26 | Automatic DDoS attack signature generation for SNORT and Suricata

Author: Vincent Dunning

Abstract

Our society is increasingly dependent on services in the internet such as banking, reading the news and shopping from anywhere at any time. This depency makes it attractive for malicious people to take these services down. Distributed Denial of Service (DDoS) attacks aim to do this and are getting increasingly stronger, causing average revenue losses of over $300K per hour. Because of this, effective solutions are needed to block these attacks. Our hypotheses is that Signature-based intrusion detection systems (SIDS) are suitable for this task with the problem, however it is a cumbersome process to maintain an up-to-date list of signatures. In this paper we try to overcome this problem for two SIDSs; Snort and Suricata. In this paper we propose a way to automatically generate a signature based on an attack fingerprint from DDoSDB and measure the performance of these signatures. We show that we can generate these rules on average withing 10ms and that both IDSs can detect incoming attacks on average within a second, where SNORT proves to be more consistent than Suricata.

26.1 Introduction

Our society is increasingly dependent on services in the Internet. For example, online banking, reading the news and shopping from everywhere at any time. This dependency also makes it attractive for people with bad intentions to take these services offline. One example to make services offline is a Distributed Denial of Service (DDoS) attack. A DDoS attack is composed of multiple machines that send more network traffic to a target than it can handle, making it's service offline temporarily. In 2017, the number of DDoS attacks was around 50 million (*What is a ddos attack*, n.d.), which demonstrates a very high occurrence. DDoS attacks have also been increasing in power. In 2011, the record peak was 60Gb while in 2016 it was already 1.1Tb/s (Santanna, 2017). As the number of attacks is increasing and one attack can cause an average revenue loss of $5,600/minute, or over $300K/hour (*What is a ddos attack*, n.d.) sophisticated tools are needed in order to identify and protect against these attacks. For this we need a way to detect incoming attacks and drop packets that are associated to an attack. Intrusion Detection Systems (IDS) (Vasilomanolakis, Srinivasa, Cordero, & Mühlhäuser, 2016) are systems that can perform this task and could prove as a solution.
There are two main categories of IDSs: Signature-based (SIDS) and Anomaly-based (AIDS) (Liao, Lin, Lin, & Tung, 2013). A SIDS uses the process of comparing patterns in network data to known malicious data that correspond to an attack. This process can essentially be seen as fingerprint matching to detect DDoS attacks. AIDS creates a profile of the expected or regular behavior of a user of a system. It does this by monitoring the network traffic continuously and detects deviations from this constructed profile of a regular user, a DDoS attack here.
AIDS is in general less accurate than SIDS but has the advantage that it also works for unknown attacks. The advantage of SIDS is that it is a faster and effective method to detect attacks that follow the same fingerprint as known attacks. However, for unknown attacks this IDS type fails completely, leading to types of attacks bypassing the system. Furthermore, SIDS needs to be updated frequently with signatures for new types of attacks to stay accurate. However, with adequate rules SIDS is more accurate than AIDS and for systems such as banks, that can not afford to have any downtime, SIDS is the better solution to defend against DDoS attack. However, Manual generation of these signatures often requires hours or even days of work this is not feasible for ongoing attacks (Szynkiewicz & Kozakiewicz, 2017). The goal of this research is to overcome this problem and automatically generate signatures against ongoing DDoS attacks.

To pursue our goal, we have defined the following research questions (RQ) as the basis of our research:

- **RQ1:** What are the key characteristics of DDoS attacks that can be used for automatically generating signatures for the most used SIDSs?
- **RQ2:** What is the performance of automatic rule generation for a given fingerprint of a DDoS attack?
- **RQ3:** How efficient are the automatically generated rules for ongoing DDoS attacks?

In the next section we will look at how DDoS attacks work and give an overview of what are currently the most common DDoS attacks. Furthermore, we will look at the two IDSs that will be used in this research, SNORT and Suricata and the rules they use to detect incoming attacks. In section 26.3, we will look at the methods we used to generate our signatures and the setupand commands to conduct the experiments to measure the performance of automatic rule generation. In section 26.4, we show the results of these experiments and discuss them. Finally in section 29.6, we conclude this research and give directions for future work.

26.2 Background

In this section we first explain how DDoS attacks work. Then we elaborate on what are the most common attacks according to the Akamai State of the internet security report from Q4 2017 (*Akamai, state of the internet security report*, n.d.) and what their characteristics are. Akamai is the largest distributed platform on the internet that monitors 24/7 and their security knowledge is trusted by some of the biggest companies in the world (*Why Akamai*, n.d.) which is why we took their list of most common DDoS attacks. Furthermore we will look at the two most used SIDSs that will also be used in this research, Snort and Suricata and how their rules work.

26.2.1 Working of a DDoS attack & Attack types

In figure 26.1 an overview of how a DDoS attack works is depicted. A DDoS attack is started by an attacker (a). This attacker communicates with the command & control (C&C) (b) that contacts a set of infected machines (c), often referred to as *bots* that can usually attack the victim (e) in two different ways. The first one is indicated with the dotted line, where the bots attack the victim directly. The other method is by using public servers (d) such as as a DNS to send an indirect attack to the target.

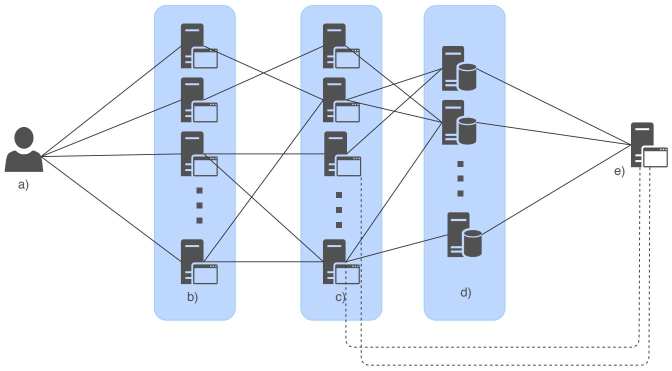


Figure 26.1: Infrastructure of a DDoS attack

In table 26.1 an overview can be found of the most common types of DDoS attacks and the main characteristics that can be used to identify these attacks. Furthermore we can find here whether they operate over a direct (straight lines) or indirect (dotted lines) connection to the victim machine from figure 26.1.

UDP flood is an attack type in which the attacker overwhelms random ports on the victim machine with UDP packets (*Security Glossary: Top 12 DDoS Attack Types You Need To Know*, n.d.). The victim machine will then have to check if an application is listening on those ports which consumes resources on the machine. The attacker might spoof its IP address to anonymize the attack and make sure response packets do not get back to his own machine.

UDP fragment exploits datagram fragmentation mechanisms. Every network has a unique maximum transmission unit (MTU) that is a limit on the size of the datagrams it can process. UDP fragment attacks transmit UDP packets bigger than the network's MTU causing them to be fragmented resulting in more packets arriving at the victim machine.

DNS amplification exploits vulnerabilities in DNS servers. The attacker sends requests to one or more DNS servers with a spoofed IP, the victims IP, prompting the server to reply back to the victim machine with a DNS response. Apart from the anonymity that comes with this way of attacking, the possibility of amplification is even more dangerous. By smartly exploiting the discrephancy between DNS request and response sizes, the attacker can send relatively small queries and get responses that are over 70 times larger, increasing the rate as which the victims resources are depleted.

NTP amplification attacks are similar to DNS amplification attacks in the sense that they also consist of making spoofed requests to servers, but instead NTP servers are used that normally are used to synchronize clocks on machines. Amplification is done using the 'monlist' command that will cause the network to send back a list of the last 600 hosts that connected to the server, a message that anywhere from 20 to 200 times larger than the request.

CLDAP attacks exploit the Connection-less Lightweight Directory Access Protocol (CLDAP). The protocol runs over UDP and is similar in nature to DNS and NTP attacks.

Chargen exploits services running the public Character Generator Protocol (Chargen). When a Chargen service receives a UDP packet, it responds with a number between 0 and 512 characters long (J., n.d.). It is similar to the NTP and DNS attacks.

SYN flood attacks exploit the three-way-handskake from the TCP protocol. With a SYN flood attack, the attackers hits the vicitim with a large amount of connection requests (SYN packets) to which the victim responds with SYN-ACK packets. The attacker never responds with an ACK packet, leaving all the connections half initialized and unusable by legitimate users.

ACK flood attacks exploit the TCP protocol as well. A large number of ACK packets, not associated with any connection, are sent towards the the victim that in turn drops all the packets. This can take up so many resources that the service the victim offers become unavailable to legitimate users.

HTTP flood attacks are different compared to these attacks because they operate on the application layer instead of the transport layer. The attack consists of sending a large number of HTTP requests to the target, depleting its resources. The attack is the most effective if it requires a lot of resources from the victim machine. More knowledge about the vulnerable application is needed to successfully invoke a HTTP flood attack, but in return they are often harder to detect and mitigate.

Attack Type	Main characteristics	Direct or indirect
UDP	UDP	Direct
UDP Frag	IPv4 & Fragments	Direct
DNS	UDP & src_port=53 & DNS_query && DNS_type	Indirect
CLDAP	UDP & src_port=389	Indirect
NTP	UDP & src_port=123	Indirect
Chargen	UDP & src_port=19	Indirect
SYN	TCP & flag=SYN	Direct
ACK	TCP & flag=ack	Direct
HTTP	HTTP & HTTP_request & src_port=80	Direct

Table 26.1: Overview of characteristics of common DDoS attacks

In the next subsection we will discuss SNORT and Suricata and the way we could use the results from this section to automatically generate rules.

26.2.2 SNORT & Suricata

In this subsection we will look at the two Intrusion Detection Systems, SNORT and Suricata and the rules they use the monitor network traffic. We chose for SNORT and Suricata because they are amongst the most popular SIDSs. Bro is an alternative as well but is mostly only used in scientific settings (Albin & Rowe, 2012).
SNORT and Suricata are both Intrusion Detection Systems (IDS) that work with a set of rules and compare incoming traffic to the ruleset they have. In fact, they use the exact same syntax for their rules. This means that the rules we can generate for SNORT will also work for Suricata and vice versa.
While SNORT and Suricata are highly similar, there are some differences that set them apart. First of, Suricata was first released more than 10 years after SNORT, 2010 versus 1998. This means that Suricata contains more modern features, making it more future proof. There are two main differences in the way SNORT and Suricata work. First of all, Suricata supports multithreading while SNORT can only run on a single thread (*Open Source IDS: Snort or Suricata?*, n.d.). This means that Suricata can do much more comparisons than SNORT on the same hardware which decreases the amount of packets that unknowingly pass through the IDS if it can not cope with the amount of traffic sent to the protected machine.
Furthermore, Suricata supports application-layer detection while SNORT does not. This means that SNORT is not able to mitigate the aforementioned HTTP flood attack while Suricata can.

26.2.2.1 Rules

An overview of the general structure of SNORT & Suricata rules is shown in listing 26.1 (*Understanding and Configuring SNORT Rules*, n.d.). As can be seen, these rules are composed of 7 mandatory fields and extra options (rule options).

```
{action} {protocol} {src ip} {src port} {direction} {dst ip} {dst
    port} ([rule options])
```

Listing 26.1: General structure of SNORT/Suricata rule

Here the action can be alert, log or pass. If alert is used, it will only alert that it found a matching packet, log will as well log the packet and pass ignores the packet and drops it. SNORT support 3 protocols, namely TCP, UDP and ICMP. In the src_ip field, we can use one specific source IP, a range of IPs, a list of certain IPs or 'any', indicating that the IP does not matter. In the src_port field, only one port or a range of ports can be used. An example of a port range can be '1000:' will include all ports above 1000.
The direction field can be "<>", indicating traffic in both directions or "->", indicating data in a specific direction, the other side of the direction, dst_ip and dst_port work the same as the sources address and port.
The most important rule options for this research will be 'msg', which allows us to report a specific message if a packet was matched on the rule, 'content', which finds a data pattern inside the packet, 'flags', which looks at which flag bits are set inside the TCP header and 'itype', which looks for a given icmp type packet.
2 example rules will now be shown. The first rule can be found in listing 26.2. It looks at icmp type 3 packets and alerts with the message 'ICMP test'. Listing 26.3 shows another rule, it looks at traffic coming from either IP 192.186.0.1 or 192.186.0.2 using the udp protocol to access port 53 (DNS) and a DNS query 'suspiciousurl.ru'.

```
alert icmp any any -> any any (msg:"ICMP test"; itype:3)
```

Listing 26.2: ICMP type 3 alert rule

```
alert udp [192.186.0.1, 192.186.0.2] any -> any 53 (msg:"Suspicious
    DNS request found"; content:"suspiciousurl.ru")
```

Listing 26.3: DNS request alert rule

In the next section we explain the methods we used to conduct our experiments based on the knowledge from this section.

26.3 Methodology

In this section we discuss the set up and the experiments conducted to achieve the results in Section 26.4. For this we use the knowledge from Section 26.2 to automatically generate signatures for 10 different attack vectors.

26.3.1 Signature Generation

Finding the characteristics of an attack vector was out of the scope for this research, instead we used fingerprints that were provided to us by DDoSDB [1]. They created an open source code that analyses a collected attack, generates a fingerprint and anonymizes the identity of the victim. We used json files associated to attack vectors that contain information about ips, ports, dns queries, icmp types etc. For replaying attacks we as well obtained the corresponding pcap files from DDoSDB. An overview of the attack vectors that we used for our research can be found in table 26.2.

ID	Protocol	# src IPs	# src ports	# dst ports	additional information
0	NTP	1288	1	11	
1	ICMP	6245	0	0	icmp type = 3
2	UDP	12	1	83	
3	ICMP	270	2821	1	icmp type = 11
4	DNS	30551	1	62591	dns query = diasp.org, dns type = 255
5	ICMP	244	12	1	icmp type == 3
6	UDP	12	1	96	
7	UDP	4	1	86	
8	ICMP	83	0	0	icmp type = 5
9	DNS	25027	1	60219	dns query = hoffmeister.be, dns type = 55

Table 26.2: Overview of the attack vectors used in this research

We created a python script that gets as input one json file from one attack, extracts all the relevant information from the json file and returns a set of rules that can detect packets from this attack vector. Because of limitations on the side of SNORT, we had to limit ourselves to rules with a maximum of 30000 characters while Suricata could not cope with large lists of IPs, so we decided to only use the source IPs if there were less than 100.

[1] https://ddosdb.org/

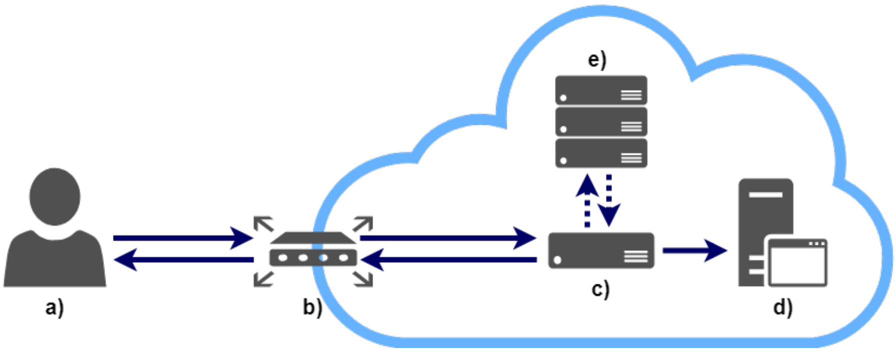


Figure 26.2: Overview of test setup

26.3.2 Test Setup

For conducting the experiments we created the test setup depicted in figure 26.2. This setup knows 5 actors. The first actor (a) is the attacker, that is directly connected to a router (b). This was done to have as less influence from outside as possible and also not hinder any other networks. The next actor is a switch (c) that allows port mirroring. Port mirroring means that we can mirror all data from a (set of) port(s) to one single port, where we will be running either SNORT or Suricata, but never both at the same time. This way we can inspect all the data to and from different machines in the network with SNORT or Suricata without interfering the traffic to the actual machines.
In this experiment we used only one machine (d) that got mirrored and acted as the victim of the attack from (a). Finally, (e) is a machine that runs SNORT or Suricata. This machine receives only all the *egress* packets to (d), which means all packets that are send out of the port to which (d) is connected since we are not interesting in what he sends back.

26.3.3 Experiments

In order to answer RQ2, we insert all attacks separately into our rule generator and measure how long it takes for each rule to generate a complete set of rules. Both SNORT and Suricata require us to write the set of rules to a file called *local.rules* and a restart of the IDS for the new rules to become active. To have a realistic view of how fast the IDS can adjust to a newly identified attack, we will also restart the IDS and measure the time it takes until the IDS is ready to inspect traffic with its new

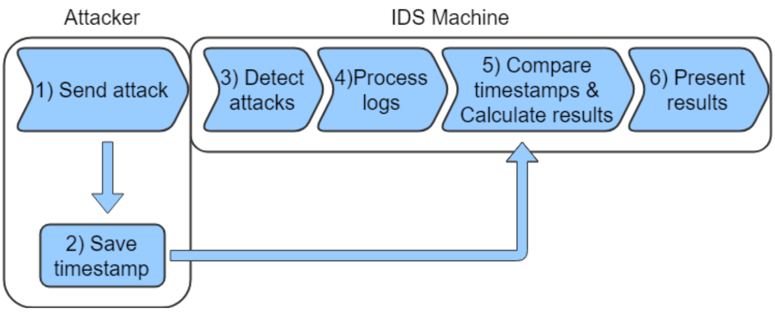


Figure 26.3: The process of the experiment

set of rules. We do this 100 times for every rule and present the average in section 26.4.

In order to then replay the attacks obtained from DDoSDB as pcap files, we first rewrite the destination IP and MAC addresses in the pcap file using tcprewrite [2]. We created a python script that, for every pcap file we have, executed the script *tcprewrite -dstipmap=0.0.0.0/0:[target_ip]/32 - enet-dmac=[target_mac] - infile=[attack_vector.pcap] -outfile=attack.pcap* .
To start suricata we use the command *sudo suricata -c /etc/suricata/suricata.yaml -s test.rules -i eno1*. For SNORT we used *sudo snort -A console -i eno1 -u snort -g snort -c /etc/snort/snort.conf*. Both commands allow us to write our alerts to log files that we can use to measure the time for the IDS to detect the attack. To launch an attack we used *sudo tcpreplay -i eno1 [attack_vector].pcap*. Here eno1 is the ethernet adapter of our machines. In figure 26.3 an overview of the process of this experiment is given, where launching the attack is represented in arrow (1).

Before we can do the measurements, we need to know the travel time of the packets and the time difference between the two machines. To find out this time we let the attacker send a message containing its current timestamp to the machine running the IDS, which compares it to its own current timestamp, do this 10000 times and take the average. We can then compare the timestamp of the first alert raised by the IDS to the timestamp at which the attack was sent, distract the travel time + time difference and find the time it takes for the IDS to identify the attack. We did this 10 times for every attack and both IDSs.

For both IDSs we save a timestamp at which the attack was sent on the attacking machine, seen in action (2). However, measuring the time at which the attack was

[2]http://tcpreplay.synfin.net/

detected differs for both IDSs since they log their alerts in different ways, detecting and logging is done in action (3). SNORT logs every *packet* it matches to an attack in a pcap file that is specific to a run of SNORT while Suricata logs every *alert* in a single file. Because the generated rules are not always specific to 1 attack but can also match on other attacks, we can not be sure what packets or alerts are exactly part of which attack.

To overcome this problem we wrote a python script that restarts SNORT for every attack so we get a pcap file corresponding to the log for a specific attack. Afterwards we use the command *sudo tshark -T json -r [attack_id].pcap -w [attack_id].json* to create a json file from which we can extract the timestamp at which the first packet was identified by SNORT, this is represented with action (4). Finally we can compare this timestamp in action (5) and find the difference between the moment the attack was sent and when the attack was identified by SNORT.

Suricata logs its alerts in a different way which requires us to change the approach slightly on the side of the IDS. Suricata logs every alert in a single file in ASCII format, so we can not use the same trick we used for SNORT. Instead we let the attacker send a dummy packet that gets picked up by a specific Suricata rule after every attack so we know when the next attack starts. An example of how the log file then look can be found in listing 26.4.

```
07/20/2018-16:30:10.12345 - Alert('Attack 1')
07/20/2018-16:30:10.23456 - Alert('Attack 1')
07/20/2018-16:30:10.34567 - Alert('Dummy for next rule')
07/20/2018-16:30:10.45678 - Alert('Attack 2')
07/20/2018-16:30:10.56789 - Alert('Attack 2')
```

Listing 26.4: Example Suricata log

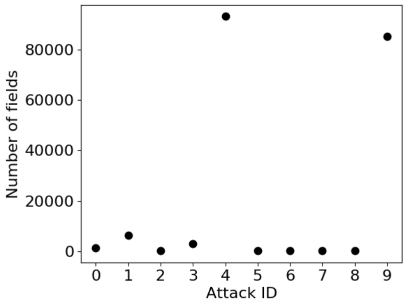
While SNORT logs in seconds since epoch format (amount of seconds passed since January 1, 1970 at 00:00 GMT), Suricata prints the data readable format as can be seen in listing 26.4. For this we first parse this to a python datetime object [3] that we can then transform to a timestamp in the same format as SNORT and compare it in the same way as SNORT.

We first tried to gather our results with a Dell M2800 running ubuntu 16.04 as our attacker, the D-Link DIR-605L as router, as switch the tp-link TL-SG105E, as target a Raspberry pi 2 and as our SNORT/Suricata machine the Raspberry pi 3 Model B. When replaying an attack at the found 80Mbps, the pi 3 could not cope up with the speed and started dropping packets, this also happened when we lowered the attackspeed to 1Mbps. This happened because the raspberry pi uses its CPU to

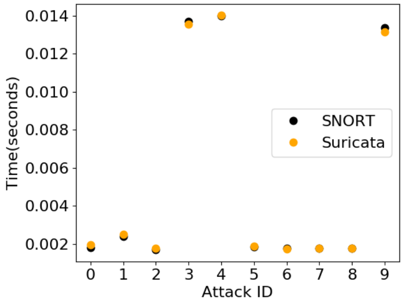
[3] https://docs.python.org/2/library/datetime.html

process incoming packets while already 60-80% of its CPU capacity was used by either SNORT or Suricata. From this we concluded that the Raspberry Pi 3 is not suitable to act as an IDS even at low replay speeds, so we had to change the setup and swapped the attacker and IDS around. Now the Pi 3 was the attacker while the Dell M2800 was now the machine running SNORT or Suricata. We again used iperf3 to measure the bandwitdh, which was the same as before.

26.4 Results & Discussion



(a) Number of fields



(b) Generation time

Figure 26.4: Number of fields per attack vector (left) and signature generation times (right)

In this section we discuss the results we obtained by executing the experiments from Section 26.3. The results for generating the rules and restarting SNORT/Suricata can be found in figure 26.4b. Here, the numbers correspond to the attacks from table 26.2 and the time is the time it took for generating a set of rules for an attack and restarting SNORT/Suricata to adjust to the new ruleset. As can be seen, there is practically no difference between the two different IDSs. However, there are some attacks that take significantly longer to process by the rule generator than others. These are attack 3,4 and 9. If we compare this to figure 26.4a, we see a high correlation between the amount of fields in the signature and the time needed to generate the signature. One exception here is attack 3. While attack 3 does not contain an exceptional high amount of fields its generation time is rather long. This is because looking back at table 26.2, while the attack does not contain more fields than other attacks, it does contain more source ports than other attacks. Where we drop the IP addresses if the number of IPs becomes too large, we do not do this for source ports, hence why the generation time of this vector is longer than the others. Finally we conclude that the time needed to generate a signature has a positive correlation with the amount of ports in the attack.

The second experiment was finding out the difference between the system times of the 2 machines and the travel time of a packet in the network. On average this time was 1.15s over 10000 runs.

The third and final experiment was finding the time it takes for both IDSs to detect an incoming attack. The results of the average time and standard deviation for this measurement can be found in figure 26.5. We can see that most attacks are detected within a second for both SNORT and Suricata. However, for attack 1 and 8 Suricata performed significantly worse than SNORT by being almost a second slower. More research needs to be done to find out why Suricata is more inconsistent than SNORT.

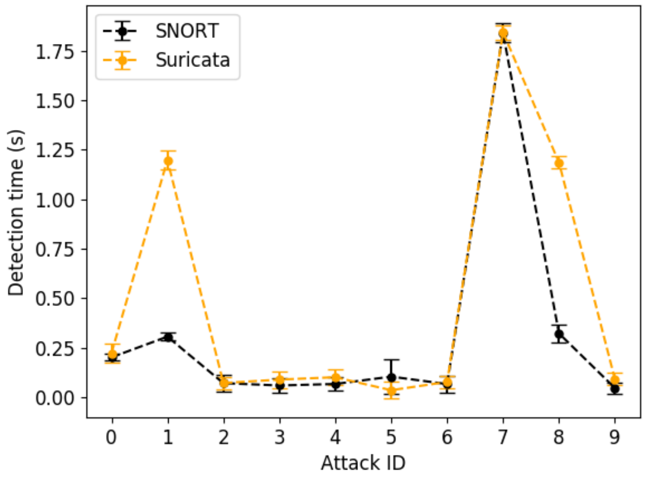


Figure 26.5: Detection times for SNORT and Suricata

26.5 Conclusion

In this paper we investigated whether SNORT and Suricata are suitable for a system where rules are automatically generated from given fingerprints. We showed that it is possible to find characteristics that can be used for generating rules within 15 milliseconds. However, some vectors were processed significantly slower than others, which is most likely caused by the amount of fields in the attack vector. Rules are also not always specific to one type of attack with our methodology for generating rules so further research can be done into optimizing the chosen fields to use for the signature and removing duplicate signatures.

We then showed the amount of time it takes for both SNORT and Suricata to identify incoming attacks. We show that in most cases this is done within 0.2 seconds. Both SNORT and Suricata showed 3 exceptions in our set of attacks where Suricata had some outliers while SNORT was more consistent. More research can be done into this to find out what causes both IDSs are slower on certain attacks and why Suricata is less consistent than SNORT.

In future work the accuracy of both IDSs and the rules should be tested. Right now only the time to detect an attack is investigated but it is interesting to know how much packets can we dropped with the IDS. Also more types of attacks can be tested for detection times. Furthermore more improvements to generating rules should be investigated. This research used a rather naive way to generate the rules so more improvements to mitigate the problem of slower generation for more fields and duplicate rules could be found.

26.6 Acknowledgements

We thank Jair Santanna for supplying the tools needed for the research, providing access to the signatures from DDoSDB, the valuable feedback and finally the amount of enthusiasm that has driven us during the research. Furthermore we thank René Boschma for providing help with setting up the experiments and discussing our work.

References

Akamai, state of the internet security report. (n.d.). https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q4-2017-state-of-the-internet-security-report.pdf. (Accessed: 2018-07-06)

Albin, E., & Rowe, N. C. (2012, March). A realistic experimental comparison of the suricata and snort intrusion-detection systems. In *2012 26th international conference on advanced information networking and applications workshops.*

J., P. (n.d.). *Character generator protocol.* https://www.incapsula.com/ddos/attack-glossary/dns-amplification.html. (Accessed: 2018-07-21)

Liao, H.-J., Lin, C.-H. R., Lin, Y.-C., & Tung, K.-Y. (2013). Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, *36*(1), 16 - 24. Retrieved from http://www.sciencedirect.com/science/article/pii/S1084804512001944 doi: https://doi.org/10.1016/j.jnca.2012.09.004

Open source ids: Snort or suricata? (n.d.). https://resources.infosecinstitute.com/open-source-ids-snort-suricata/#gref. (Accessed: 2018-07-06)

Santanna, J. J. (2017). *Ddos-as-a-service: Investigating booter websites* (Doctoral dissertation, University of Twente). doi: 10.3990/1.9789036544290

Security glossary: Top 12 ddos attack types you need to know. (n.d.). https://www.incapsula.com/blog/security-glossary-top-12-ddos-attack-types-need-know.html. (Accessed: 2018-07-06)

Szynkiewicz, P., & Kozakiewicz, A. (2017). Design and evaluation of a system for network threat signatures generation. *Journal of Computational Science*, *22*, 187 - 197. Retrieved from http://www.sciencedirect.com/science/article/pii/S1877750317305197 doi: https://doi.org/10.1016/j.jocs.2017.05.006

Understanding and configuring snort rules. (n.d.). https://blog.rapid7.com/2016/12/09/understanding-and-configuring-snort-rules/. (Accessed: 2018-07-06)

Vasilomanolakis, E., Srinivasa, S., Cordero, C. G., & Mühlhäuser, M. (2016, April). Multi-stage attack detection and signature generation with ics honeypots. In *Noms 2016 - 2016 ieee/ifip network operations and management symposium* (p. 1227-1232). doi: 10.1109/NOMS.2016.7502992

What is a ddos attack. (n.d.). https://www.verisign.com/en_US/

security-services/ddos-protection/what-is-a-ddos-attack/index.xhtml. (Accessed: 2018-04-22)

Why akamai. (n.d.). https://www.akamai.com/us/en/why-akamai/world-class-digital-experiences.jsp. (Accessed: 2018-07-21)