A User-friendly Approach to Write and Enforce Rules for Detecting Anomalous Network Traffic in IoT Environments

Davino Mauro Junior
Federal University of Pernambuco (UFPE)
Recife, Brazil
dmtsj@cin.ufpe.br

Kiev Gama
Federal University of Pernambuco (UFPE)
Recife, Brazil
kiev@cin.ufpe.br

ABSTRACT

Enforcing security on IoT devices is not an easy task, due to several vulnerabilities in many products that reach consumer shelves. With the rapid growth of the IoT market in the recent past there are specific network attacks targeting IoT devices, thus it is paramount to create mechanisms aiming this niche. Network Intrusion Detection Systems (NIDS, or IDS for short) can be used to employ defenses and detect anomalous traffic on IoT networks. However, due to the nature of these tools and the typical sysdamin users they target, usability is not one of the main concerns, with tools usually available through console and also demanding very specific network knowledge market. Since a large share of the IoT market is represented by consumers on Smart Home contexts, usability must be treated as a crucial feature on IDS systems that target IoT environments. We present a user-friendly approach that helps writing rules to enforce the detection of anomalous behavior on network traffic in IoT networks. This approach was applied in our platform that works as an IDS system monitoring network traffic that continuously applies rules programmed by its users or administrators.

CCS CONCEPTS

• **Software and its engineering** → *Software usability*; • **Security and privacy** → **Intrusion detection systems**.

KEYWORDS

Internet of Things, security, intrusion detection systems, usability

ACM Reference Format:
Davino Mauro Junior and Kiev Gama. 2020. A User-friendly Approach to Write and Enforce Rules for Detecting Anomalous Network Traffic in IoT Environments. In *IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3387940.3392248

1 INTRODUCTION

There are different approaches and tools addressing attacks where Internet of Things devices are targets. Among these solutions, one type stands out, 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 [12]. Due to the rapid growth of the IoT market, which is represented mostly by consumers on Smart Home contexts [3], it is important for these systems to be easy to use and extend. For example, in a real world scenario where one of these systems is employed in a Smart Home context, the user should be able to create new rules that map anomalous communication patterns. The capability of a system that allow creating new rules to match other patterns of anomalous enable to be always prepared to detect new threats. For that, **usability must be a crucial point of these systems**. However, previous work showed that usability was a difficult challenge for traditional Intrusion Detection Systems (IDS)s as the rules created and enforced on these systems were non-intuitive and difficult to understand [14]. Normally, providing ways to add new rules to the tools as to enable them to target detect more attacks, **it is overly complicated even to IT domain's users**[2, 14].

In this work, we propose a platform that enables users to create rules in an intuitive way with an user-interface (UI), while keeping the core of an Intrusion Detection Systems (IDS) with network monitoring and detection of anomalies using a pattern-matching method, commonly used on Signature-based Intrusion Detection Systems (IDS)s. As a preliminary evaluation of the platform's usability, we compare it with a traditional open-source IDS, Suricata [8]. We focused specifically on the process of creating the rules with a brief comparison of these systems, providing examples of rules for a DDoS attack.

2 BACKGROUND AND RELATED WORK

2.1 Intrusion Detection Systems

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 [12]. This analysis is usually done by identifying signature of well-known attacks. Once detected, the IDS triggers an alert to the users or even an automated response such as disconnecting a suspicious device from the network. In this work, we focus on Signature-based IDS due to its popularity on the IoT context [12]. There are mainly three types of IDS [6]:

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 [5].

Specification-based. This type of IDS is 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

ICSEW'20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7963-2/20/05…$15.00
https://doi.org/10.1145/3387940.3392248

events to detect an anomaly, this type of IDS uses manually developed specifications that capture legitimate system behaviors [11].

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 [6]. Popular tools of this type (e.g., Zeek, Snort, Suricata) 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*.

2.2 IDS and IoT

Many solutions addressing security concerns on IoT networks emerged recently. 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 focuses 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 [10]. IoT-IDM presented a Host-Based Intrusion Detection Systems (IDS) focusing on Smart Homes IoT [7] where a software-defined network and its protocol, OpenFlow, was used to detect intrusions with customized machine learning techniques and learned signature patterns of known threats.

The work of Werlinger et al. [14] showed that other than the expected difficulties involving configuration of IDSs, 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. 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.

Other studies have focused on variables such as performance, memory consumption and number of false positives/negatives by replicating network traffic while running the Intrusion Detection Systems (IDS)s. For instance, a comparison of two open-source Intrusion Detection Systems (IDS)s, Suricata and the system that originated it, Snort [1], pointed out that Suricatarequired 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 Intrusion Detection Systems (IDS)s has a better detection algorithm.

3 RULE WRITING APPROACHES COMPARED

We present two platforms that act as IDSs that support an extensible approach where new rules for detecting anomalous communication patterns can be added to the system. Suricata was chosen due to its popularity, for comparison to our platform [4] in a brief discussion.

3.1 Suricata

Suricata [8] is an open-source Signature-based IDS based on Snort [13] (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.

```
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 1: Suricata's Rule to detect an ICMP Flood Attack

3.1.1 Rule writing approach. 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 [9]. 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 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 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.

After creating the rule file (with a .rules extension), a user has to edit Suricata's main configuration file (usually located on */etc/suricata/suricata.yaml*) and add the name of the newly created file (as seen in Figure 1). Finally, the user (re)compile Suricata by restarting it so it can reload the configuration file and the newly created rule.

```
##
## Configure Suricata to load Suricata-Update managed rules.
##
## If this section is completely commented out move down to the "Advanced rule
## file configuration".
##

default-rule-path: @e_defaultruledir@

rule-files:
  - suricata.rules
  - new_rule.rules

##
## Auxiliary configuration files.
##

classification-file: @e_sysconfdir@classification.config
reference-config-file: @e_sysconfdir@reference.config
# threshold-file: @e_sysconfdir@threshold.config

##
## Include other configs
##

# Includes.  Files included here will be handled as if they were
# inlined in this configuration file.
#include: include1.yaml
#include: include2.yaml
```

Figure 1: Suricata's Main Configuration File

3.2 Our platform: IoT-Flows

We are developing a platform that focuses on surveilling communication between IoT devices. It acts on the different network layers, providing a multilayer defense for IoT environments, and being 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. 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/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, thus 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, Esper[1], and understanding the metadata of the packet structure in order to write the rules.

3.2.1 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.

Users can see which rules are active at any moment while also create, edit and delete rules, as per Figure 2. 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.

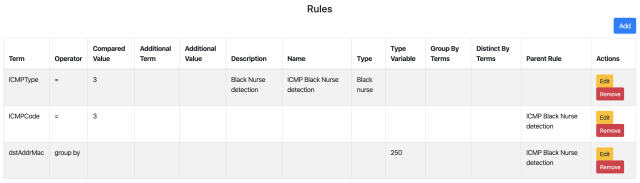

Figure 2: A set of rules beind listed on IoT-Flows

For the example on Figure 2, the rules being showed are used for detection of the ICMP Flood 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.

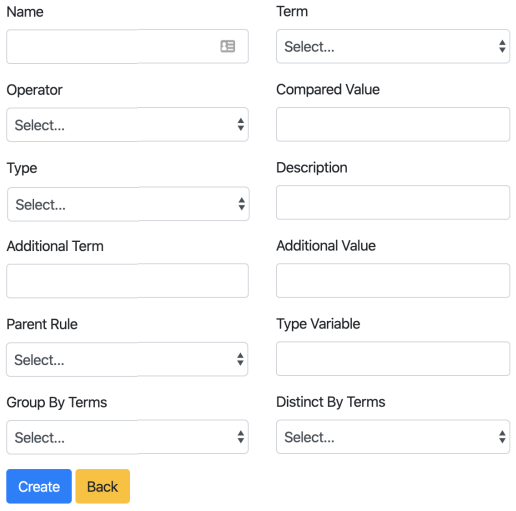

Figure 3: Form used to create a rule on IoT-Flows

Figure 3 shows the UI that enables the users to create a rule on IoT-Flows and contains the necessary network packet's fields. It also supports some advanced queries such as nested queries, with the usage of the "Parent rule" field exposed in the UI. Any rule to be create would basically be under the following template, where most UI fields are self-explained:

```
SELECT {Term|*} FROM NetworkPacket
[WHERE {Term} {Operator} {ComparedValue}]
[GROUP BY {Group_By_Terms}]
[HAVING COUNT(*) {AdditionalTerm} {TypeVariable}
```

[1] www.espertech.com/esper

```
SELECT * FROM NetworkPacket
WHERE ICMPType = 3
GROUP BY srcAddrMac
HAVING COUNT(*) > 250
```

Listing 2: Rule to detect an ICMP Flood Attack in our platform

An example of a generated rule is presented in Listing 2, which shows a rule for detecing an ICMP flood attack.

3.3 General Comparison

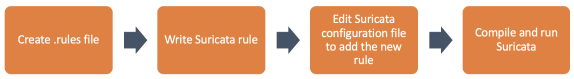

Figure 4: Suricata's Process for Creating a Rule

Figure 5: Process for Creating a Rule in IoT-Flows

The process of creating a rule for Suricata can be seen in Figure 4. 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 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, Suricata's main configuration file has to be edited and the name of the newly created file has to be added to it. In the last step, the user has to recompile the rule by restarting Suricata, so it can load all rules, including the new one.

Figure 5 illustrates the process for creating a rule or pattern on our platform. The user accesses the UI system, where existing rules are listed, then initiates the process for creating a new rule, which requires using the predetermined UI fields. Then the rules is automatically loaded into the platform.

When comparing both approaches, it can be seen that our approach takes less steps than Suricata. In addition, the manual steps from Suricata may hinder the adoption by less experienced users. The UI provided by our platform makes it more attractive to those who are not comfortable using the command line or dealing with configuration files. In addition, the level of complexity from the rules syntax in our platform resembles the popular SQL approach, as it can be seen in Listing 1 and Listing 2.

4 CONCLUSIONS AND FUTURE WORK

Currently there are many security flaws in IoT devices and applications that have been exploited by malware. IDSs are tools that allow analyzing network traffic by monitoring network nodes (e.g., the IoT devices) and its data packets with the goal of detecting anomalies in network communication. Some IDS solutions have recently started to address security concerns specific to IoT networks. In some IDSs, users are able to write themselves the rules that map the patterns of suspicious communication. However, these systems were non-intuitive and difficult to understand, thus being too complicated even to IT domain's users. We proposed a platform that enables users to create rules in an intuitive way with a user-interface (UI) and an underlying engine that allow matching the patterns written on those rules to detect anomalous communication. We compared our approach agains Suricata, one of the most utilized IDSs. Our approach is promising, since it presentes a UI that avoids users to deal with syntax details of the rules and keeps them away from dealing with the command-line or configuration files. As future work, we intend to evaluate our approach under the perspective of users. We also see the possibility of integrating this approach with other tools such as Suricata.

ACKNOWLEDGMENTS

This work is supported by RNP (Brazil) under Grant No. 002951 and by NSF (USA) under Grant Nos. 1740897/1740916.

REFERENCES

[1] Eugene Albin and Neil C Rowe. 2012. 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, 122–127.
[2] Dr Saad Butt and Vera Anatol'evna Gnevasheva. 2018. Efficiency in the Processes of Intrusion Detection System Through Usability Evaluation Methods. *Available at SSRN 3151216* (2018).
[3] Forbes. 2005. 2017 Roundup Of Internet Of Things Forecasts. https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts/#7b71aae11480
[4] Davino Mauro Junior, Walber Rodrigues, Kiev Gama, José A. Suruagy, and Paulo André da S. Gonçalves. 2019. 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*. IEEE / ACM, 17–20.
[5] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. 2013. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications* 36, 1 (2013), 16 – 24.
[6] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. 2013. A survey of intrusion detection techniques in Cloud. *Journal of Network and Computer Applications* 36, 1 (2013), 42 – 57. https://doi.org/10.1016/j.jnca.2012.05.003
[7] M. Nobakht, V. Sivaraman, and R. Boreli. 2016. 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)*. 147–156.
[8] OISF. [n.d.]. Suricata Open Source IDS. https://suricata-ids.org/
[9] OISF. [n.d.]. Suricata Rules. https://suricata.readthedocs.io/en/suricata-4.1.5/rules/intro.html
[10] Shahid Raza, Linus Wallgren, and Thiemo Voigt. 2013. SVELTE: Real-time intrusion detection in the Internet of Things. *Ad hoc networks* 11, 8 (2013), 2661–2674.
[11] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. 2002. Specification-Based Anomaly Detection: A New Approach for Detecting Network Intrusions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security* (Washington, DC, USA) *(CCS '02)*. Association for Computing Machinery, New York, NY, USA, 265–274.
[12] Tariqahmad Sherasiya, Hardik Upadhyay, and Hiren B Patel. 2016. A survey: Intrusion detection system for internet of things. *International Journal of Computer Science and Engineering (IJCSE)* 5, 2 (2016).
[13] Sourcefire. 1998. Snort. https://www.snort.org/
[14] Rodrigo Werlinger, Kirstie Hawkey, Kasia Muldner, Pooya Jaferian, and Konstantin Beznosov. 2008. The Challenges of Using an Intrusion Detection System: Is It Worth the Effort?. In *Proceedings of the 4th Symposium on Usable Privacy and Security* (Pittsburgh, Pennsylvania, USA) *(SOUPS '08)*. Association for Computing Machinery, New York, NY, USA, 107–118.