Note: I've used Adobe Acrobat Reader for annotating your text.

Automatic DDOS Attack Rule Generation Applied to Bro SIDS

René Boschma
University of Twente
r.boschma@student.utwente.nl

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1 Introduction

A Denial of Service (DoS) ~~attack~~ is an attack that aims to disable services of a target system. There are two main types of DoS attacks: *vulnerability* ~~*DoS*~~ and *flood* ~~*DoS*~~ [10]. In one hand, a vulnerability DoS aims to exploit a vulnerability of a target system to reduce its performance or render it useless. An example of such an attack is to send a malformed message to the target machine which can not deal with this message and as a result stops working. On the other hand, a flood DoS attack tries to exhaust the resources of the target. An example of such an attack is to fill the entire bandwidth of the target with messages of the attacker. The attacker can accomplish such bandwidth flood by using multiple machines to produce traffic. When multiple machines are used in the attack, it is called a Distributed Denial of Service (DDoS) attack.

DDoS attacks have increased in power and frequency. In 2011, the peak attack was measured at 60 Gb/s [12], in 2015, 500 Gb/s and in 2016 1.1 Tb/s [4]. In the third quartile of 2016 more than 5000 attacks where observed, whereas 200 in the entire 2012 [3]. As the number of attacks increase and downtime costs are exceeding on average $300K per hour [1] a need for an efficient and effective mitigation method has become crucial. The first task before the mitigation is the detection of an attack. Intrusion Detection Systems (IDSs) are such systems that can fulfill this task. An IDS ~~is a system that~~ monitors a system or network for malicious and/or suspicious activities. Based on the detection methods of IDSs, two categories can be identified: *Anomaly-based* and *Signature-based* [6]. An ~~Anomaly-based IDS (~~AIDS~~)~~ bases its detection on a constructed baseline and detects deviations from this baseline. A ~~Signature-based IDS (~~SIDS~~)~~ bases its detection on key characteristic of an attack for which predefined signatures are known. An AIDS has as benefit that it can detect unknown attacks but with the weaknesses that it has a low accuracy, needs time to learn a baseline of a system and has difficulties to trigger alerts before an attack scales up. A SIDS has as benefit that it has a high accuracy but with the weaknesses that it is ineffective in detecting unknown attacks and it is hard to maintain an up to date signature list [9].

Our hypothesis is that due to the high accuracy a SIDS is a suitable system that can fulfill the requirement of successfully and efficiently detecting DDoS attacks when the major downside of keeping an up to date signature list is tackled. The solution for this problem is to generate signatures for new attacks. This can be done either manually or automatically. As a manual approach requires significant amount of manual effort [10], we propose an automatic method. For this research we generate rules from extracted features of DDoS attacks for the Bro SIDS[1]. Bro is an open source network security monitor that offers the functionality of a SIDS. The features of DDoS attacks are extracted by a different research of DDoSDB[2].

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

- **RQ1:** What are the DDoS characteristics that could be used for generating BRO detection/mitigation rule?
- **RQ2:** What is the performance of automatic rule generation against a DDoS attack for the Bro SIDS?
- **RQ3:** What is the efficiency for Bro automatic generated rules when applied on an ongoing DDoS attack?

The first RQ will be answered by analyzing the most common DDoS attack vectors described by Akami [5]. The second RQ will be answered by building a proof of concept that generates signatures based on a given stream of features of DDoS attacks. The third and last RQ will be answered by replaying an attack for which a signature was generated and analyze what the performance of Bro is with these signatures implemented.

~~2 Content~~

2.~~1~~ DDoS Attacks and BRO

In this section, we will first define the notion of a DDoS attack by explaining the infrastructure. Then we elaborate on various DDoS attacks used ~~nowadays~~ and discuss their main characteristics. After this, we will discuss the syntax of the signature rules of the BRO SIDS.

2.1.~~1~~ The DDoS Attack

Figure 2.1.1 shows the infrastructure of a DDoS attack. Actors involved in an attack are denoted by a letter (A-~~D~~) whereas data streams are denoted by numbers (1-5).

[1] `https://www.bro.org/`
[2] `http://ddosdb.org/`

A DDoS attack starts with an attacker (A). The attacker sends data needed to start the attack (1) to the Command and Control (C&C) servers (B). The C&C servers control the infected machines (C). The infected machines are also known under the name of bots. The C&C servers plus the infected machines are more commonly known as a botnet. In case of the Ramnit botnet, only the infected machines counted 3.2 million machines [11]. When the C&C servers receive a message from the attacker, they at their turn send a message (2) to the infected machines. At this point, two paths are used to get to the target machine (E). The first path possible is aiming the infected machines directly to the target (4). This means that traffic from the infected machines will go directly towards the target. The second path possible is using public services (D). This means that traffic from the infected machines will first go through a public service, like a DNS, to reach the target (5).

In the next subsection, we describe various types of DDoS attacks and relate them to the architecture described above.

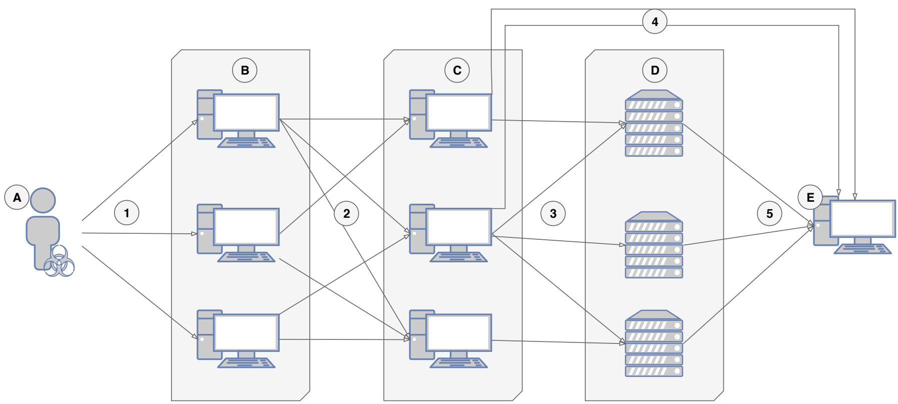


Figure 1: Overview of DDoS attack infrastructure

2.~~1.~~2 Types of Attacks

In this subsection, we briefly elaborate on ~~some of the most~~ common types of DDoS attacks mentioned in the security report of Akamai Q4 2017 [5]. On each attack, we discuss which protocol is used and which mechanism is exploited. An overview of the main characteristics per attack type can be found in Table 1.

UDP The UDP attack exploits UDP. The attack consists of sending a large number of packets to random ports of the target. Hereby the target machine will check if an application listens to this port and if not will reply with an ICMP Destination Unreachable (ICMP type 3) packet. ~~Looking at~~ Figure 2.1.1, path 4 is used.

UDP Fragment The UDP Fragmentation attack exploits the fragmentation used in the IP protocol [7]. When a packet is too big to be sent across a network link, it will be broken down into smaller packets and later on resembled again. In a UDP Fragmentation attack, UDP packets are sent which are larger than the maximum transmission unit (MTU) of the network thereby forcing fragmentation. This results in a higher ~~per~~ packet ~~bandwidth consumption~~ than a normal UDP attack. Looking at Figure 2.1.1, path 4 is used.

DNS The DNS attack exploits the public Domain Name System (DNS) services on the internet. DNS runs over UDP and is used to resolve IP addresses from website names. In a DNS attack, the attacker spoofs its IP replacing ~~it~~ with ~~that of~~ the target. The DNS server sees the request as it came from the target and replies as if it were a normal request. This way of attacking allows an attacker to amplify its own bandwidth by using the asymmetry between request and response size. Looking at Figure 2.1.1, path 5 is used.

NTP The NTP attack exploits the public Network Time Protocol (NTP) services on the internet. NTP runs over UDP and allows for time synchronization between machines. This attack is similar to the DNS attack, but this time the NTP services are used instead of the DNS services. Looking at Figure 2.1.1, path 5 is used.

Chargen The Chargen attack exploits the public Character Generator Protocol (Chargen) services. Once a Chargen service receives a packet via UDP, it responds by sending a datagram containing a random number between 0 and 512 characters long [8]. The attack approach works the same as the NTP and DNS. Looking at Figure 2.1.1, path 5 is used.

CLDAP The CLDAP attack exploits the Connection-less Lightweight Directory Access Protocol (CLDAP). CLDAP runs over UDP and is designed to provide access to directories while not needing the resource requirements of the Directory Access Protocol (DAP) [2]. The attack is similar to the DNS, NTP and Chargen attack. Looking at Figure 2.1.1, path 5 is used.

Attack Type	Main Characteristics
UDP Frag	IPv4 && fragments
DNS	UDP && src_port==53 && DNS_query && DNS_type
CLDAP	UDP && src_port==389
NTP	UDP && src_port==123
UDP	UDP
Chargen	UDP && src_port==19
SYN	TCP && flag==SYN
SSDP	UDP && src_port==1900
ACK	TCP && flag==ACK
HTTP	HTTP && HTTP_request && src_port==80

Table 1: Attack types main characteristic overview.

SYN The SYN attack exploits the three-way handshake of TCP. During this attack, a large number of SYN packets are sent to the target machine. The target machine will respond by sending a SYN-ACK packet. The attacker at this point does not respond by sending an ACK packet, leaving the connection half initialized. This way the attacker keeps connections from being used by legitimate users. Compared to the attacks mentioned above, this one runs over TCP rather than UDP. Looking at Figure 2.1.1, path 4 is used.

SSDP The SSDP attack exploits the Simple Service Discovery Protocol (SSDP). SSDP runs over UDP and is used to discover network services. The attack is similar to DNS, NTP and Chargen but when looking to Figure 2.1.1 the machines in D are for instance home routers, printers or other IOT devices rather than a dedicated server.

ACK The ACK attack exploits TCP. During this attack, a large number of ACK packets are sent towards the target. These ACKs do not belong to any connection and are therefore dropped at the target machine. This, however, does deplete resources of the target. Looking at Figure 2.1.1, path 4 is used.

HTTP Rather than the attacks mentions above, the HTTP attack targets the application layer. The attack consists of sending a large number of HTTP requests (like GET, PUSH and POST) to the target, hereby depleting its resources. Looking at Figure 2.1.1, path 4 is used.

2.~~1~~.3 Bro Rule Syntax

This subsection discusses the various parts used of the Bro ~~SIDS~~. Bro's primary focus is on its scripting language. With this language, one can define various analyzing scripts and detection policies. Besides the scripting language, Bro also offers also a signature language[3]. This language is similar to Snort rules and relies on low-level pattern matching. ~~In this paper, we will focus mainly on the Bro signature language to detect various types of DDoS attacks~~. In this subsection, first, an introduction to the signature language is given. Second, the signature language is applied to some of the attack types described in Section 2.1.2. Third and lastly, a brief explanation of the Bro scripting language is given.

The format of a signature is shown in Listing 1. As can be observed, each signature starts with the *signature* keyword followed by a signature identifier. The body contains attributes. These attributes can be rules to match on. Besides rules, it is also possible to specify in the attributes whether an event must be raised when all attribute rules within a signature are matched for a specific package. Both possibilities will be explained further.

```
signature [SIGNATURE-ID] {
  ATTRIBUTES
}
```

Listing 1: Bro Signature Format

An attribute rule follows the format *KEYWORD CMP VALUE*. Here the *KEYWORD* indicates a certain field to match, *CMP* indicates the way to compare and *VALUE* indicates the value to compare to. Various keywords are known to the Bro signature language. In this paper, we will only mention the ones used. *src-ip* and *dst-ip* specify the source and destination address respectively. Addresses can be IPv4 or IPv6. *src-port* and *dst-port* specify the source and destination port respectively. *ip-proto* specifies the protocol used. Values possible are: *tcp, udp, icmp, icmp6, ip* and *ip6*. It is interesting to note that Bro allows supplying a list of values in the format $v_1, \ldots, v_n$. Whenever one of those values matches a value within a received packet, the match of that rule will be evaluated to true. Furthermore, Bro allows ~~various~~ comparators such as $==, <=, >=, >, <$ and $!=$.

```
signature icmp-type-3 {
  ip-proto == icmp
  header icmp[0:1] == 3
  event "ICMP Type 3 Detected"
}
```

Listing 2: Bro signature which matches all ICMP type 3 packets

[3] `https://www.bro.org/sphinx/frameworks/signatures.html`

Besides matching on general conditions it is also possible to match specific bytes of a header. An example of this is given in ~~line 2~~ of Listing 2 where an attribute rule is written which matches all ICMP type 3 packets. As can be seen, the attribute rule starts with the keyword *header*. Then the protocol is specified (which can be any of the ip-proto values mentioned above). Then one defines within the square brackets the offset and size in bytes separated by a colon. In the example, the value of byte position 0 (which indicates the ICMP type) is matched against the value 3. It must be noted that not every protocol defines values that fall perfectly within the space of bytes such as the ICMP type. For instance the TCP flags. An example of a signature which matches all TCP SYN packets targeted at destination port 80 is given in Listing 3. This example shows that while the SYN flag's bit position in the header is 111, the offset specified in the header condition corresponds to bit position 104. This means that this signature will only match whenever the SYN flag is set to 1 and none of the other (including one reserved bit). This means that when one wants to write an attribute rule that matches whenever the SYN flag is set (and doesn't care about the values of the other flags), the rule has to contain 2^7 entries.

```
1 signature icmp-type-3 {
2   ip-proto == tcp
3   header tcp[13:1] == 2
4   event "TCP SYN Detected"
5 }
```

Listing 3: Bro signature which matches all TCP SYN packets targeted at port 80

Both Listing 2 and Listing 3 illustrate in line 4 the *event* keyword followed by a string. The Bro scripting language is event based. Scripts are defined to be executed whenever a certain event is triggered. One of the events defined in the language is the *signature_match* event. This event is triggered whenever a signature, which has an event defined in its body, is matched against a package. When this signature matches, the contents of the string is passed to the event listener, including various other parameters. For our experiment, we used one Bro script which can be found in Listing 4. The purpose of this script is to execute a python script whenever a signature match has occurred. The Bro script has some things to note. The first is the *@load-sigs ./sig* line. This line is responsible for loading the signatures files. The second is the *event signature_match(. . .)* line. This rule illustrates the event-driven programming script of Bro. Whenever a signature is matched, the contents defined within the curly brackets is executed. The *msg* parameter is given string specified after the event keyword in the signature.

```
@load base/utils/exec
@load-sigs ./sig

redef exit_only_after_terminate=T;
```

```

event signature_match(stage: signature_state, msg: string, data:
     string)
{
        local t= fmt("python3 send_to_attacker.py '%s'", msg);
        local cmd = Exec::Command($cmd=t);
        when (local res = Exec::run(cmd))
        {
                print msg;
        }
}
```

Listing 4: Bro script which executes whenever signature match occurs

~~2.2 Methodology~~

In this section, we discuss the experiment used to achieve the results presented and discussed in Section 2.3. First, we explain the test setup. Second, we discuss the data used for the experiment. Third and lastly, we discuss the tools and commands used to run the experiment.

For this research, we created a test setup which is ~~schematically shown~~ in Figure 2.2. ~~This setup knows 4 actors~~. The first actor (1) is the attacker. The second actor (2) is a router. The attacker is directly connected via cable to this router to n[illegible]nder any other networks. The third actor (3) is a switch that allows port mirroring. The fourth actor (4) is the target machine. The fifth and final actor (5) is ~~the Bro machine: the~~ machine that runs the Bro SIDS. The Bro machine receives all egress packets of the target machine.

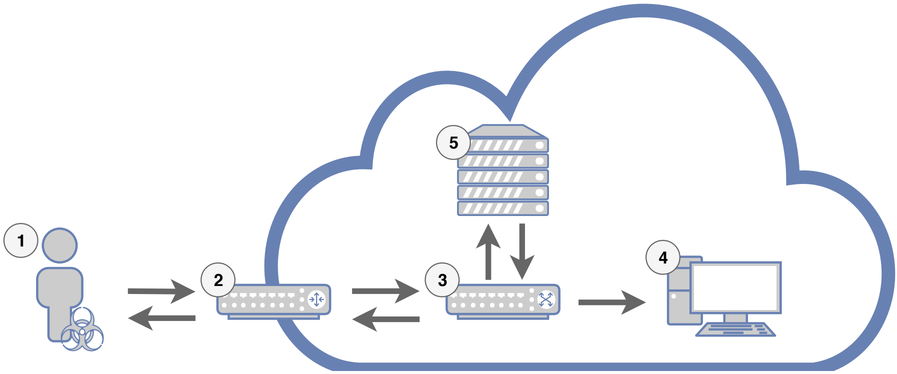


Figure 2: Schematic overview test setup

For this research, we choose 13 attack vectors retrieved from DDoSDB to test against our setup. Each vector has a pcap file which contains only the data belonging to ~~this~~ vector extracted from a real attack. From ~~this~~, DDoSDB ~~generated~~ a JSON file which describes the main characteristics of that vector. Each vector has been assigned a four-digit identifier to which we refer to in the remainder of this paper. A brief summary of each attack vector based on the received JSON file can be found in Table 2.

ID	Type	# src IPs	# src ports	# dst ports	special
e0b2	ICMP	83	0	0	icmp_type = 5
0292	TCP	6	1	1	tcp_flags = ··········S·
dd26	Chargen	439	1	5	
e6ee	DNS	25027	1	60219	dns_query = hoffmeister.be
54a7	ICMP	270	2821	1	icmp_type = 11
8219	UDP	12	1	96	
39b6	TCP	66610	41757	1	tcp_flags = ············
13c4	UDP	12	1	83	
c606	TCP	13452	12110	1	tcp_flags = ····CE····S·
7bf0	NTP	1288	1	11	
151e	ICMP	244	12	1	icmp_type = 3
9d61	ICMP	6245	0	0	icmp_type = 3
072a	DNS	30551	1	62591	dns_query = diasp.org

Table 2: Attack charasterics

A python script has been created that generates the signatures for the Bro SIDS. This script receives as input a JSON file received from DDoSDB and converts it to a valid Bro signature rule. Each attack vector has its own generated signature rule. For each vector the signature is generated on the Bro machine and is measured how long generation takes~~. The results of this can be found~~ in Figure 3.

To measure the response time of the Bro SIDS against an ongoing DDoS attack, first, the maximum bandwidth of the target and Bro machines are determined. This is done using iperf3[4]. The attacker runs the script: *iperf3 -s* and the other machine runs *iperf3 -c [attacker_ip] -R*. This way the server sends the data to the connecting machine. Second, the attacker will rewrite the destination IP and MAC-address of the supplied pcap using tcprewrite[5]. The command used is *tcprewrite –dstipmap=0.0.0.0/0:[target_ip]/32 –enet-dmac=[target_mac] –infile=[supplied_pcap] –outfile=attack.pcap*. Third, Bro is started in bare mode with the written Bro script described in Listing 4. This way only the needed files are loaded. The command used to start Bro is: *sudo bro -b -i [network_device] [path_script]*. Fourth, the attack against the target machine is launched using tcpreplay. The command used to launch the attack is: *sudo tcpreplay -i [network_device] –mbps=[speed_limit] attack.pcap*. Fifth and lastly, when a signature match is received, the Bro machine will send a message to the attacker. The times it took of each attack between launching it and receiving the message are displayed in Figure 4.

2.3 Evaluation and Discussion

In this section, we evaluate and discuss our findings we found by executing the methodology described in Section 2.2. We start by discussing the hardware used for the test setup and elaborate on each of the measured data.

In our first attempt at gathering results we used as the attacker a laptop with Intel i7-4700MQ @ 2.4 GHz with 8 GB of RAM running Ubuntu 18.04 LTS, as router the D-Link DIR-605L, as switch the tp-link TL-SG105E, as target machine a Raspberry Pi 2 and as Bro machine a Raspberry Pi 3 Model B. From iperf, we measured a maximum bandwidth of 90 Mbps. When replaying an attack vector at this rate, the Pi could not cope with it. We then decided to turn down the replay speed and concluded that even at a speed of 1 Mbps the Pi could not handle every attack. Therefore we concluded that a Pi, even for a signature set consisting of one signature, is not suitable to run the Bro SIDS.

[4] `https://iperf.fr/`

[5] `http://tcpreplay.synfin.net/`

We then switched to using the attacking laptop as Bro machine and as attacker a different laptop. We again used iperf3 to measure the bandwidth that could be accomplished. This measured 90 Mbps.

~~Several measurements were done during the experiment.~~ Every measurement is executed 10 times. The first measurement performed is the time required to generate the Bro rules from the supplied JSON signatures. The results are shown in Figure 3. This figure shows the time it took in seconds to generate the signature on the Bro machine for each attack vector. As can be observed, the vectors e6ee, 072a and 39b6 took considerable more time than the other vectors. When combining this data with Table 1 we can conclude that the generation time grows with the number of entries.

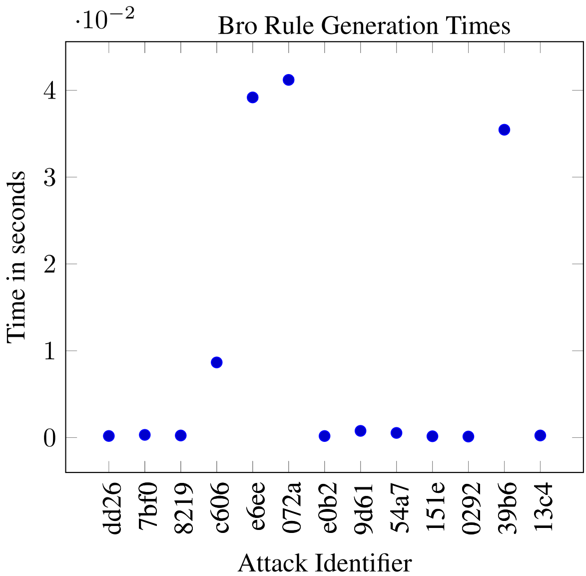


Figure 3: Bro Rule Generation Times

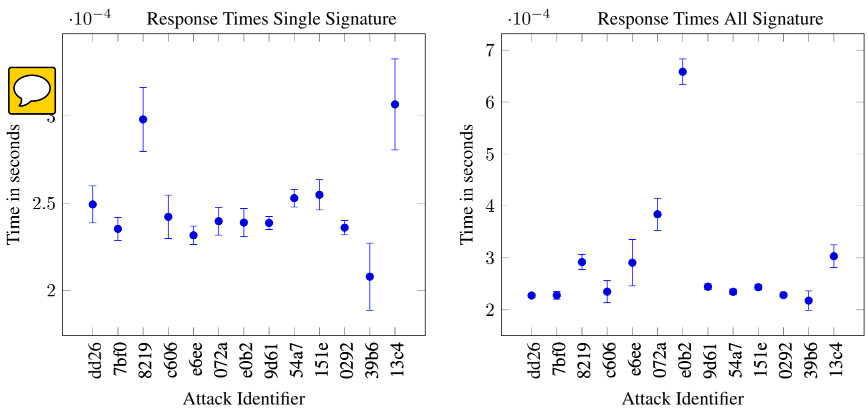
The second measurement performed is the time it takes to send a message from the Bro machine to the attacker. The time measurement includes setting up a TCP connection, sending a message and then closing the connection. The average of these measurements was 8.41E-5 seconds with a standard ~~error~~ of 1.29E-5 seconds.

The third and final measurement performed is the time it took for the Bro machine to react to an incoming attack. The results of these measurements are shown in Figure 4. This measurement is executed in two variants. The first variant is shown in Figure

4a and pictures the response times when every attack vector has its own signature loaded in the Bro machine. The second variant shown in Figure 4b pictures the response times when all attack vector's signatures are loaded.

In Figure 4a we observe that one vector, 39b6, takes less time than the other vectors. It is, however, visible that this measurement has quite a large variance compared to the other ones and might be that the reason for its speed is due to irregularities in the test setup. Furthermore, vectors 8219 and 13c4 take significantly longer to detect than the other attack vectors. It is interesting to note that both attack vectors are UDP attacks. One would more likely expect that the DNS attack vector would take longer to detect due to the fact that regex matching needs to be done against a packet's payload. Furthermore, the DNS attack vector e6ee has also more source IPs and destination ports to match against than both the UDP attacks.

In Figure 4b we observe that Bro does not increase significantly for all attack vectors. For most of the vectors, the times roughly stay the same. Most interesting are the increases in 072a and e0b2. Especially the latter as it increased by more than 2.5 times the time it took with only a single signature. ~~Our initial guess was that this~~ might be due to the order in which the rules were listed in the signature file. The order in the file was e0b2, 0292, dd26, e6ee, 54a7, 8219, 39b6, 13c4, c606, 7bf0, 151e, 9d61, 072a. From this, we conclude that the order is not what causes the increase in response time.



(a) Single Signatures

(b) All signatures

Figure 4: Graphs Showing response times after attack is launched

3 Conclusion

In this paper, we ~~try to show~~ whether Bro ~~is a suitable SIDS which~~ is capable of detecting attacks with automated generated rules based on the signatures retrieved from DDoSDB. We first show that it is possible to automatically generate Bro signature rules from a JSON file that describes the attack vector's main characteristics. Secondly, we show whether Bro is capable of detecting the attack vectors by executing an experiment that resembles a common internet architecture. In this experiment, we replayed various attack vectors targeting a target machine which port was mirrored to a machine running the Bro IDS.

From our experiments, we showed that ~~(1) it is possible to generate Bro signatures from various characteristics of attacks. (2) showed the generation time of signatures is reasonable and~~ (3) showed that Bro does not increase dramatically in response time when more signatures are added. However, we ~~do note that there are~~ some limitations to the automatic generation of signatures for Bro such as the byte addressing of the header fields.

For future work, ~~one could~~ investigate the accuracy of the automatically generated rules. ~~One could also~~ further increase the size of the signature list to determine the maximum number of signatures the Bro SIDS can handle. Lastly, it would be interesting to see if the more expressive Bro scripting language can compete with the signature framework in terms of efficiency and accuracy.

4 Acknowledgements

We would like to thank Jair Santanna for his valuable feedback, supplying the tools needed for the test setup and enabling us to gain access to the data stored in DDoSDB. Furthermore, We would like to thank Vincent Dunning for aiding in the process of creating the test setup and carrying out the experiments.

References

[1] Cost of hourly downtime soars: 81% of enterprises say it exceeds $300k on average. http://itic-corp.com/blog/2016/08/cost-of-hourly-downtime-soars-81-of-enterprises-say-it-exce Accessed: 2018-03-21.

[2] Young A. Connection-less Lightweight X.500 Directory Access Protocol. 1995.

[3] Akamai. State of the Internet/Security (Q3/2017). 2016.

[4] Akamai. State of the Internet/Security (Q1/2017). 2017.

[5] Akamai. State of the Internet/Security (Q4/2017). 2017.

[6] Alexandros G Fragkiadakis, Vasilios A Siris, Nikolaos E Petroulakis, and Apostolos P Traganitis. Anomaly-based intrusion detection of jamming attacks, local versus collaborative detection. *Wireless Communications and Mobile Computing*, 15(2):276–294, 2013.

[7] Imperva Incapsula. IP FRAGMENTATION ATTACK.

[8] Postel J. Character Generator Protocol. 1983.

[9] Hung Jen Liao, Chun Hung Richard Lin, Ying Chih Lin, and Kuang Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[10] Dong Lin. Network Intrusion Detection and Mitigation against Denial of Service Attack. *WPE-II Written Report*, (January):1–28, 2013.

[11] Europol Corporate Communications Lisanne Kosters. BOTNET TAKEN DOWN THROUGH INTERNATIONAL LAW ENFORCEMENT COOPERATION. 2015.

[12] Arbor Networks and Arbor Networks. Arbor Networks 9th Annual Worldwide Infrastructure Security Report. 2014.