s2ipt: A Lightweight Network Intrusion Detection/Prevention System based on IPtables

Gerardo Canfora[1], Antonio Pirozzi[1,2] and Aaron Visaggio[1]
[1]*Department of Engineering, University of Sannio, Via Traiano, 3, 82100 Benevento BN, Italy*
[2]*KOINE srl, Via Porta Rufina 13, Benvento 82100, Italy*

Keywords: Intrusion Detection System, Intrusion Prevention System, IPtables, Snort, NIDS, NIPS, Application-level Control Flow, Netfilter.

Abstract: Since each organization has its own security culture and background, there is not an *out-of-the-box* solution that fits all the possible security requirements. There may be some contexts in which it is necessary to monitor and prevent certain application-level attacks with less impact on pre-existent configuration. For example, there may be some constraints on processing resources of some embedded devices. Starting from this consideration, we developed *s2ipt*, a python-powered tool which aims to implement a lightweight Netfilter-based network intrusion detection and prevention system (IDS/IPS) by translating Snort community rules into iptables rules-set. *s2ipt* utilizes the netfilter string matching module to detect application-level attacks. Netfilter reduces the impact on a system, has less memory and CPU footprint, which makes it suitable to run even on low-cost devices than a solution like Snort. s2ipt allows iptables to detect application layer attacks in a transparent way, in fact it only adds new application layer ruleset leaving the existing ones unchanged.

1 INTRODUCTION

One of the most powerful and widespread firewall, already integrated within Linux machine, is iptables (ipt,). Iptables is a stateful packet filter firewall embedded in the most UNIX installations, and is based on Netfilter module; however it lacks the functionality of inspecting the Application layer data. Another network defence technology is Snort (sno,), a widely deployed network intrusion detection and prevention system (IDS/IPS). The two technologies are complementary since iptables can be adopted to a lower level than Snort, while Snort can perform filtering at application level. For this reason the two technologies can be integrated, obtaining the following benefits: iptables could inherit the rich set of rules from Snort, and, differently from Snort, can be adopted as a firewall on thin devices like wearable devices and Arduino controllers. Moreover, as authors in (Zhong, 2016) claim, to elicit attack path patterns by modeling and mining the traces of analysts' data triage operations is feasible and necessary, thus technologies that capture such information must be developed and adopted by organizations. This paper presents a tool called *s2itp* which aims to adapt Snort ruleset to iptables, which is made possible by the string matching support of Netfilter which adds content inspection capabilities to iptables.

s2ipt allows iptables to detect application layer attacks in a transparent way, in fact it only adds a new application layer ruleset leaving the existing ones unchanged. Since this tool produces a low system footprint, it is suitable also for low-powered devices. s2ipt combines logging, network based intrusion detection and prevention system within a packet filter firewall.

The usage of Snort rules in scenarios with poor computational resources is scarcely discussed in literature: at the best knowledge of the authors only Arney and Wang (Arney and Wang, 2016) investigate the relationship between the number of enabled rules maintained by Snort and the amount of computing resources necessary in low-powered systems as the IoT devices. The paper proceeds as follows: section 2 illustrates the solution in detail; section 3 shows a validation of the tool; and, finally, section 4 draws the conclusions.

2 SOLUTION

Establishing a correspondence between a Snort rule and an Iptables' one is not always possible. For this

Canfora, G., Pirozzi, A. and Visaggio, A.
s2ipt: A Lightweight Network Intrusion Detection/Prevention System based on IPtables.
DOI: 10.5220/0006431904620467
In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE 2017) - Volume 4: SECRYPT*, pages 462-467
ISBN: 978-989-758-259-2

reason *s2ipt* performs a "best-effort" translation, i.e. at the best of the possibilities offered by the Iptables' syntax. Unfortunately Snort provides for functions of a higher level than Iptables, as well as a regular expressions engine, which are not available in Iptables.

A Snort rule is formed by a two elements-structure: *header(options)*, where the first part is fixed, while the second one is variable and depends on the options of the rule. The *header* includes the following fields:

- *action*: indicates the action to take when the packet specified by the rule is intercepted;
- *protocol*: indicates the protocol the packet is associated to (Snort supports TCP, UDP, IMCP, and IP);
- *Source IP*: indicates the source IP;
- *Source port*: indicates the number of the source port;
- *Operator*: indicates the direction of the packets and three modes are considered: from source to destination, and in both the directions between the source and the destination;
- *Destination IP*: indicates the destination IP;
- *Destination Port*: indicates the number of the destination port;

All the options within a Snort rule are expressed in the form: *keyword:value* where *keyword:* represents the type of option, and *value* the corresponding value. These options can be classified into four categories:

- *General*: provides information about the rule without effects on the rule's execution;
- *Payload*: allows to find data within a packet's payload, and different rules of this kind can be correlated to form a one only rule;
- *Non-Payload*: allows to look for information outside the payload;
- *Post-detection*: lets to define mechanisms to be triggered after the rule matching;

Iptables rules can be defined by command line within a Linux operating system with this structure: *iptables[-t⟨table-name⟩]⟨command⟩ ⟨chain-name⟩\⟨parameter-l⟩⟨optiono-l⟩\⟨paramter-n⟩⟨oprtion-n⟩* where:

- ⟨*table-name*⟩ indicates which table the rules must be applied to;
- ⟨*command*⟩ indicates the action to perform;
- ⟨*chain-name*⟩ indicates the chain to modify, create or remove;
- the pairs ⟨*parameters*⟩-⟨*options*⟩ represents the parameters and the corresponding values of the options associated, which define how a packet which triggers a rule must be handled.

The high flexibility allowed by the Iptables syntax makes a rule's length and complexity to significantly vary with regards to its goal. Finally, in order to create a valid Iptables rule, three key further parameters must be defined:

- *Packet type*, which is the type of packet to be monitored;
- *Packet source/destination*, which is the packet's source or destination that must be filtered;
- *Target*, which defines the type of action to perform when the rule is matched.

2.1 Translating Algorithm

The approach used for the translation is based on a "best effort" philosophy, due to the lack of a perfect correspondence between the Snort syntax and the Iptables syntax. When the correspondence between a Snort rule and an Iptables rule is complete, the algorithm translates the rule from a syntax to the other one. When there is not a complete correspondence, the algorithm rephrases the semantics into the syntax allowed by Iptables. If this attempt fails, the translation of the entire rule fails. As first step, the algorithm translates the *header* of the Snort rule, and than it copes with the single options. The translation strategies adopted are the following:

- *Action*: although the Snort sintax supports eight different types of action, *s2ipt* takes into account only the rules with the *alert*.
- *Protocol*: in this case there is a 1:1 mapping with the Iptables option `Iptables -p protocolname` where all the protocols supported by Snort can be made explicit;
- *Source/destination IP*: for specifying the IP addresses, sources and destinations, the commands `-s IPaddress` and `-d IPaddress` are respectively used. Nevertheless, the Snort syntax provides for static variables tagged as *$HOME_NET* and *$EXTERNAL_NET* (representing the local network and the the external network), which are not supported by Iptables, and are not translated by the algorithm. Snort lets specify the multiple IP addresses, classes of IP addresses (in *CIDR* notation) and the negation of a certain IP address. All these options are supported by Iptables (and thus translated by *s2ipt*) except for the negation of multiple IP addresses;

- *Source/destination port*: Snort supports three different fields for the number of source/destination ports. Iptables makes use of `--sport value` and `--dport value` options. In this case there is not a direct correspondence, but we adopted a trick. Iptables allows the definition of the port numbers, like Snort. Additionally, Iptables lets to define an interval, separating the extremes with a colon. For defining any port (corresponding to `any` in Snort), the user must omit the port number in Iptables. Iptables allows to exclude a specific port number, as well as Snort, but for indicating a set of port numbers, Iptables requires the `-m multiport` module before the commands for the specification of the port numbers;
- *Operator*: there is not a correspondence for the Snort's direction tag in Iptables, where the same information is represented by the concept of *chain*. In the case of *s2ipt*, a custom chain is created, which is referred to by the Iptables engine, as discussed later in the paper.

Once the *header* is translated into a Snort rule, the algorithm examines the options, analyzing them singularly, trying to accomplish the translation. Even if not all the Snort options can be translated in Iptables rules as they are not supported, the most common ones can be mapped.

All the Snort options that *s2ipt* can translate into the Iptables format are listed along with the adopted assumptions and strategies:

- *Content*: allows to find a specific byte sequence within a payload, applying the Boyer-Moore algorithm. The Iptables extension that performing the "string-matching" can be selected with the command `-m string -string "string" -- algo bm`, where *string* is the string to search within the payload. Snort allows the search of hexadecimal value that can be accomplished with Iptables option `-m string --hex-string`;
- *Uricontent*: allows to handle data codified in URLs, transferred on HTTP. *S2ipt* manages this option as well as the previous one;
- *Offset*: searches for a content within the payload starting from an offset location. Iptables allows the same option with the command `--from`;
- *Depth*: forces that the packet matching does not overcome a certain number of bytes after the payload. In Iptables this option is specified through the command `--to` followed by the number of desired bytes. This and the following option can be present in more than an instance within a Snort rule. This is not allowed by Iptables, with the additional constraint tha the number of bytes indicated by `--from` be smaller than that indicated by `--to`. In such case, *2s2ipt* sums the values of the *depth* (`--to`) fields, considering the minimum value of the *offset* (`--from`) field;
- *TTL*: allows to match a packet relying on the TTL value within the IP header. Both Snort and Iptables allow to specify a value of TTL smaller or greater than, and equals to a certain number;
- *Tos*: indicates the bit within the *Type of Service* field in the IP header. This option is supported by Iptables with the command `-m tos --tos value`;
- *Flow*: this Snort option allows to define a specific state of a connection: for instance the combination of *from_client* and *established* lets to analyze only the packets on the client side of a TCP connection after the three-way handshake was completed. Iptables lets to filter the state of a TCP connection with `-m state --stat CONNECTION_STATE` command. Differently from Snort, Iptables allows to check only five states (INVALID, ESTABLISHED, NEW, RELATED and UNTRACKED). For this reason we made the decision to consider in *s2ipt* only the Snort rules referring to the ESTABLISHED;
- *Ip_proto*: resctricts the scope of rule on the basis of the *protocol* field of the IP header.Iptables lets to specify the same option with the command `-p` followed by the type of the desired protocol;
- *Pcre*: stands for *Perl Compatible Regular Expression* and lets Snort to apply regular expressions of high complexity. Since Iptables does not support any kind of regular expression, for this option the algorithm must use mainly the best-effort approach. *s2ipt* is able to translate only a little subset of all the possibile variants of Snort's *pcre*. The algorithm can translate only those rules whose *pcre* option contains constant values, as it begins the aforementioned *content*. Another case that allows the transaltion in Iptables rule is represented by static patterns defined by numeric values.

The complete algorithm of translation includes four main phases:

- *Pre-processing*: the tool carries out a preliminary evaluation of the rules for excluding those that can not be translated because of a missing correspondence between the Snort and the Iptables sintax. The remaining rules are gathered in two sets: one contains the part concerning the header and the other contains the part concerning the option;

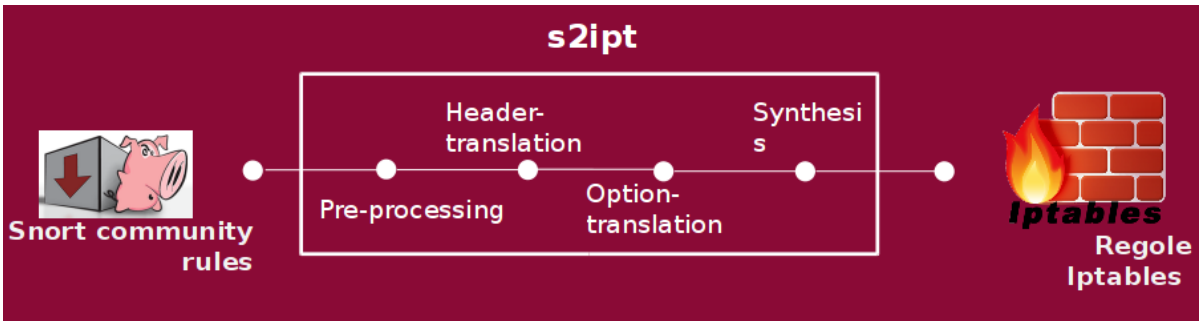

Figure 1: s2ipt. Translation process.

- *Header translation*: all the fields of the Snort rule's header are properly translated in the corresponding Iptables rule(s);
- *Options translation*: all the options of the Snort rule's are properly translated in the corresponding Iptables rule(s);
- *Synthesis*: the result of the translation is transformed into a string used as Iptables command.

The translation process is depicted in Fig. 1

2.2 Best-effort Policy

The lack of a complete correspondance between the Snort and Iptables sintax lead us to define a "best-effort" policy which consists into the generation of an Iptables rule which is similar to the corresponding Snort rule, but that is not an exact translation. The policy is defined in this way:

- if the Snort rule contains an *action* different from `alert`, the rule is not translated;
- if the number of Snort options within an input rule is greater than the number of options that *s2ipt* is able to translate, the rule is considered translated in "best-effort";
- if the rule contains the *pcre* option, and there are patterns that can not be translated, the rule is discarded, otherwise the rule is considered translated in "best-effort";
- if the rule contains the negation of a range of IP addresses as source or destination, the rule is not translated;
- if the the rule contains a *content* option, it is not translated as it is considered too generic.

Each rule is assessed against this list of controls in order to decide if the translation is in "best-effort", or not.

2.3 Coverage of Translation

In order to obtain a first assessment of the level of the translation coverage, s2ipt was launched getting as input data the complete rules-set of the Snort community(at 06/16/2016). In particular, compared to 3419 rules taken into account, the tool is able to correctly translate 2860 rules, ie approximately the 84%. The *best-effort* approach was applied to 2268 rules (about 80%), while the remaining 592 have been translated *1:1* by the tool. Such a high percentage in the best-effort is essentially justified by two reasons. First, many options are not taken as part of the algorithm translation because they did not have corresponding options in Iptables. Furthermore, since the Iptables syntax does not provide for the use of regular expression patterns, whenever the algorithm meets the *pcre* option in a Snort rule and manages to translate it, s2ipt makes the conservative assumption for which the translation is done in best-effort.

Let's see how *s2ipt* translates a Snort rule; let's consider the following rule: `alert`

```
tcp $HOME_NET any -> $EXTERNAL_NET any
(msg:"MALWARE-BACKDOOR Infector.1.x";
flow:established,to_client; content:
"WHATISIT"; metadata:ruleset community;
reference:nessus,11157; classtype:
misc-activity; sid:117; rev:16;)
```

This rule tracks all the packets that refer to an established TCP connection headed to the client application, whose content includes the "WHATISIT" string at application layer. a packet having these characteristics will be written in the log with the tag: "MALWARE-BACKDOOR Infector.1.x"". Such a rule does not contain any option that is not supported by Iptables, thus it will be straight translated by s2ipt. In particular, the translation process will analyze the header content as first, then it will scan all the options of the rule. The result of the translation will be:

```
    iptables -I CUSTOM_CHAIN 1 i IFACE p
tcp m state --state ESTABLISHED m string
--string "WHATISIT" --algo bm -m comment
--comment "MALWARE-BACKDOOR Infector.1.x"
-j LOG --log-prefix [IDS-117]
```

where, as it is easy to observe, the options regarding metadata have been ignored (metadata, reference, classtype, sid and rev), whereas the remaining have been taken as they are. In particular, CUSTOM_CHAIN indicates a chain where the rule (IDS or IPS) must be added and IFACE represents the interested interface (eth0, wlan0, and others).

3 VALIDATION

The goal of the validation phase is to evaluate the effectiveness of s2ipt and the accuracy of the produced iptables rules. Since these rules are generated from semantically equivalent Snort rules, the validation has been focused on the analysis of the observed behavior of the iptables firewall (configured with s2ipt) compared to the corresponding Snort firewall while traversed by the same traffic pattern. Three classes of test cases were defined: port scanning, attacks against Apache Web Server, and Web Application Fuzzing. The results of the two systems were then observed and compared. All the tests were carried out starting from 2860 Snort rules published by the same community on 06.16.2016, which s2ipt was able to translate.

3.0.1 Port Scanning Test Case

This test case consisted of 12 different port scanning techniques defined as follows:

- *Syn Scan* : Sending a SYN (synchronization) packet to initiate a three-way handshake.
- *Fin Scan* : Sending a packet with just the TCP FIN bit set.
- *Null Scan* : Sending a packet without setting TCP flags.
- *Maimon Scan* : This probe is exactly the same as NULL, FIN, and Xmas scans, except that the probe is FIN/ACK.
- *Xmas Scan* : Sending a packet with all TCP flags set. Sets the FIN, PSH, and URG flags.
- *Connect Scan* : Full TCP scan, trying to establish a 3-way handshaking.
- *Ack Scan*: Sending a packet with only the ACK flag set.
- *IP Protocol Scan* : IP protocol scan allows to determine which IP protocols (TCP, ICMP, IGMP, etc.) are supported by target machines.
- *Intensive Scan* : Scan the most common TCP ports. It will try to determine the OS type and which services (and versions) are running.
- *Intensive Scan plus UDP* : Same as the regular Intense scan, but UDP ports are also scanned.
- *Intensive Scan plus TCP* : Same as the regular Intense scan, but TCP ports are also scanned.

The results given in Table 1 show the results of the I test case (port scanning):

Table 1: Test Case I : Port Scanning.

Type of Scan.	Snort Alerts	Ipt Alerts
Syn Scan	0	0
Fin Scan	0	0
Null Scan	0	0
Maimon Scan	0	0
Xmas Scan	0	0
Connect Scan	0	0
Ack Scan	0	0
IP Protocol Scan	0	0
Intensive Scan	5	11
Intensive Scan plus UDP	8	15
Intensive Scan plus TCP	8	11

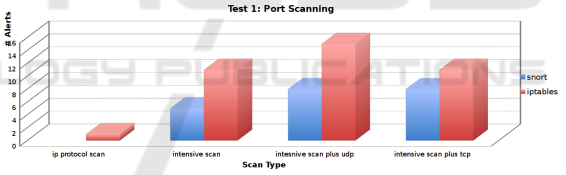

Figure 2: Test Case I : Port Scanning.

As we can notice from the histograms in figure 2, both Snort and iptables generated alerts for the following types of port scanning: *ip protocol scan*, *intensive scan*, *intensive scan plus UDP*, *intensive scan plus TCP*. We notice that iptables generates some more alerts, because the *best effort* approach tends to relax iptables rules that becomes more greedy.

3.0.2 Attacks against Apache Web Server

This test case consisted of attacking Apache web server using Metasploit and then see if Snort and iptables would be able to detect the attacks. The first attack, also known as Apache Killer, is called Apache Range Dos Attack *CVE-2011-3192* (CVE,). It submits a single short request to trigger these applications into replying with a large amount of arbitrary data, causing the service to become temporarily unresponsive. The Apache Killer attack was car-

ried out with the Auxiliary Metasploit module *auxiliary/dos/http/apache_range_dos)*. The second attack is against *mod_isapi* extension in Apache and it was carried out with the Auxiliary Metasploit module *auxiliary/dos/http/apache_mod_isapi*.
The results given in Table 2 show the results of the II test case (Attacks against Apache Web Server) :

Table 2: Test Case II : Attacks against Apache Web Server.

Type of Attack.	Snort Alerts/pkt	Iptables Alerts/pkt
Apache Range DoS Attack	1/1	3/1
Apache mod isapi	2/20	3/20

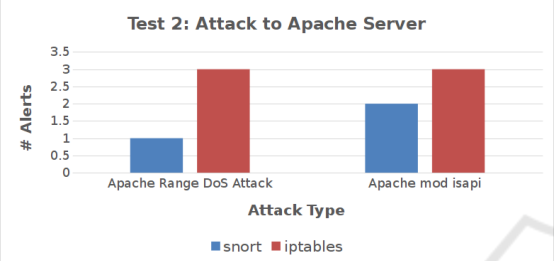

Figure 3: Test Case II : Attacks against Apache Web Server.

As we can see from the histograms in Figure 3, iptables tend to generate more alerts due to the ”best effort approach” in the translation process but the important thing is that the attack had been detected by iptables as well as snort.

3.0.3 Web Application Fuzzing

This test case consisted of sending Random Data against both Snort and iptables and verifying if they would be able to detect the attack. To carrying out this test case, ZAP Proxy was used (OWASP,). The results given in Table 3 are related to the III test case:

Table 3: Test Case III : Web Application Fuzzing

Type of Attack.	Snort Alerts/pkts	Iptables Alerts/pkts
Web application Fuzzing	1/53	5/53

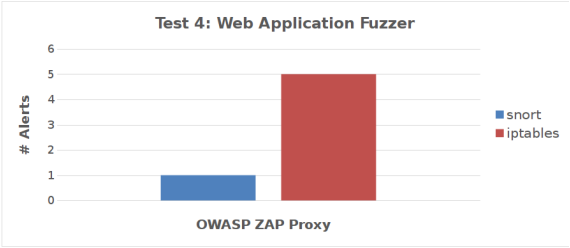

Figure 4: Test Case III : Web Application Fuzzing.

4 CONCLUSION

s2ipt is a tool that lets to configure the iptables firewall as an NIDS/NIPS based on the Snort community rules, in a simple and transparent way, without observing substantial changes to the environment. The power of s2ipt lies in the combination of the positive aspects of two different types of technologies: on one hand the power of Snort syntax and the support contributed by his community, on the other hand the efficiency resulting from the use of the Linux kernel and the simplicity of iptables commands, which have a minimal impact on system resources.

ACKNOWLEDGEMENTS

This Research was supported thanks to the contribution of Luigi Pino, Alessandro Esposito and Luciano Ocone of the Department of Engineering of University of Sannio.

REFERENCES

Cve-2011-3192. http://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2011-31922. Accessed: 2017-02-09.

iptables. https://www.netfilter.org/. Accessed: 2017-02-09.

Snort ids. https://www.snort.org. Accessed: 2017-02-09.

Arney, C. A. and Wang, X. (2016). Active snort rules and the needs for computing resources: Computing resources needed to activate different numbers of snort rules. In *Proceedings of the 5th Annual Conference on Research in Information Technology*, pages 54–54. ACM.

OWASP. Zap proxy. https://github.com/zaproxy/zaproxy. Accessed: 2017-02-09.

Zhong, C., Y. J. L. P. . E. R. F. (2016). Automate cybersecurity data triage by leveraging human analysts' cognitive process. In *IEEE International Conference on Intelligent Data and Security (IDS)*, pages 357–363. IEEE.