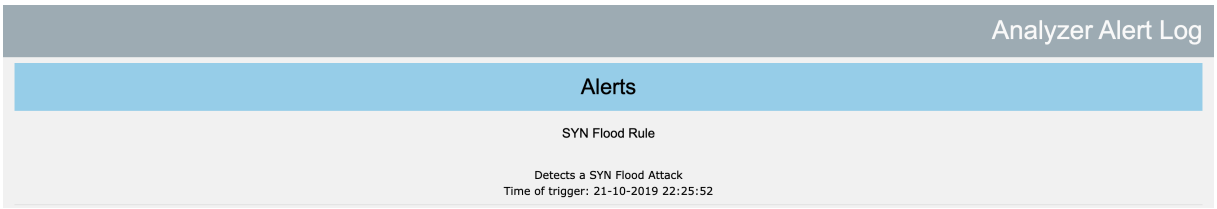


Figure 8 Analyzer Web Server

Execute. The *Alerter* and *Router* components are logically the same in terms of responsibility. The *Alerter* component is responsible for generating an email or SMS alert to the user after a suspicious activity is detected or the component is configured to do so under certain circumstances, e.g., if a smartphone tries to connect to an IoT device. The *Router* component is responsible for applying enforcement policies to the devices. For instance, it can block a request to an unwanted endpoint originating from an IoT device on the network. At the moment of writing, only the *Router* is used to enforce rules such as disconnect an IoT device from the network once its involved in suspicious behavior. However, future work envisions usage of multiple execute components as to complement the detection part of the IoT-Flows platform.

3.2.2 Rules Mechanism

A rule on the IoT-Flows platform is represented by a pattern. The **Pattern Mapper** enables users to seamless create a rule through a UI, while also having an API that is actively consumed by the **CEP Analyzer** component in real time as to reconfigure the rules being used on the IoT-Flows platform.

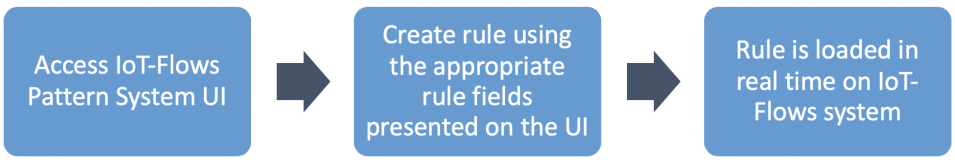


Figure 9 IoT-Flows Process for Creating a Rule

Figure 9 illustrates the process for creating a rule or pattern. Users can see which rules are active at any moment while also create, edit and delete rules, as per Figure 10. These rules represent the direct input for the Analyzer component (together with the network data being monitored). They are built internally as *SQL*-like queries and matched against the network packets being monitored.

Figure 10 List of Rules on IoT-Flows

For the example on Figure 10, the rules being showed are used for detection of the ICMP Flood attack (refer to section 4.2 for a full description of this attack). There are three rules, the first one being the main one and representing a pattern that evaluates whether the field *ICMPType* of the packets (Term) "is equal to" (Operator) the value 3 (Compared Value). The second rules extends the first and has the same goal, this time using the *ICMPCode* field. Finally, the last rule tells the Analyzer component to group the network packets by their MAC address (represented by *dstAddrMac* on the packet) and look for matches of these three rules on 250 or more cases (Type Variable). These rules are compiled internally by the Analyzer and together form the full signature for detecting an ICMP Flood Attack.

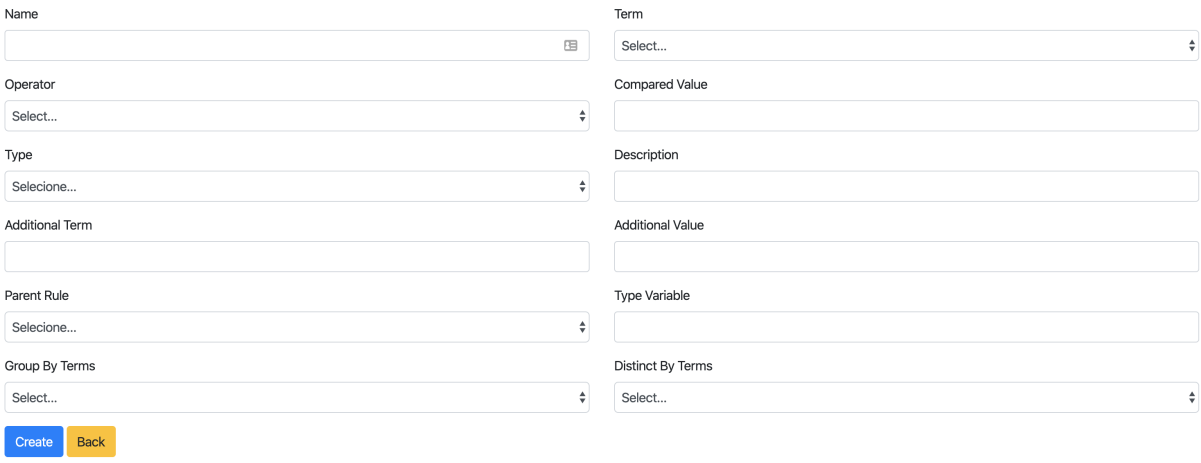


Figure 11 Form used to create a rule on IoT-Flows

Figure 11 shows the UI that enables the users to create a rule on IoT-Flows and contains the necessary network packet's fields. Considering the same fields that were described in the earlier example for the ICMP attack, the *Term* field has a selection of all possible fields that can be analyzed on the network packet. For instance, it contains flags such as `ICMPType` and `SYNFlag` (used for identifying ICMP and SYN packets, respectively); and addresses fields such as `srcAddrMac/dstAddrMac` and `srcAddrIP/dstAddrIP`.

The *Operator* field contains a selection of possible operators, e.g., "=", "!=", "<" and ">" and is used to compare two network packet fields. The field to be compared can be seen in *Compared Value*, which follows the same pattern as the *Term* field. The remaining fields can be used to create more complex rules. For instance, if one needs to identify a certain pattern in a group of packets coming all from the same device (as the ICMP example), they can create a rule using the field *Group By Terms* with the selected packet field *srcAddrMac* and the number of occurrences on *Type Variable*.

In the next chapter, we present an evaluation of usability of Suricata and IoT-Flows focusing on the target of our work, i.e., the creation of rules.

4 USABILITY TEST

While there are several studies that evaluate IDSs in terms of correctness, false positive/negatives and performance [51; 52; 44], to the best of our knowledge, no previous work focused on the usability of these systems. This is mainly because usability is an often subjective topic. For example, one can analyze usability looking at different factors such as productivity, satisfaction, accessibility and learnability, to name a few [6].

An IDS system needs to be easy to use and extend, with the creation of rules being a determining factor here [7]. We focus on the process of creating a rule to compare the platform proposed on this work, IoT-Flows, against the most popular open-source IDS, Suricata [11].

To compare IoT-Flows and Suricata, we conducted a usability test evaluating the creation of rules on these systems. This experiment was done with participants who did not have any prior knowledge of Intrusion Detection Systems (IDS)s. We first demonstrated the selected tool using the ICMP flood attack (see section 4.2 for details). While demonstrating the creation of the rule for this attack, we focused on the syntax, not considering other factors such as configuring the rule on the platform. This is mainly because, by doing so, we leave no margin to configuration-complexity to interfere with the experiment and feedback from the participants.

After demonstrating the creation of the rule, we presented another DDoS attack, SYN Flood [53] (also described in 4.2), focusing on the intrinsics of the attack and asking the participants to create a rule to detect the attack once we launched it on the network. We split the total time of the experiment with each tool in such a way that the participants would have a chance to test the rule in the platform. Each test consisted on running the attack targeting a device connected to the network and evaluating whether the tool would detect the attack with the created rule. We describe the method used in detail next.

4.1 METHODS

In the next subsections, we give an overview of the experimental design and introduce the questionnaire that was applied to the participants. Then, we elaborate on the attacks that were used as well as the tasks that were assigned to the participants on the experiment.

4.1.1 Experimental Design

The experimental design is the core of any experiment and defines which variables will be examined, which data will be collected and how the experiment should be executed and repeated (if needed) [54].

The goal of this study is to compare the usability of Signature-based Intrusion Detection

Systems (IDS)s focusing on the creation of rules, which is one of the core aspects of these systems. After the goal is defined, we can determine which factors the experimental subjects, i.e., the participants, will be subjected to evaluate the experimental units, i.e., the platforms [54]. In this work, the participants were assigned the task of creating the same rule on both IDSs on three rounds (or time intervals). The first one consisting on 30 minutes and the remaining two consisting on 15 minutes each.

Another core concept of the experiment is to decide which data will be collected. On this work, the collected data consisted of both qualitative and quantitative variables, forming a mixed-method to assess the usability of the systems [55].

During the experiment, we evaluate the following quantitative variables:

- Did the participant write the correct rule?
- If so, how many rounds were needed?
- How many times did the participant ask for help?

We also requested the participants to answer a System Usability Scale (SUS) questionnaire, which is a straightforward and reliable tool to measure usability of systems [56].

On the same questionnaire, we included three open-ended questions as to gather additional information not covered by the SUS questionnaire and general feedback of the experience as a whole with a qualitative feedback on the platforms.

To evaluate the answers, we used a process known as **Coding**, which is a heuristic with the goal of discovering the meaning of individual data by labeling, classifying and reorganizing the qualitative data in different categories for analysis [57]. The coding process is done by selecting pieces of *code* (represented by a word or small phrase) that capture the essence of a portion of data, in this case, the answer of the participants. By doing that, we synthesize these answers classifying the codes that were found them into similar clusters which share the same category. It is important to note that different researchers can analyze the same piece of data and develop different codes, depending on their interpretation. However, this proves to be a thorough process to analyze qualitative data [57].

We also made general observations on the participants behavior (e.g., how did they look for information online, how long they took to start writing the rule, etc.) throughout the whole experiment as to provide qualitative feedback on the process.

Finally, to complement our analysis and mitigate the complexity of analysing various sources of data of both quantitative and qualitative feedback, we did a triangulation of the results. For that, we compared the SUS questionnaire with the open-ended answers as well as the general observations made throughout the experiment. We discuss these on chapter 5.

4.1.2 Questionnaire

The SUS questionnaire was applied twice (one for each system) and followed a scale-based method, presenting an affirmative which the participant would choose from 1 to 5, i.e., *Strongly disagree* to *Strongly Agree*, respectively. The SUS questionnaire consists of the following questions:

(Q1) I think that I would like to use this system frequently.

(Q2) I found the system unnecessarily complex.

(Q3) I thought the system was easy to use.

(Q4) I think that I would need the support of a technical person to be able to use this system.

(Q5) I found the various functions in this system were well integrated.

(Q6) I thought there was too much inconsistency in this system.

(Q7) I would imagine that most people would learn to use this system very quickly.

(Q8) I found the system very cumbersome to use.

(Q9) I felt very confident using the system.

(Q10) I needed to learn a lot of things before I could get going with this system.

Other than the standard questions listed above, we also asked the participants the following open-ended questions regarding the experiment experience:

(OEQ1) How would you compare the two approaches for writing rules?

(OEQ2) What were the biggest difficulties found?

(OEQ3) What is your opinion regarding the flexibility on the rules creation?

4.1.3 Participants

For the experimental evaluation, eight male and one female volunteered for this experiment. They are students from the Federal University of Pernambuco. Table 2 shows the disposition of the participants. All of them were undergraduate students at the time and all of them were involved on IT courses. Five of them studied Computer Science while four of them studied Computer Engineering. All of them were between the third and tenth semester of their

Participant	Gender	Course	Semester
P1	Male	Computer Science	7th
P2	Male	Computer Engineering	6th
P3	Male	Computer Science	5th
P4	Male	Computer Engineering	10th
P5	Male	Computer Engineering	4th
P6	Male	Computer Science	5th
P7	Male	Computer Science	5th
P8	Female	Computer Science	3rd
P9	Male	Computer Engineering	5th

Table 2 Disposition of Participants

courses (for reference, at Federal University of Pernambuco the IT courses usually goes up to the ninth semester in normal circumstances, i.e., fulfilling all possible credits every semester). It is important to notice that although participants were all IT students, they did not have any prior knowledge of IDSs, effectively being a representative of a non-experienced user.

The participants were split into two groups according to their preferences of time: one group did the evaluation in a Saturday morning and the other a Saturday afternoon. The first group consisted of five students, and the second with the four remaining ones. Due to the complexity of the experimental setup with the number of devices, computers, and overall limited space, we originally chose to use up to 10 participants on this experiment but one of them could not make it due to external reasons.

4.2 ATTACKS

The two network attacks presented in this section and used throughout the experiment belongs to a category named distributed denial-of-service (DDoS) attacks. By definition, a denial-of-service (DoS) is characterized by a tentative of an attacker to prevent legitimate users from using a certain resource, i.e., the service [58]. A distributed denial-of-service (DDoS) occurs when the tentative comes from multiple sources, usually by unaware devices that were previously compromised and are actively used as a bot, therefore representing a *botnet* [58]. This is a common scenario when it comes to IoT as these devices are often constrained in terms of resources and security in general, therefore representing an easy target to be compromised [4].

The two attacks described next represent a DDoS Flood attack, largely used on IoT contexts and distinct from common DDoS attacks where the device is compromised to execute malicious code. A Flood attack has the single goal of flooding the victim with a large and

continuous volume of traffic, preventing the victim to provide services to legitimate users [32].

ICMP Flood Attack. The first network attack that we used to demonstrate the process of creating a rule was the Internet Control Message Protocol (ICMP) Flood, which is largely used as a DDoS attack. The ICMP protocol is used for sending messages that convey information about the network conditions, e.g., "TTL exceeded" or "need more fragments" [59]. An ICMP packet is not, by default, an "evil" packet and therefore should not be blocked. However, when the occurrences of these packets on the network increase to the point of exhausting a target server, this consists of a ICMP Flood attack.

This attack is often executed through botnets, i.e., a number of once unaffected internet-connected devices, each running a malicious software after being compromised. These botnets are largely present on the IoT context, especially in conjunction with the ICMP Flood attack [60]. For reproducing this attack, one can make use of a program to send multiple ICMP network packets (e.g., a *ping*) targeting a certain device that if not prepared, will have its resources exhausted while trying to answer these packets.

SYN Flood Attack. The second network attack was the SYN Flood and, similarly to the first attack, represents a common DDoS threat used in IoT contexts as a means to flood a certain server. We used this attack as an assignment to participants to evaluate the process of creating a rule by the participants of the experiment for both IDSs.

The "SYN" acronym stands for Synchronize flag used on TCP headers. This flag is turned on whenever a system sends a network packet to start a new TCP connection. First, a TCP packet is sent with the flag SYN activated to a destination device. Then, this device answers by sending an acknowledgment (ACK) packet, initiating the connection [61].

Being a core part of any network communication, a network packet with the SYN flag does not represent a threat by itself. However, if an attacker uses multiple SYN requests with a spoofed IP address to a victim, they will not be able to answer the packet to the source address as it does not exist. This results in rapid increase of half-open connections where the victim cannot answer with the corresponding ACK packet, to the point of exhausting the victim resources when it can no longer store the open connections. For reproducing this attack, one can use a program to send multiple SYN packets with different IP source addresses to a victim, which was exactly what we did on the experiment.

4.3 SETUP

The experimental setup consisted of the components seen in Figure 12 and 13 representing the Suricata and IoT-Flows setup, respectively.

- 1 Raspberry PI 3 running an OpenWRT software;

- 1 Raspberry PI 3 running the top-layer network protocol monitor from IoT-Flows system;
- 1 Notebook running a Windows system and acting as a victim IoT device;
- 1 Notebook running the *Analyzer* and *Pattern Mapper* components from IoT-Flows system;
- 1 Notebook running the Suricata system;
- 1 Notebook for each participant in which they could create the rules for the system in question.

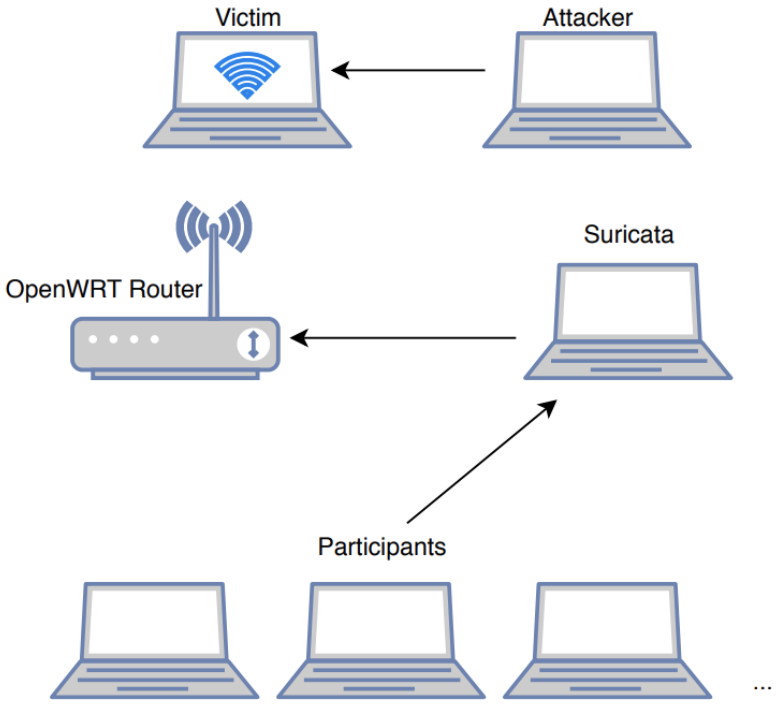


Figure 12 Suricata Setup

Network. For the experiment, the raspberry PI running the OpenWRT [1] software on Access Point mode was used to create a network so that all devices could connect to. The *pseudorouter* also enforced security with a pre-shared key (PSK) authentication via WPA2 (WPA2-PSK). All devices were linked to the router and connected on the same generated network.

Notebooks. Each participant was given a notebook connected to the OpenWRT router to create the rule. For Suricata, the participants could choose to write the rule directly on the command-line or use a text editor. We advised using Google Docs [2] for simplicity as after the

[1] https://openwrt.org/
[2] https://docs.google.com/

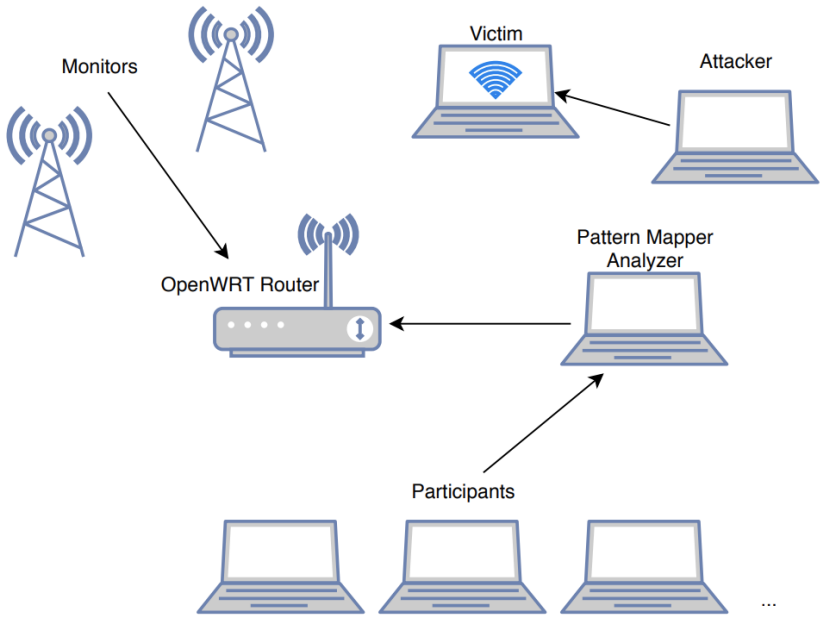


Figure 13 IoT-Flows Setup

rule were created, we would need to load them on Suricata(as seen in 3.1). For IoT-Flows, the participants used the notebook to navigate to the system's URLs for creating and seeing the rules. These were loaded automatically to the platform as described in 3.2.

Monitor. Although the IoT-Flows platform has two types of monitors, as described in 3.2.1, for the experiment only one of them was needed due to the nature of both types of attack, which needed information from the top network layers, i.e., Network, Transport and Application layer.

Target Device. For simplicity on validating the attack, we used a notebook acting as the victim device. The notebook was connected to the Internet and actively consuming a streaming video website, which made it easier to identify exactly when the device lost connection to the Internet, i.e., the video streaming stopped whenever the device was attacked.

4.4 ACTIVITY STEPS

Recall the goal of the study is to evaluate usability of an IDS focusing on the creation of the rules and the two platforms described on chapter 3, Suricata and IoT-Flows. To remove undesired variability and bias on the experiment due to previous experience on one platform or the other (depending which one were presented first), we used a experimental block design with a Latin square approach [54]. Figure 14 illustrates the group/platform accommodation for the experiment.

For each of the two groups of participants, the following tasks were assigned (see Ta-

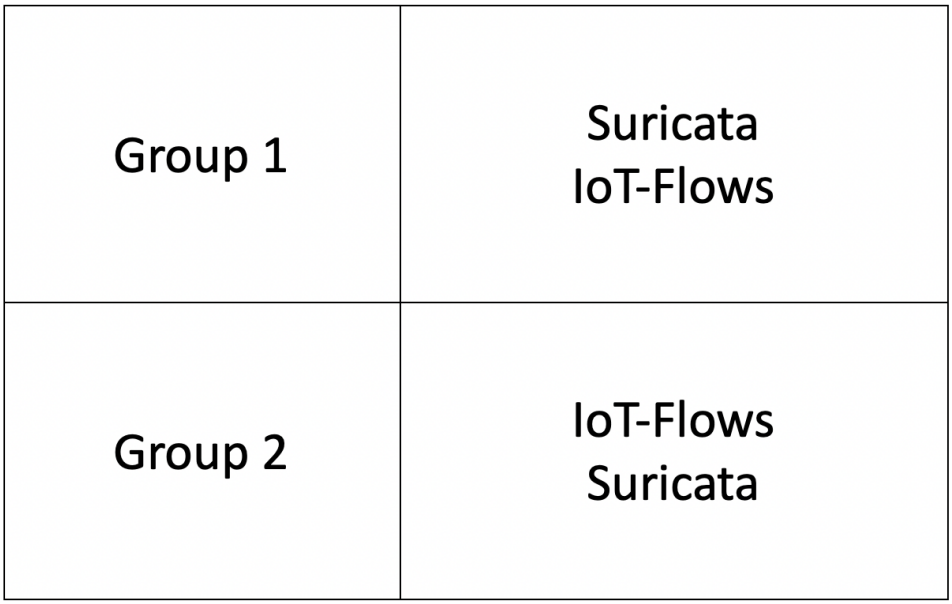


Figure 14 Experiment Latin Square Order of Platforms

Duration (in minutes)	**Description**
15	Overview of target platform focusing on creation of rules
5	ICMP Flood Attack demonstration
10	Presenting the SYN Flood Attack
30	1st round for creation of rules on target platform
10	Execution of the SYN Flood attack validating whether the created rules detected it or not
15	2nd round for creation of rules on target platform
10	Execution of the SYN Flood attack validating whether the created rules detected it or not
15	3nd and final round for creation of rules on target platform
10	Execution of the SYN Flood attack validating whether the created rules detected it or not

Table 3 Steps executed on the usability test.

ble 3 for the duration of each step):

1. The first of the two platforms were presented (Suricata in the first group, and IoT-Flows on the second). In this step, we gave an overview of the platform in question, emphasizing on how to create a rule for detecting the ICMP Flood Attack.

2. We executed the ICMP flood attack with the same rule created on the previous step.

3. We present the basics of the SYN Flood attack. Notice that no hint was given on how to create the rule itself for either platform.

4. We start the first round of 30 minutes, where the participants are asked to create a rule to detect the SYN Flood attack on the platform in question.

5. After the first and initial round, we gather the rules created by the participants and execute the SYN Flood attack, verifying whether any participant created a rule able to detect the attack.

6. For the participants that did not get the rule right, we gave two more rounds of 15 minutes each and repeated the previous step.

7. After these steps were completed for the first IDS, we repeated the process starting at Step 4.

Next, we describe the details of configuring each platform, first with Suricata, and then with IoT-Flows.

Configuration of Suricata. The Suricata configuration was straightforward. We compiled the software and removed all Suricata's default rules belonging to the Emerging Threat (ET) [62], a set of rules that cover several traditional network attacks. Instead, for the first part of the experiment in which we demonstrated the Suricata platform with the ICMP flood attack, we compiled Suricata with the rule responsible to detect the ICMP attack (seen at Listing 3.1). Then, for the second part of the experiment, we configured the rules created by each participant on the platform to detect the SYN Flood attack. For every time we executed the SYN Flood attack, we compiled all rules for the participants and observed the log from Suricata to check whether or not the rules detected the attack.

Configuration of IoT-Flows. Because the IoT-Flows platform has different components, these were configured separately and prior to the experiment. First, the the monitor was connected to the router (emulated by the raspberry PI) using the wireless 2.4GHz network created by the raspberry with the OpenWRT software. By doing that, all network packets traffic on the surrounding Wi-Fi networks were captured and sent to the IoT-Flows's *Analyzer* component. Both the *Analyzer* and the *Mapper* components from IoT-Flows were deployed to one of the notebooks, which served as a web server for both components. For the *Analyzer*, one could navigate to the URL (http:{notebook-ip:8080}/analyzer) to check the latest alerts generated by the component (see Figure 8). For the *Pattern Mapper*, the participants were instructed to navigate to the URL (http:{notebook-ip:5001}) so they could create the rule using the UI provided by this component. Figure 10 shows the list of the registered rules whereas Figure 11 shows the form to create a new rule.

As described in section 4.1, it is important to note that configuration had no relevance on evaluating the platforms, focusing solely on the process of creating the rule. In the following chapter, we discuss the results for the experiment analyzing both quantitative and qualitative feedback from the participants for each platform.

5 RESULTS AND DISCUSSION

In this section, we will present the results of the experiment and discuss them in detail. We start discussing the collected data with a summary of the general findings. Then, we present the results of the SUS questionnaire as well as the open-ended questions. Finally, we describe the observations made during the experiment and discuss the results, also presenting the threats to validity of the experiment and how we mitigated them.

5.1 GENERAL

Regarding the Suricata system, only one participant wrote a rule able to detect the SYN flood attack whereas on the IoT-Flows platform, seven out of nine participants got the rule right.

It is important to note that we did not focus our evaluation on whether or not the participant could write the correct rule, but the process of creating the aforementioned rule itself. However, we did take note of the number of rounds each participant took to write the correct rule (recall that we executed the attack with the participant's rules three times in total). For the Suricata system, the participant who wrote the correct rule did so after three rounds. On the IoT-Flows out of the seven participants who got it right, five used two rounds and two used all three rounds.

During the experiment, we also provided answers to questions the users had, focusing specifically on the creation of the rules for each system. Tables 8 and 9 shows how many times each participant asked for help. For the IoT-Flows system, three users requested help once, four users requested twice, and two users requested help three times in total. On the Suricata system, one user requested no help, three users requested help once, two asked twice, two asked questions three times and, finally, one requested help four times.

Regarding the nature of the questions, they were more focused on the syntax for the creation of the rule and the SYN flood attack, and less on the system itself. We elaborate on this on subsection 5.1.2.

It is worth noting that, during the initial presentation of the system, the participants showed a great interest on the topic overall, discussing possible scenarios in which the system could be used and how.

5.1.1 System Usability Scale (SUS)

Interpreting scoring on the SUS questionnaire can be complex. Due to the nature of the scale and the number of questions, one can be tempted to interpret them as percentages, which they are not. A standardized way involves normalizing the resulting answers by adding them

Group	Suricata	IoT-Flows
Group 1	50	65
Group 2	33,75	73,75

Table 4 SUS Average Score

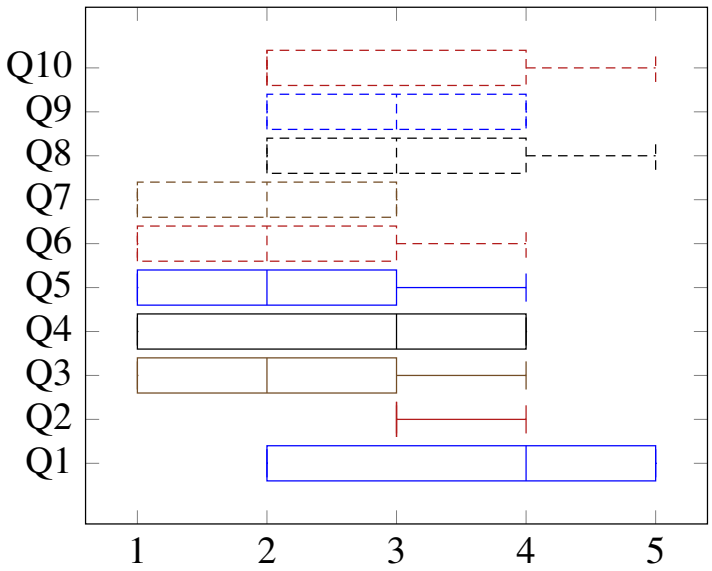


Figure 15 Group 1 - SUS Boxplot on Suricata

together in the following way: (1) for each of the odd numbered questions, subtract 1 from the score; (2) for each of the even numbered questions, subtract their value from 5; and (3) take the new values and add up to the total score, then multiply this by 2.5. Albeit not a percentage, this method gives an approximate percentile ranking. A SUS score above 68 can be considered *average*, with anything below it being *below average* [56].

Table 4 shows the SUS Average score for each platform. Considering the Suricata system, the participants on Group 1, which had (Suricata presented first) rated an average SUS score of **50**, which shows the system as *slightly below average*. Group 2, which had Suricata presented last, gave an average SUS score of **33,75**, indicating the system as *below average*.

For the IoT-Flows platform, the participants on Group 1, which had IoT-Flows presented last, rated the system with an average SUS score of **65**, which shows the system as slightly below average. Group 2, which had IoT-Flows presented first, gave an average SUS score of **73,75**, indicating the system as *above average*. Figures 15 to 20 shows the distribution of the answers for both groups. Notice the questions are labeled (Q1) to (Q10) for space limit reasons.

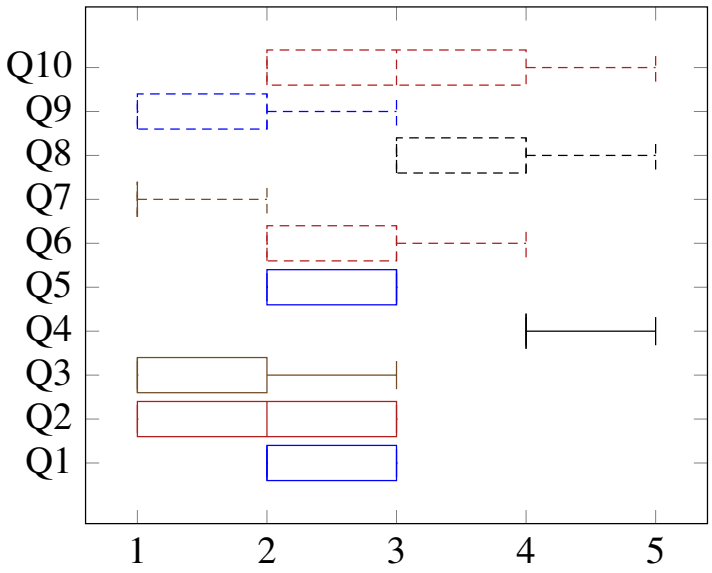


Figure 16 Group 2 - SUS Boxplot on Suricata

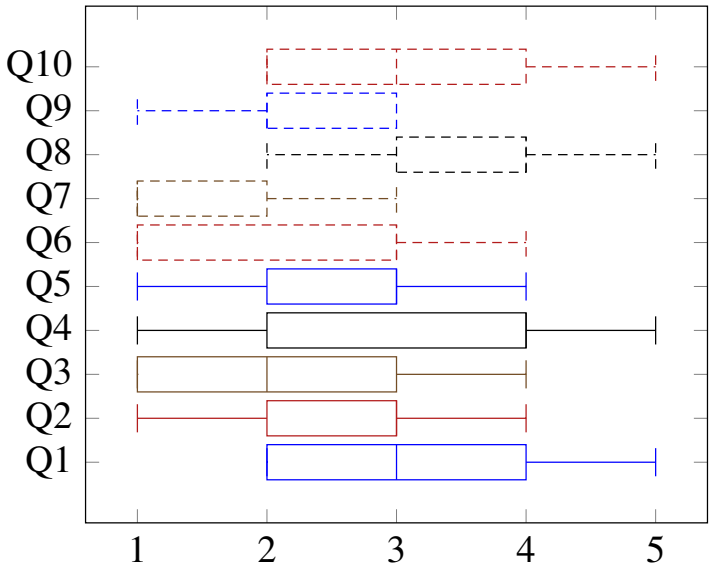


Figure 17 Group 1 and 2 - SUS Boxplot on Suricata

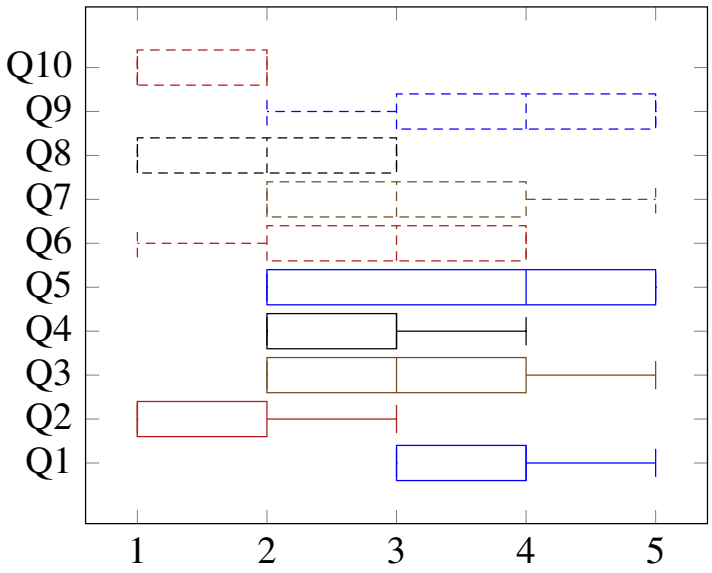


Figure 18 Group 1 - SUS Boxplot on IoT Flows

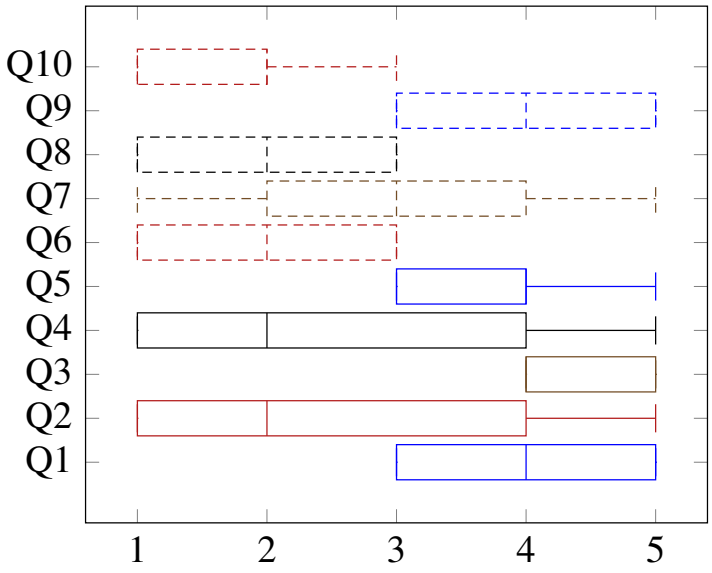


Figure 19 Group 2 - SUS Boxplot on IoT Flows

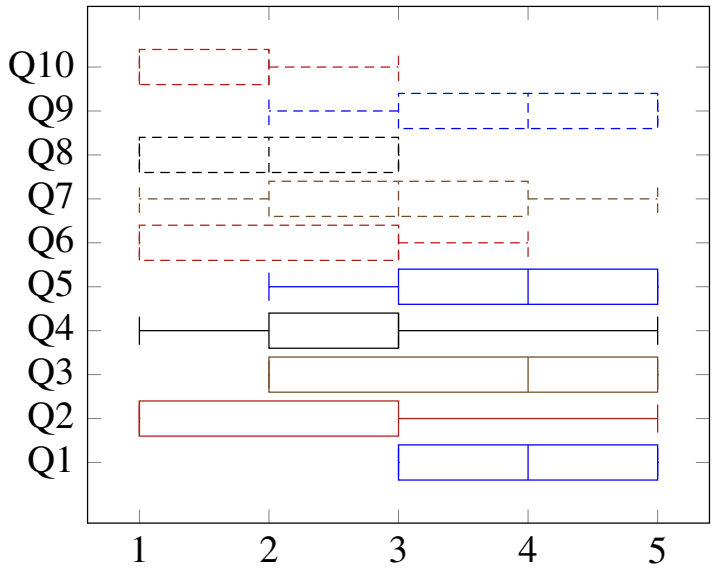


Figure 20 Group 1 and 2 - SUS Boxplot on IoT Flows

	Syntax	**Learning**	**Documentation**	**UI**	**Speed**
Suricata	"conventional way ofwiring rules" "command-line approach" "overcomplicated syntax"	"difficult for first-time users" "less intuitive" "confuse for first-time users"	"complex" "confuse" "lots of parameters"	-	"conventional way of writing rules" "faster to long-time users" "faster not considering finding parameters"
IoT-Flows	"more intuitive" "easier to write the rules"	"easier for first-time users" "parameters all presenton UI" "more explicit"	-	"UI makes it easier to write rules" "More intuitive due to its UI" "Easier to use due to its UI"	"slower but more intuitive" "slower to combine rules"

Table 5 Key phrases found on the coding process for the answers of Open Ended Question(OEQ1)

5.1.2 Open-ended questions

To better understand the process of creating the rules for the Suricata and IoT-Flows systems, we asked the participants open-ended questions not related to the SUS scale. We also Codified the answers (see section 4.1 for details) to better understand the overall qualitative feedback on writing the rule on both platforms. Tables 5, 6 and 7 summarizes the key words/phrases found during the coding process of the answers which lead to the clustered topics described below.

(OEQ1) How would you compare the two approaches for writing rules?

For this question, we evaluated the answers and coded the following topics: **Syntax**, **Learning**, **Documentation**, **User-interface (UI)**, and **Speed**.

Syntax. The consensus among the participants was that although Suricata presents a conventional way of creating the rules with its command-line approach, its syntax is over-complicated and presents several parameters that are rarely used and are difficult to find on the documentation. Regarding IoT-Flows, the participants highlighted the fact that the system provides a more intuitive way to create the rules

with its UI, allowing the users to select among a predetermined set of parameters, however still presents some problems relative to understanding the syntax due to the project immaturity. Participant 2 wrote: *"IoT-Flows, albeit a promising system, stil have some problems relative to the understanding of the syntax and semantics that can be mitigated to improve user's experience."*.

Learning. The participants pointed that Suricata, although presenting a thorough documentation, was difficult for first-time users due to its complexity. On IoT-Flows, the consensus was that the system provided a more intuitive way to learn the rules due to fact that the parameters were all presented in the UI, therefore being more "explicit to the users". Participant 4 wrote: *"The approach for creating rules on Suricata is simple in terms of code, but hard for first-time users as the documentation is too vast and make it difficult understanding the parameters to be used for detecting the attacks"*.

Documentation. The consensus among the participants was that Suricata presented a vast documentation but made it hard to find the multiple parameters one can use to create a rule. Participant 8 wrote: *"Suricata has a very complex documentation, which made it difficult to write even a simple rule"*. The IoT-Flows system did not provide documentation for consulting and as such, did not qualified for this topic.

User-interface (UI). For Suricata, the participants pointed the fact that creating a rule was straightforward but confusing for first-time users as everything was made through a command-line terminal or text editor. On the IoT-Flows, the users highlighted its user-interface as the strongest point of the system, presenting an easier way for the creation of rules. Participant 9 wrote the following: *"The IoT-Flows system presented a more friendly user interface, making it easy for creating the rules on the network. However, Suricata seemed to have more variety for creating and combining the rules."*

Speed. The participants cited Suricataas faster than IoT-Flows in terms of creation of the rules, highlighting, however, the trade-off between speed and complexity, specially for first-time users of this system. Participant 6 wrote: *"The approach of the IoT-Flows platform, albeit slower, seemed to be more intuitive and different from Suricata, which was a little harder for producing the correct commands but faster to write the rules."*

(OEQ2) What were the biggest difficulties found?

For this question, we coded the following topics: **Documentation**, **Parameters**, and

	Documentation	Parameters	Rules
Suricata	"confuse documentation" "complex"	"too many parameters" "difficult to find"	"understanding the syntax to build a rule"
IoT-Flows	"no documentation available"	"hard to combine parameters" "using the correct parameters"	"combining the rules"

Table 6 Key phrases found on the coding process for the answers of Open Ended Question(OEQ2)

	Rules	Parameterization	UI
Suricata	"consistency" "straightforward to build rule"	"lots of parameters to choose" "more options"	"no UI available"
IoT-Flows	"more options to build rules"	"limited number of parameters"	"possible parameters explicit to users" "greater flexibility due to its UI"

Table 7 Key phrases found on the coding process for the answers of Open Ended Question(OEQ3)

Rules.

Documentation. Looking at the answers showed a general feeling from the participants that although Suricata provided a complete documentation, it proved too confuse/complex for non-experienced users. Participant 5 answered: *"One of the biggest difficulties was the time to understand the documentation"*. Participants also highlighted the fact that IoT-Flows had no documentation available.

Parameters. The consensus among the participants was that finding parameters on Suricata's documentation proved to be a difficult task. On IoT-Flows, the participants noted that although the parameters were made clear with the UI, they were not easy to combine with the goal of building a rule. Participant 3 wrote: *"One difficult that I experienced was understanding how to combine the parameters to build a rule as there was no documentation available"*.

Rules. Understanding the syntax used to build a rule for Suricata was hard, according to the participants. Participant 8 noted: *"Understanding the syntax used and how the rules work was hard"*. On IoT-Flows, consensus was that combining the "child rules" (a characteristic of IoT-Flows only) was not easy due to the fact that no documentation was available.

(OEQ3) What is your opinion regarding the flexibility on the rules creation?

For this question, the coding process returned the following topics: **Rules**, **Parameterization**, and **UI**.

Rules. While using Suricata, the participants highlighted the consistency while creating the rules, with the system offering a great deal of customization and number of parameters to choose from. Participant 2 wrote: *"The process of creating rules on Suricata was consistent offering good tools to customize the rules"*. On IoT-Flows,

consensus was that the system had more options to build rules if considering the different combinations one could make with the parameters.

Parameterization. The participants noted that Suricata presented a great deal of parameters to choose from whereas IoT-Flows showed a limited number. However, more parameters did not make it easier to write the rule itself. Participant 3 wrote: *"I find it important to the system to have a great number of parameters as a means to provide flexibility. The price to pay is how easy it is to use this system"*.

UI. The consensus among the participants was that the absense of a UI on Suricata made it harder to write the rules if compared to IoT-Flows. On IoT-Flows, the participants also noted that the available UI enabled the process of creation of rules to be more intuitive in general. Participant 4 wrote: *"With practice, it was much more easier to understand the interface and create the rules"*.

All in all, we noted that the participants which had Suricata introduced first highlighted the UI of IoT-Flows as a strong point in terms of flexibility and consistency for creating the rules. This is understandable as when using the IoT-Flows, the users already had some experience in terms of syntax for creating the provided rule but had to do it through a command-line or text-editor interface. However, this did not hold true for the second group where the participants pointed the high number of parameters and documentation of Suricata as the biggest difference with IoT-Flows, leading to a more flexible process while creating the rules.

5.2 OBSERVATIONS

Recall that the goal of the experiment was not writing the correct rule, but evaluating the process of creating the rule for each system. During the experiment, we made notes about the different questions asked by the participants and whether they wrote the correct rule by the end of the interaction with the system in hand. This provided qualitative feedback for each tool in an indirect manner, indicating the difficulties found and overall learning experience which were not covered by the questionnaire. We split the different answers asked into the two groups of participants.

Group 1. Out of the five participants introduced to Suricata first, only one wrote the correct rule, needing three *rounds* to do so [1]. Questions about Suricata were focused on the syntax and the nature of the proposed SYN Flood attack. The participants pointed that finding the necessary parameters on the documentation as to enable them to create the rule for the attack was a difficult task.

[1] *round* here means the time window we gave the participants for writing the rule before executing the attack)

Participants	Suricata			IoT-Flows		
	Wrote the correct rule?	# Rounds to write correct rule	# Times that asked for help	Wrote the correct rule?	# Rounds to write correct rule	# Times that asked for help
Participant 1	NO	N/A	1	YES	3	1
Participant 2	NO	N/A	2	NO	N/A	1
Participant 3	NO	N/A	0	NO	N/A	1
Participant 4	NO	N/A	1	YES	2	2
Participant 5	YES	3	1	YES	2	2

Table 8 Group 1 Experiment Data

Participants	Suricata			IoT-Flows		
	Wrote the correct rule?	# Rounds to write correct rule	# Times that asked for help	Wrote the correct rule?	# Rounds to write correct rule	# Times that asked for help
Participant 6	NO	N/A	3	YES	2	2
Participant 7	NO	N/A	2	YES	2	2
Participant 8	NO	N/A	3	YES	2	3
Participant 9	NO	N/A	4	YES	3	3

Table 9 Group 2 Experiment Data

For the IoT-Flows system, out of five participants, three wrote the correct rule, one needing three rounds and the other two rounds to do so. The questions revolved around the possible parameters for creating the rule, specifically the network packet fields that were valid. The participants also asked details of the SYN Flood attack as a means to identify the parameters to be used and create the rule.

Group 2. For this group, the participants were introduced the IoT-Flows system first. All of the participants wrote the correct rule, three of them after two rounds, and one of them after three rounds. The questions revolved around syntax and the rules structure on the system, e.g., whether one could create a rule and link to other, both targeting detection of the same attack.

None of the participants wrote the correct rule on the Suricata system. The difficulties encountered by the participants revolved mostly around the compilation of the rules, citing the complex syntax and hard-to-read documentation as the causes.

5.3 DISCUSSION

In this discussion we confront the data from the SUS questionnaire data against the answer of the open-ended and our observations. Considering the order which the participants were exposed to the platforms on the experiment (as seen in Figure 14), we can analyze the difference and similarities of the SUS answers and relate them to the open-ended questions.

For instance, consider the SUS Q4 (*"I think that I would need the support of a technical*

person to be able to use this system."). Answers from Group 1 for Suricata ranged from 1 to 4 whereas Group 2, which had IoT-Flows presented first, had no answer rated less than 4. This is in line with the (OEQ3), which had Group 2 consensus on the opinion that IoT-Flows presented greater flexibility due to its UI comparing to Suricata when being presented to the system after IoT-Flows.

Considering SUS Q7 (*"I would imagine that most people would learn to use this system very quickly"*), answers from Group 1 ranged from 1 to 3 whereas Group 2 had three out of four participants rating 1 for Suricata while varying from 1 to 5 on IoT-Flows. Looking at the answers for (OEQ2), this is in line with the opinion of participants on Suricata presenting a detailed but hard to read documentation whereas IoT-Flows had no documentation at the moment of the experiment.

Considering SUS Q8 (*"I found the system very cumbersome to use."*), both Group 1 and Group 2 rated Suricata with an average of 4, which is in line with the answers of open-ended (OEQ1) that presented a consensus that the process of creating the rule with a command-line or text editor was not intuitive. This drastically contrasts with IoT-Flows, which had answers rating from 1 to 3 on both groups. This is also compatible to the pointed fact from the participants that IoT-Flows's UI made the process of creating the rules easier than Suricata.

Considering SUS 10 (*"I needed to learn a lot of things before I could get going with this system."*), both Group 1 and Group 2 rated Suricata from 2 to 5 while rating IoT-Flows 1 to 3. This corroborates to (OEQ1) answers which highlighted the difficult in finding the correct parameters to use on Suricata while pointing that, although IoT-Flows had no documentation, presented itself with greater learnability as the parameters were all presented on the UI.

5.4 THREATS TO VALIDITY

In this section, we discuss the limitations of our study, specifically the experimental evaluation process and how we handled them. We describe the external, internal, and construct threats to the validity of our results.

5.4.1 External Validity

External validity concerns the representativeness of our results to the population observed, in our case, the participants of the experiment. Our findings were limited to the scope of the experiment, which as described in chapter 4, had a relative small population of nine participants. This is in line with the nature of the experiment and complexity of the setup where multiple devices were involved, therefore limiting the physical space for the experiment and how the resources were distributed among the participants.

Because the participants were all IT students, one can question whether or not they

represent the target user of the proposed platform, i.e., a non-experienced user (consumer). Although the participants were experienced on the IT domain, they did not have any prior experience with IDS, which led them to perform similarly to a non-experienced user. The work of Salman et al. [63] provides evidence that students and practitioners tend to perform similarly in software engineering experiments that evaluate development approaches where participants do not have prior experience. Thus, minimizing threats to validity under that perspective.

5.4.2 Internal Validity

Internal validity concerns the consistency of our measurements during and after the experiment. External factors may affect the causality of the observed results during our experiments. The results could be influenced by a possible bias towards the platform presented on this work. To mitigate this, we explicitly let the participants know that feedback from both platforms used on the experiment were crucial to the future of the project, i.e., if the platform proposed in this work were to be poorly evaluated during the experiment, it could be dropped in favor of the other without concerns. Judging by the feedback for both platforms, this advice was fruitful as the participants did not show any bias towards one system or the other.

5.4.3 Construct Validity

In this work, we considered a number of metrics that could influence some of our interpretations towards them. Specifically, the open-ended questions were subjective and open to our own interpretation, which could lead to bias towards the proposed platform. This is demonstrated by the possible limitation on the Coding process explained on 4.1, as this process was done only by one person.

6 RELATED WORK

Multiple studies discussed how vulnerable IoT devices are to traditional and new threats in the recent years, especially on Smart Home contexts [14; 64]. Yoon et al. focused on the Smart Home IoT scenario, pointing that although this context provides a more convenient life as supplementary services are provided through day-to-day IoT devices, the more home devices linked, the more security flaws are revealed [64]. Still focusing on the Smart Home context, alarming threats involving IoT devices also made the news with baby monitors and smart locks proving to be vulnerable to both local and remote attacks [4; 65]. This work proposes a platform that can mitigate these threats, as even non-IT users are enabled to create rules to detect suspicious behavior involving the Smart Home devices with an intuitive user-interface, contrary to standard solutions that proved to be complex as they often target network admins and experts.

Khaled Salah presented a comprehensive review of IoT security with multiple open-challenges and future research directions [12]. Among them, limitations of resources on IoT devices and interoperability of protocols used stand out, with the former representing a core feature of IoT devices (i.e., being limited in processing, memory, etc.) and the latter presenting a challenge as these devices, being network devices, encompass multiple network layers. This work proposes a platform that mitigates those challenges as detection of traditional and new threats occurs independently of the IoT devices. The platform covers all network layers by **monitoring the devices**, **analysing the network traffic**, and **enforcing intrusion detection rules** while enabling rules to be created intuitively without deep knowledge of network features, characteristics and protocols.

Due to the rapid growth of the IoT market, a number of solutions addressing security concerns on IoT networks emerged on the recent past. Considering Intrusion Detection Systems (IDS)s, specifically Signature-based ones that make use of pattern matching to detect threats, few of them have its focus on IoT contexts. SVELTE is one such Intrusion Detection Systems (IDS) which focus on targeting specific routing attacks such as spoofed or altered information, sinkhole, and selective-forwarding while presenting small overhead, a crucial feature when it involves constrained networks such as IoT environments [27]. IoT-IDM presented a Host-Based Intrusion Detection Systems (IDS) focusing specifically on Smart Home IoT [66]. On that work, Nobakht et al., used a software-defined networking techonlogy and its protocol, OpenFlow, to detect intrusions with customized machine learning techniques and learned signature patterns of known threats. Similarly to these, the proposed platform on this work envisions addressing network IoT threats by providing a flexible way to create rules to detect attacks on all network layers.

Werlinger et al., analyzed the challenges of Intrusion Detection Systems (IDS) by interviewing 9 IT Security practitioners who have worked with IDSs and performed participatory observations in an organization deploying a network IDS [7]. The work showed that other than the expected difficulties involving configuration of these systems, usability was a crucial challenge when using the system as the rules enforced by Intrusion Detection Systems (IDS)s are mostly non-intuitive and difficult to understand. [7]. In this work, we proposed a platform that focus on tackling this problem by enabling users to create Intrusion Detection Systems (IDS) rules in an intuitive way while also enforcing these rules with a real time monitoring and analysis, similar to traditional Intrusion Detection Systems (IDS)s.

Similarly to this work, a number of studies also involved evaluation of Intrusion Detection Systems (IDS)s in general. However, the evaluation of these platforms focused on variables such as performance, memory consumption and number of false positives/negatives by replicating network traffic while running the systems. For instance, Eugene Albin compared two open-source IDSs, Suricata and the system that originated it, Snort [44]. Albin pointed that Suricata required more memory and CPU resources than Snort due to its multi-thread architecture. However, Snort's need for multiple instances running to accomplish what Suricata does counter this factor. In terms of false positive/negatives, the study was inconclusive on which of the IDSs has a better detection algorithm.

Jun Chen et al. proposed an Intrusion Detection Systems (IDS) architecture that used Complex Event Processing (CEP) technology to detect threats on IoT networks [67]. The authors demonstrated that CEP mechanisms are more suitable for applications and services which needs to process streaming originating from underlying sensors and devices in IoT environments. Similarly, we proposed a platform that uses CEP analysis to detect threats on IoT environments, but focusing on two points that are crucial to these systems, extensibility and usability with the creation and enforcing of user-created rules.

All in all, although these studies explored the security concerns on IoT in general, they often focused on proposing solutions that show a clear trade-off between robustness and usability of the proposed systems. This work proposed a platform that focus on usability (among other things) enabling users to create rules for detecting both traditional and new threats on IoT networks. By comparing the proposed platform with Suricata, this work also corroborated with previous studies [6; 7] demonstrating that usability is a crucial point of any security system targeting IoT devices.

7 CONCLUSIONS

Securing IoT devices it not an easy task, but crucial due to the rapid growth of the IoT market in the recent past. For that, Network Intrusion Detection Systems (NIDS) can be used to employ defenses on IoT networks. However, due to the nature of this market, with its large share being represented by consumers on Smart Home contexts, usability is one of the core features of these systems but is often neglected, as discussed on previous studies [6; 7]. Previous studies also demonstrated that these systems often act on a single network layer and provides no easy means to extended them [41]. To mitigate these limitations, in this work we proposed IoT-Flows: a platform that stands as an Intrusion Detection Systems (IDS) that treats usability as a core feature. The proposed system enables users to create rules on the platform with a user-interface (UI). IoT-Flows also acts on all network layers and provides extensibility enabling users to create rules for detecting both traditional and new threats in real-time.

To evaluate the proposed platform focusing specifically on the creation of rules, we conducted a usability test comparing the platform with the most popular open-source IDS, Suricata. The usability test consisted on splitting nine student volunteers into two groups following a Latin-square approach (as a means to eliminate bias) and later assigning them the task of creating a rule to detect a popular DDoS attack for each system. We then asked the participants to answer a SUS questionnaire and three open-ended questions that provided qualitative feedback on the process of creating the rules. We also made general observations throughout the test and later triangulated the results of both qualitative and quantitative feedback. The consensus among the participants was that Suricata, albeit providing a complete documentation, lacks flexibility for creating the rules due to its complex syntax and non-existing user-interface (UI), being a negative point specially for new users. For the proposed system, IoT-Flows, the participants highlighted its UI and flexibility as its strongest points, providing a intuitive way of creating the rules. However, they also noted that creating the rules was slower if compared to Suricata, which shows a trade-off between complexity and easy-of-use on these systems. To sum, during this work, we observed that usability is indeed a crucial point that needs to be taken into consideration when developing security systems, especially if the systems focus on IoT contexts, where the presence of non-IT users is a common thing.

For future work, we envision usage of machine-learning techniques applied to the historical network data coming from the IoT devices with the goal of finding new threats automatically. This is in-line with the concept of a self-managing platform, which could employ MAPE-K's *Knowledge* component for this. We also plan to expand the capability of monitoring network data from a distributed perspective using only one type of monitor instead of two

and expand the *Execute* components of the platform (e.g., enabling users to configure email and SMS alerts to be generated), which will provide even more flexibility for the platform and its users. Finally, we look forward to evaluate the system against other traditional IDSs by exploring common variables in such comparisons such as number of false positive/negatives, performance and resources consumption.

REFERENCES

[1] M. Nawir, A. Amir, N. Yaakob, and O. B. Lynn, "Internet of things (iot): Taxonomy of security attacks," in *Electronic Design (ICED), 2016 3rd International Conference on*. IEEE, 2016, pp. 321–326.

[2] I. Lee and K. Lee, "The internet of things (iot): Applications, investments, and challenges for enterprises," *Business Horizons*, vol. 58, no. 4, pp. 431 – 440, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0007681315000373

[3] Business Insider, "There will be 24 billion iot devices installed on earth by 2020," 2016. [Online]. Available: https://www.businessinsider.com/there-will-be-34-billion-iot-devices-installed-on-earth-by-2020-2016-5

[4] Michigan News, "Hacking into homes: 'smart home' security flaws found in popular system," 2016. [Online]. Available: https://news.umich.edu/hacking-into-homes-smart-home-security-flaws-found-in-popular-system/

[5] Forbes, "Cyberattacks on iot devices surge 300% in 2019, 'measured in billions', report claims," 2019. [Online]. Available: https://www.forbes.com/sites/zakdoffman/2019/09/14/dangerous-cyberattacks-on-iot-devices-up-300-in-2019-now-rampant-report-claims/#71e1b7a65892

[6] D. S. Butt and V. A. Gnevasheva, "Efficiency in the processes of intrusion detection system through usability evaluation methods," *Available at SSRN 3151216*, 2018.

[7] R. Werlinger, K. Hawkey, K. Muldner, P. Jaferian, and K. Beznosov, "The challenges of using an intrusion detection system: Is it worth the effort?" in *Proceedings of the 4th Symposium on Usable Privacy and Security*, ser. SOUPS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 107–118. [Online]. Available: https://doi.org/10.1145/1408664.1408679

[8] T. Sherasiya, H. Upadhyay, and H. B. Patel, "A survey: Intrusion detection system for internet of things," *International Journal of Computer Science and Engineering (IJCSE)*, vol. 5, no. 2, 2016.

[9] Forbes, "2017 roundup of internet of things forecasts," 2005. [Online]. Available: https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts/#7b71aae11480

[10] InfoSec Institute, "Open source ids: Snort or suricata?" 2019. [Online]. Available: https://resources.infosecinstitute.com/open-source-ids-snort-suricata/#gref

[11] OISF, "Suricata open source ids." [Online]. Available: https://suricata-ids.org/

[12] M. A. Khan and K. Salah, "Iot security: Review, blockchain solutions, and open challenges," *Future Generation Computer Systems*, vol. 82, pp. 395 – 411, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X17315765

[13] Z. B. Celik, L. Babun, A. K. Sikder, H. Aksu, G. Tan, P. McDaniel, and A. S. Uluagac, "Sensitive information tracking in commodity iot," *arXiv preprint arXiv:1802.08307*, 2018.

[14] TrendMicro, "Inside the smart home: Iot device threats and attack scenarios," 2018. [Online]. Available: https://www.trendmicro.com/vinfo/us/security/news/internet-of-things/inside-the-smart-home-iot-device-threats-and-attack-scenarios

[15] The Guardian, "Ddos attack that disrupted internet was largest of its kind in history, experts say," 2018. [Online]. Available: https://www.theguardian.com/technology/2016/oct/26/ddos-attack-dyn-mirai-botnet

[16] S. Samonas and D. Coss, "The cia strikes back: Redefining confidentiality, integrity and availability in security." *Journal of Information System Security*, vol. 10, no. 3, 2014.

[17] D. E. Bell and L. J. LaPadula, "Computer security model: Unified exposition and multics interpretation," *MITRE Corp., Bedford, MA, Tech. Rep. ESD-TR-75-306, June*, 1975.

[18] R. S. Sandhu, "On five definitions of data integrity." in *DBSec*. Citeseer, 1993, pp. 257–267.

[19] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on software engineering*, no. 2, pp. 222–232, 1987.

[20] A. Tayebi, S. Berber, and A. Swain, "Wireless sensor network attacks: An overview and critical analysis," in *Sensing Technology (ICST), 2013 Seventh International Conference on*. IEEE, 2013, pp. 97–102.

[21] D. R. Raymond and S. F. Midkiff, "Denial-of-service in wireless sensor networks: Attacks and defenses," *IEEE Pervasive Computing*, no. 1, pp. 74–81, 2008.

[22] D. M. Junior, W. Rodrigues, K. Gama, J. A. Suruagy, and P. A. da S. Gonçalves, "Towards a multilayer strategy against attacks on iot environments," in *Proceedings of the 1st International Workshop on Software Engineering Research & Practices for the Internet of Things, SERP4IoT@ICSE 2019, Montreal, QC, Canada, May 27, 2019*, 2019, pp. 17–20. [Online]. Available: https://doi.org/10.1109/SERP4IoT.2019.00010

[23] N. Namvar, W. Saad, N. Bahadori, and B. Kelley, "Jamming in the internet of things: A game-theoretic perspective," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–6.

[24] NEC Corporation, "Lightweight-architecture tamper detection technology to protect iot devices," 2018. [Online]. Available: https://www.nec.com/en/global/rd/technologies/falsification_find/index.html

[25] P. Ruckebusch, E. De Poorter, C. Fortuna, and I. Moerman, "Gitar: Generic extension for internet-of-things architectures enabling dynamic updates of network and application modules," *Ad Hoc Networks*, vol. 36, pp. 127–151, 2016.

[26] D. Nabil, D. Tandjaoui, I. Romdhani, and F. Medjek, "Trust-based defence model against mac unfairness attacks for iot," 07 2017.

[27] S. Raza, L. Wallgren, and T. Voigt, "Svelte: Real-time intrusion detection in the internet of things," *Ad hoc networks*, vol. 11, no. 8, pp. 2661–2674, 2013.

[28] H. Deng, X. Sun, B. Wang, and Y. Cao, "Selective forwarding attack detection using watermark in wsns," in *Computing, Communication, Control, and Management, 2009. CCCM 2009. ISECS International Colloquium on*, vol. 3. IEEE, 2009, pp. 109–113.

[29] S. Abbas, M. Merabti, D. Llewellyn-Jones, and K. Kifayat, "Lightweight sybil attack detection in manets," *IEEE systems journal*, vol. 7, no. 2, pp. 236–248, 2013.

[30] P. Pongle and G. Chavan, "Real time intrusion and wormhole attack detection in internet of things," *International Journal of Computer Applications*, vol. 121, no. 9, 2015.

[31] V. P. Singh, S. Jain, and J. Singhai, "Hello flood attack and its countermeasures in wireless sensor networks," *International Journal of Computer Science Issues (IJCSI)*, vol. 7, no. 3, p. 23, 2010.

[32] N. Dao, T. V. Phan, U. Sa'ad, J. Kim, T. Bauschert, and S. Cho, "Securing heterogeneous iot with intelligent ddos attack behavior learning," *CoRR*, vol. abs/1711.06041, 2017. [Online]. Available: http://arxiv.org/abs/1711.06041

[33] K. Fan, W. Jiang, H. Li, and Y. Yang, "Lightweight rfid protocol for medical privacy protection in iot," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, pp. 1656–1665, April 2018.

[34] S. Demetriou, N. Zhang, Y. Lee, X. Wang, C. A. Gunter, X. Zhou, and M. Grace, "Hanguard: Sdn-driven protection of smart home wifi devices from malicious mobile apps," in *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2017, pp. 122–133.

[35] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash, "Flowfence: Practical data protection for emerging iot application frameworks." in *USENIX Security Symposium*, 2016, pp. 531–548.

[36] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan, "A survey of intrusion detection techniques in cloud," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 42 – 57, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804512001178

[37] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16 – 24, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804512001944

[38] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou, "Specification-based anomaly detection: A new approach for detecting network intrusions," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, ser. CCS '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 265–274. [Online]. Available: https://doi.org/10.1145/586110.586146

[39] International Computer Science Institute-Berkley CA, "The zeek network security monitor," 2019. [Online]. Available: https://www.zeek.org/

[40] Sourcefire, “Snort,” 1998. [Online]. Available: https://www.snort.org/

[41] SANS Institute, “Network intrusion detection - keeping up with increasing information volume,” 2019. [Online]. Available: https://www.sans.org/reading-room/whitepapers/detection/network-intrusion-detection-keeping-increasing-information-volume-347

[42] I. Butun, S. D. Morgera, and R. Sankar, “A survey of intrusion detection systems in wireless sensor networks,” *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 266–282, First 2014.

[43] F. Sabahi and A. Movaghar, “Intrusion detection: A survey,” in *2008 Third International Conference on Systems and Networks Communications*, Oct 2008, pp. 23–26.

[44] E. Albin and N. C. Rowe, “A realistic experimental comparison of the suricata and snort intrusion-detection systems,” in *2012 26th International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 2012, pp. 122–127.

[45] OISF, “Suricata rules.” [Online]. Available: https://suricata.readthedocs.io/en/suricata-4.1.5/rules/intro.html

[46] CIn-UFPE, “Iot-flows,” 2019. [Online]. Available: https://iot-flows.cin.ufpe.br/

[47] IBM, “An architectural blueprint for autonomic computing,,” 2005. [Online]. Available: https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf

[48] Q. M. Ashraf and M. H. Habaebi, “Autonomic schemes for threat mitigation in internet of things,” *Journal of Network and Computer Applications*, vol. 49, pp. 112–127, 2015.

[49] R. J. Mitchell, *Managing complexity in software engineering*. IET, 1990, no. 17.

[50] EsperTech, “Esper,” 2019. [Online]. Available: http://www.espertech.com/esper/

[51] E. Biermann, E. Cloete, and L. M. Venter, “A comparison of intrusion detection systems,” *Computers & Security*, vol. 20, no. 8, pp. 676–683, 2001.

[52] M. K. Rafsanjani, A. Movaghar, and F. Koroupi, “Investigating intrusion detection systems in manet and comparing idss for detecting misbehaving nodes,” *World Academy of Science, Engineering and Technology*, vol. 44, pp. 351–355, 2008.

[53] Cloudfare, “Syn flood ddos attack.” [Online]. Available: https://www.cloudflare.com/learning/ddos/syn-flood-ddos-attack/

[54] N. Juristo and A. M. Moreno, *Basics of software engineering experimentation*. Springer, 2011.

[55] V. J. Caracelli and J. C. Greene, “Data analysis strategies for mixed-method evaluation designs,” *Educational Evaluation and Policy Analysis*, vol. 15, no. 2, pp. 195–207, 1993. [Online]. Available: https://doi.org/10.3102/01623737015002195

[56] usability.gov, “System usability scale.” [Online]. Available: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html

[57] S. Johnny, *Fundamentals of qualitative research*. Oxford University Press, 2011.

[58] F. Lau, S. H. Rubin, M. H. Smith, and L. Trajkovic, "Distributed denial of service attacks," in *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0*, vol. 3, Oct 2000, pp. 2275–2280 vol.3.

[59] N. Gupta, A. Jain, P. Saini, and V. Gupta, "Ddos attack algorithm using icmp flood," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, March 2016, pp. 4082–4084.

[60] Cloudfare, "Ping (icmp) flood ddos attack." [Online]. Available: https://www.cloudflare.com/learning/ddos/ping-icmp-flood-ddos-attack/

[61] M. Bogdanoski, T. Shuminoski, and A. Risteski, "Analysis of the syn flood dos attack," *International Journal of Computer Network and Information Security*, vol. 5, pp. 1–11, 06 2013.

[62] Emerging Threat, "Emerging threat suricata rules," 2019. [Online]. Available: https://rules.emergingthreats.net/open/suricata/rules/

[63] I. Salman, A. T. Misirli, and N. Juristo, "Are students representatives of professionals in software engineering experiments?" in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 666–676.

[64] S. Yoon, H. Park, and H. S. Yoo, "Security issues on smarthome in iot environment," in *Computer science and its applications*. Springer, 2015, pp. 691–696.

[65] Washington Post, "'I'm in your baby's room': A hacker took over a baby monitor and broadcast threats, parents say," 2018. [Online]. Available: https://www.washingtonpost.com/technology/2018/12/20/nest-cam-baby-monitor-hacked-kidnap-threat-came-device-parents-say/

[66] M. Nobakht, V. Sivaraman, and R. Boreli, "A host-based intrusion detection and mitigation framework for smart home iot using openflow," in *2016 11th International Conference on Availability, Reliability and Security (ARES)*, Aug 2016, pp. 147–156.

[67] C. Jun and C. Chi, "Design of complex event-processing ids in internet of things," in *2014 Sixth International Conference on Measuring Technology and Mechatronics Automation*, Jan 2014, pp. 226–229.