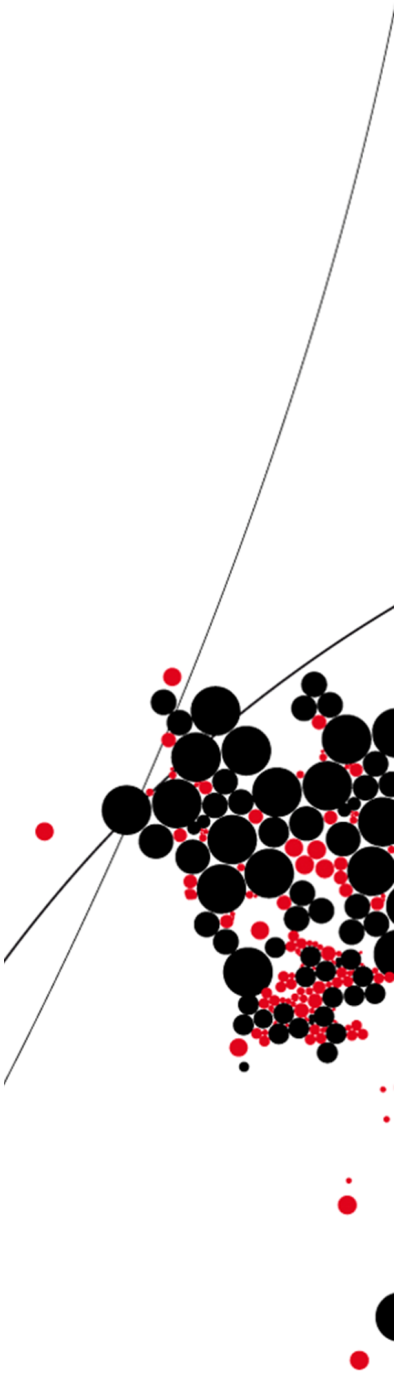
UNIVERSITY OF TWENTE.

Faculty of Electrical Engineering,
Mathematics & Computer Science



Payload Based Signature Generation for DDoS Attacks

Kareem M. I. A. Fouda

Master Thesis

August 2017

Examination Comittee:
Prof. dr. Anna Sperotto,
M.Sc. José Jair C. Santanna

Design and Analysis of Communication Systems Engineering Group
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Abstract

Distributed Denial of Service (DDoS) attacks present a serious threat to online organizations. These attacks can have a serious impact on organizations in terms of brand image and revenues. An efficient way to combat DDoS attacks is to store a signature for every attack. This way any attack can be detected as soon as this signature is found in traffic. However, the process is not that easy and this signature generation process is usually time and effort consuming. To help solve this signature generation problem, in this thesis, we propose an automatic methodology to generate payload based signatures for DDoS attacks. In doing so, this work sheds light on the different types of DDoS attacks and the techniques commonly used in performing DDoS attacks. It also inspects the relations between different payloads of the same attack. The research concludes by proposing a signature generation algorithm and validating it using practical tools.

Contents

List of Acronyms

A-IDS	Anomaly based Intrused Detection System
API	Application Programming Interface
Chargen	Character Generator
DDoS	Distributed Denial of Service
DNS	Domain Name Service
DRDoS	Distributed and Reflective Denial of Service
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IDWG	Intrusion Detection Working Group
IP	Internet Protocol
NTP	Network Time Protocol
QotD	Quote of the Day
RPC	Remote Procedure Call
S-IDS	Signature based Intrusion Detection System
SNMP	Simple Network Management Protocol
SSDP	Simple System Discovery Protocol
SVM	Support Vector Machine
SYN	Synchronize
TCP	Transmission Control Protocol

TCP/IP Transmission Control Protocol/ Internet Protocol

UDP User Datagram Protocol

UPnP Universal Plug and Play

Chapter 1

Introduction

Distributed Denial of Service (DDoS) is a coordinated attack that aims to degrade Internet services making them inaccessible to legitimate users. DDoS attacks reach their aim by flooding the services bandwidth or depleting their resources. The impact of DDoS attacks can be a major hit to online businesses revenue and brand. DDoS attacks can cause an average loss of over US $300K/hour [8] and can result in the loss of customers trust.

The impact of DDoS attacks have made it a major concern for businesses and an exploitable target for abusers; that is why attacks continue to evolve and become more complex and at the same time continuous advancements are made in mitigating DDoS attacks. One of the most prominent ways to combat DDoS attacks is the use of an Intrusion Detection System (IDS) that detects DDoS attacks as soon as possible minimizing their impact on the business.

IDSs are classified into two main classes based on their attack detection mechanism. The first is Signature based Intrusion Detection System (S-IDS) which detects attacks by comparing current network traffic to known attacks signatures; S-IDS provides an efficient way to detect known attacks however, it is useless against new (unknown) attacks. In addition S-IDS requires manual effort of experts to characterize new attacks and generate descriptive signatures for those new attacks. The second class of IDSs is Anomaly based Intrused Detection System (A-IDS) which works by having a profile of normal (attack-free) behavior of the network and a profile of the current behavior of the network. A-IDS operates by continuously comparing the current profile with the normal profile of the network and if the difference exceeds a certain threshold an attack alert is raised. Although, A-IDS provides a way to detect unknown attacks however; it suffers from false positives (legitimate traffic classified as attacks) due to the changing user behavior and the trade-off between the alert threshold, the false positive rate and the detection time.

A mixture of both IDSs can be deployed to act as a strong defense line against DDoS attacks, in this mixture A-IDS is used to detect a new attack and S-IDS is

used to detect old attacks. In the case of using the two classes of IDSs as a protection from DDoS attacks a new attack will be detected as a suspicious behavior by the A-IDS even if it is repeated multiple times which signals a missing link in the attack detection loop. This mechanism is because the suspicious behavior has no signature in the signatures database of the S-IDS so more traffic has to be monitored and compared to the normal profile of network usage. This detection operation slows down the detection of a DDoS attack and puts the network under more strain compared to that if the attack was detected through its signature. In addition for a signature to be generated for a specific attack; time and manual effort are needed to efficiently classify and characterize a new attack which would put the network under some risk till the signature is ready.

1.1 Research Objective

To be able to adapt quickly to the dynamic nature of DDoS attacks, an automatic signature generation system is required to be put in place to quickly and efficiently generate descriptive signatures of anomalous network traces. The work in this thesis proposes an efficient way of automatically generating payload based signatures that characterize new attacks without the need for manual work. The generated signatures are intended to be incorporated into the attack signature database of the signature based IDS allowing faster and more efficient detection of the same attack (if it occurs again). In this sense, this work aims to efficiently close the loop of detection providing faster and more robust detection of recurrent attacks.

1.2 Research Questions

In the pursuit of the research objective the following Research Questions (RQ) are defined as the basis of the research:

- **RQ1:** What types of attacks are likely to share a payload signature?

  Unlike packet headers packet payloads do not have a fixed format or expected pattern, a payload is just a sequence of bytes that can follow any order. This implies that there should be some criteria satisfied in inspected protocols and network applications in order to be able to get a meaningful signature by inspecting their payloads. This question aims to find these criteria and define suitable protocols.

- **RQ2:** How can signatures be extracted from attacks?

After answering RQ1, individual payloads and relations between different payloads are to be inspected so that relationships are identified and used to generate an attack signature that correctly represents a specific attack.

- **RQ3:** How representative are the extracted signatures for detecting attacks?

  After answering RQ2, an algorithm for signature generation is ready, in order to put the generated signatures into a practical use case, they are stored in the database of a commercially used signature based IDS and tested upon the original attack trace.

1.3 Contributions

The end goal of this work is to devise and implement an algorithm for automatic signature generation for payload based DDoS attacks. In the process of reaching this goal, the following contributions were made:

- A study of different types of DDoS attacks and their impact characterizing them into payload-based attacks or not.
- A comparison of different string similarity metrics for clustering attack packets.
- Two algorithms for signature generation are proposed, implemented and tested.

1.4 Thesis Structure

This thesis is divided into three parts. In ***Part 1*** we present the background information necessary to follow this research. In this part we explain DDoS attack types and IDS as a way of dealing with attacks. Part 1 ends with presenting the state of the art in the field of automatic signature generation. Part 1 answers **RQ1** and serves as a first step in our research.

In ***Part 2*** we explain the steps taken in generating a DDoS attack signature automatically. Implementation decisions made along the research are justified and different implementation options are explained. Part 2 answers **RQ2** and **RQ3**.

Finally, In ***Part 3*** we conclude the research done in this thesis highlighting the achieved results and contributions. Future work possibilities are also presented in part 3.

Part I

Background and Related Work

Chapter 2

Understanding Denial of Service Attacks

2.1 DoS versus DDoS Attacks

A Denial of Service (DoS) attack is an attack that aims to prevent legitimate users of a website from accessing its legitimate services; it does so by overloading the webservers of the targeted system consuming its bandwidth and/ or computing resources. This over consumption of resources either makes the website inaccessible or highly degrades its services.

The key difference between a DoS attack and a DDoS attack is the number of attackers and machines involved in the attack; in a DoS attack only one machine is used to perform the hostile activity while in a DDoS attack multiple machines that can reach millions are used to carry out the attack. Those machines are usually compromised machines of unaware users (called zombies) that are controlled by the original attacker to perform the attack.

A demonstration of DoS and DDoS attacks is shown in figure 2.1.

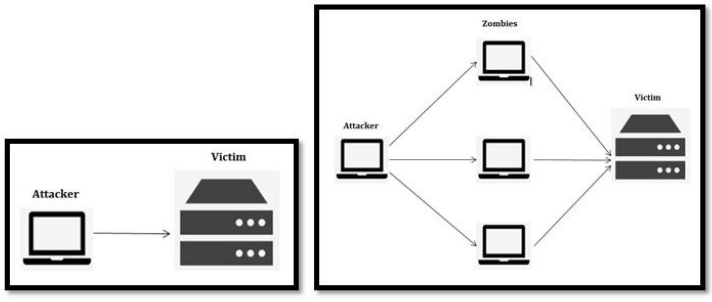


Figure 2.1: DoS Vs DDoS attacks

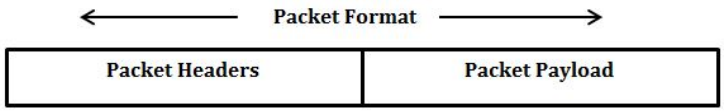


Figure 2.2: Packet Format

2.2 DoS Attacks Classification

In its simplest form DoS attacks are a repetition of traffic. Traffic is composed of packets where each packet follows the format in figure 2.2 in which a packet can be divided into headers part and payload part. The headers are for different layers within the Transmission Control Protocol/ Internet Protocol (TCP/IP) communication protocol suite [12] while, the payload part varies according to the application protocol used.

According to attack mechanism, DDos attacks can be classified into:

2.2.1 Bandwidth Depletion Attacks

These attacks aim to exhaust the network bandwidth available for the target uplink by sending a huge amount of small, average requests or by sending larger requests which are known as flooding attacks and amplification attacks. Bandwidth Depletion attacks can be carried out using:

Flood Attacks: In a flood attack, compromised zombies send a large amount of network traffic towards the victim congesting its network bandwidth which can have an impact reaching from slowing the victim servers to totally stopping them making the service inaccessible to the legitimate users. Common flood attacks have been launched using User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP).

Amplification Attacks: In amplification attacks, the amount of traffic directed towards the victim is a lot larger than that originally sent by the attacker (zombies) and hence, amplified. This type of attack can be done by having the zombies sending requests to broadcast addresses of routers which can cause all machines on that subnet to send replies to the victim. Common techniques used in this type of attacks are Internet Protocol (IP) spoofing and reflection which are explained in section 2.3.

2.2.2 Resource Depletion Attacks

These attacks aim to exhaust the server computing resources like computing power or operating system buffers and resources. This type of attacks usually exploits a

vulnerability of some protocols or operating systems. Resource Depletion Attacks can be carried out using:

Protocol Semantic Attack: In a protocol semantic attack a bug in the implementation of the protocol is misused to exhaust the server. An example of this type of attack is Transmission Control Protocol (TCP) Synchronize (SYN) attack that exploits the three-way handshake of the TCP leaving many semi-established TCP connections until connection buffers are depleted and legitimate users can no longer connect to the service. This specific attack can be solved using an approach called SYN Cookies that monitors a cookie value to know whether a client is a new client (in first step of handshake) or an older client (with a cookie value) and doesnt reserve buffers until the full handshake has occurred.

Malformed Packets: A malformed packet is a packet that uses a non-correct format for a sent packet to confuse the server and consume its time and resources in a useless trial to understand the packet. An example of this is packets with false combination of flags or wrong source and destination IPs.

2.3 Techniques Supporting (D)DoS Attacks

Some of the techniques that are commonly used in different types of DDoS attacks are:

2.3.1 IP Spoofing

This is the change of an IP address of a machine to resemble another IP that can be random, permutated or chosen from a subnet or a hitlist. In a DDoS attack when the source IPs (which is that of the zombies) are spoofed, it keeps the true identity of the attack machines hidden which complicates back-tracing the attack to reach the original attacker and also makes it difficult to recognize the IPs performing the attack to be blocked. On the other hand, source IPs can be spoofed to perform another technique called reflection.

2.3.2 Amplification

Amplification is an attack technique used to multiply the size and hence, the power of an attack without increasing the number of attackers. It relies on exploiting protocol features that provide a considerably larger response to a small request. The amplification factor can be defined as $\frac{length\ of\ response\ packet\ from\ server\ to\ victim}{length\ of\ request\ packet\ from\ attacker\ to\ server}$ Some of the common protocols used in amplification attacks are Domain Name Service (DNS)

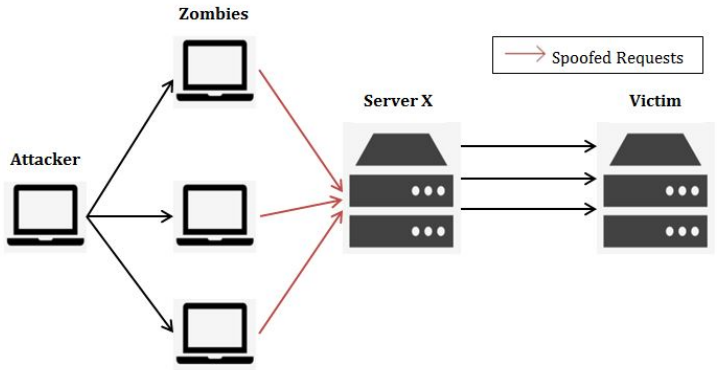


Figure 2.3: Reflection Technique

and Network Time Protocol (NTP) with amplification factors reaching hundreds of times.

2.3.3 Reflection

This is when the zombies send requests with the source IP spoofed to the victim IP so that all the responses are directed to the victim network that gets overwhelmed with responses for requests it did not send. This technique is very common in amplification attacks and can result in serious amplification factors. Nowadays, reflection using UDP services are more expected due to the connectionless nature of UDP and the high amplification factors of its services [27].

As shown in figure 2.3, the zombies are sending spoofed requests to Server X where the source IP is spoofed to be that of the victim, therefore, all the replies from server X are reflected to the victim. Server X can be any server that would abused to amplify the attack like a DNS Server or an NTP Server.

2.4 DDoS Attacks with Recurrent Payloads

Unlike packet headers payloads do not follow a fixed order or hierarchy that can be declared generic and analyzed for all protocols; instead payloads are sequences of characters that can follow different orders for different protocols. In order to be able to look into packet payloads and identify a meaningful signature that could efficiently represent the attack some conditions have to be met. In the following sections the decision criteria for judging an attack are presented then the most common DDoS attacks are discussed. Finally, the results and concluding remarks are given.

2.4.1 Decision Criteria

This work approach relies on pattern matching techniques to identify and generate a signature that accurately resembles an attack; for this to happen the attack traffic should have the following three main properties:

1. The attack should be payload based and not based on header values for example a certain combinations of flags.

2. All payloads that are part of the same attack should share a common attack parameter for example a DNS request in a DNS attack or a specific resource in an Hypertext Transfer Protocol (HTTP) flood attack.

3. Attack traffic should not be server dependent which means that for the same request similar responses should be returned from different servers.

2.4.2 Most Common DDoS Attacks

According to reports from top security companies [3] [4] [5] [7], DDoS attacks have been on the rise in the last three years. More protocols were abused in reflection and amplification attacks which are called Distributed and Reflective Denial of Service (DRDoS) attacks. DRDoS attacks are of the most impactful due to the high amplification rates achieved [27]. We focus in this section on the most common DDoS attacks and specifically DRDoS attacks due to their rising popularity and impact. Most common attacks can be classified into three main broad categories which are: UDP-based attacks, TCP-based attacks and multi-vector attacks that combine two or more types of attacks. Since the focus of this thesis is on single-vector attacks, we will focus on the first two categories.

2.4.3 UDP based Attacks

UDP based attacks employ UDP protocol as their transport protocol. UDP is one of the two most popular transport protocols used in the internet accompanied with TCP. It is a connectionless protocol that requires no authentication between the sender and receiver. This makes it a very good candidate for DDoS attacks especially for those attacks that employ IP spoofing and reflection. The most common class of UDP based attacks according to the mentioned reports is UDP floods.

2.4.4 UDP Floods

In a UDP flood attack a large volume of UDP packets are sent towards the victim server consuming its resources and degrading its service. UDP flood attacks can be carried out in one of two ways, the first is having a large network of botnets perform the attack and the second is through using amplification which leverages the botnets compromised and multiply the power of the attack. The most common protocols that are used in amplification are:

- **Domain Name Service DNS:** is used in the translation of host names into IP addresses. Requests of type Any produce the highest amplification factors as they respond with all known information about a DNS zone for a single request. One interesting observation about DNS attacks is that all the responses contain the requested website.

- **Network Time Protocol NTP:** is used to synchronize time over a network. NTP can be exploited in DDoS attacks mainly because of the ”monlist” command that responds with the latest 600 requests to that server and provides amplification factor that can reach hundreds or even thousands. It can be deduced that the different servers will issue different responses depending on their latest requests.

- **Simple Network Management Protocol (SNMP):** is a protocol used for collecting information and configuring network devices. SNMP most interesting service (from DRDoS perspective) is the GetBulk which responds with a list of SNMP identifiers that can be monitored. Responses for this type of attack are expected to depend on the network queried.

- **Simple System Discovery Protocol (SSDP)**: is a protocol that provides discovery for the Universal Plug and Play services running on the network. There are different types of requests however, the most notable one in our case is the discovery request because it provides higher amplification rates. The discovery request is supposed to be sent by a new device joining the network to know what kind of Universal Plug and Play (UPnP) services are supported on that network. Every device on the network that is running UPnP services responds with a unicast message to the client for every UPnP service it supports. Responses for this type of attack are expected to depend on the network queried.

Among the other less common UDP amplification candidates are:

- **NetBIOS:** is a networkingApplication Programming Interface (API) that enables different applications on different machines on the same network to communicate. It provides different services, however, the most notable one due to its amplification factor is the name service for which the receiving system responds with its NetBIOS host name and configuration.
- **Portmap**: is a service that maps Remote Procedure Calls to their correct services/ports. Remote Procedure Call (RPC) processes notify portmap when they start with the ports they are monitoring and the RPC program numbers they expect to serve. The client then contacts portmap service sending its RPC program number and then portmap replies with the correct port for the request RPC.
- **Character Generator (Chargen):** is a service that responds with random characters to received UDP datagrams of any length.
- **Quote of the Day (QotD):** is a protocol that sends a quote when receiving a request.

2.4.5 TCP based Attacks

TCP based attacks are the attacks that employ TCP as their transport protocol. TCP is a connection oriented protocol' it uses a three-way handshake to authenticate the server and a client. Its specifications make it a reliable protocol for critical services, however, it is still abused in DDoS attacks. The most common class of TCP-based attack according to the mentioned reports is the TCP SYN attack.

2.4.6 TCP SYN Floods

The second class of common DDoS attacks uses TCP. It exploits the mechanism of TCP handshake to open many one sided connections and consume the servers resources. The TCP SYN attack occurs on the TCP header level and might not carry any payload.

2.4.7 Less Common TCP based Attacks

Among other less common TCP based attacks are:

- **HTTP GET**: is considered an HTTP flood attack in which the attacker sends many HTTP GET requests trying to overwhelm the server. It is considered a simple attack that can scale well with the bots network.

- **HTTP POST**: is another type of HTTP flood attacks where many HTTP POST requests are sent to the server aiming to trigger as many processes as possible on the server to consume its resources. Since POST attacks usually trigger more complex operations on the server side like data retrieval compared to GET requests, HTTP POST attacks usually impact the server with less volume compared to HTTP POST attacks.

- **Slowloris**: A different type of HTTP attacks that aims to consume the server's resources. It achieves this by keeping as many connections open with the server as possible. Frequently, an HTTP header is sent to ensure the connection stays open while replicating the process on many connections. It is comparable with the TCP SYN attack but on the application level.

2.4.8 Concluding Remarks

In this section, the most common DDoS attacks were investigated. The goal of this investigation was to determine which attacks have similar recurrent payloads and can be good candidates for signature generation. The results are presented in table 2.1.

Protocol	Suitable?	Reason
DNS	Yes	All responses carry the DNS request
NTP	No	Monlist response is server dependent
SNMP	No	Response depends on queried network
SSDP	No	Response depends on queried network
NetBIOS	No	Response depends on queried network
Portmap	No	Response is server dependent
Chargen	Yes	Responses are repeated sequences of characters of different lengths
QoTD	NA	QoTD is an old protocol and is rarely used in attacks now
TCP SYN	No	It is a header based attack

Table 2.1: DDoS Attacks Inspection

The results show that from the investigated attacks, DNS and Chargen lend themselves well to payload based signature generation. For this reason, the experimental part would mainly focus on DNS and Chargen attacks. However, we expect the devised algorithms to work on other protocols that follow the criteria mentioned in 2.4.1. It is expected that HTTP GET and HTTP POST attacks also can be suitbale for payload inspection if the same resource or the same POST request body are used in a large portion of the attack traffic, however, they were not investigated. It is also reported in Akamai state of the internet security report [1] that DNS and

Chargen attacks represented more than 30% of DDoS attacks in the first quarter of 2017.

Chapter 3

Intrusion Detection Systems

Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices. Incidents have many causes, such as malware (e.g., worms, spyware), attackers gaining unauthorized access to systems from the Internet, and authorized users of systems who misuse their privileges or attempt to gain additional privileges for which they are not authorized. [28]

3.1 IDS Architecture

A general architecture for Intrusion Detection Systems was proposed by the Intrusion Detection Working Group (IDWG) consisting of 4 components [13]:

- E-components (Event Boxes): is the first component of an IDS that is responsible for monitoring the target system and thus gathering events information to be analyzed by other blocks.

- D-components (Database Boxes): These are the database blocks used to store data gathered by the E-blocks for further analysis.

- A-components (Analysis Boxes): Those are the processing components responsible for looking into and classifying the already gathered events identifying suspicious activities for having the correct response.

- R-components (Response Boxes): This component is the final component of the IDS architecture and it has the role of taking the suitable action in case of having a security breach.

The proposed architecture is shown in figure 3.1

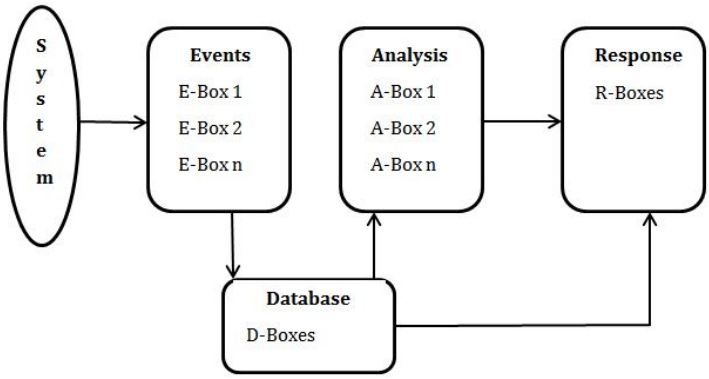


Figure 3.1: IDS Architecture

3.2 IDS Classification

Following the IDS architecture shown in 3.1, according to the location of deployment of the event boxes; intrusion detection systems can be classified into:

3.2.1 Host Based IDS

Host based IDS is deployed on the host level; it monitors and analyzes events such as process identifies, system calls and individual traffic information for a particular host. It has the advantages of more flexibility and control per host but can get complex to manage for a large number of hosts. It is suitable for individuals and personal use.

3.2.2 Network Based IDS

Network based IDS is deployed on the network level; it monitors and analyzes network traffic information such as traffic volume, IP addresses and service ports for a particular network. It has the advantages of easier management, no overhead over individual machines. It is more suitable for companies, schools and private networks with many users. The work in this thesis is concerned with this type of IDS.

According to detection mechanisms used in the analysis boxes, there are two main classes of intrusion detection systems which are:

3.2.3 Signature Based IDS

It is also known as misuse detection system. This system relies on looking for certain identified exploit patterns in incoming traffic; those identified attack patterns

can be any combination of packet features, flags or conditions and are called attack signatures and are saved in the S-IDS database.

This class of IDS has the advantages of quickly identifying attacks and having a low false positive rate; however, it behaves poorly when a new attack (one that has no signature is encountered). S-IDSs usually use pattern matching techniques to recognize attack signatures. Some of the most popular signature based IDS are Snor [26] and Bro [23].

3.2.4 Anomaly Based IDS

It is also known as Behavior Based IDS. This system relies on being able to detect suspicious behavior of the system traffic. This is done by having a profile of expected legitimate traffic of the system as training input and forming a model for this normal behavior; then the incoming traffic for the system is constantly compared to the already established normal model and if found too distant, an alert is raised.

This class of IDS has the advantages of detecting new unknown attacks but this comes on the expense of having high false positive rate. A-IDS is also considered more complex than S-IDS in terms of used techniques, processing and deployment. A-IDSs use a variety of techniques to generate the normal traffic model like statistical analysis, knowledge base and machine learning [13]. Some examples of popular A-IDSs are PAYL [31] and Anagram [32].

Chapter 4

Related Work

Research has been active in the field of internet security and DDoS attacks since the rise of the internet due to the serious impact of cyber-crimes on both persons and businesses. The work in this thesis can be related to two subfields in the DDoS research realm which are research that inspects packet payload for various goals like anomaly detection or traffic classification which is referred to in this work as *payload inspection subfield*. The other subfield is that dealing with signature generation of a (D)DoS attack which is referred to as *signature generation subfield*.

4.1 Payload Inspection Sub-field

Wang and Stolfo [31] propose an anomaly based intrusion detection system based on packet payloads. It analyzes normal payloads expected for the monitored network application through calculating their byte frequencies. Using the byte frequency distribution a model is then created for the service normal behavior and the anomaly detection phase starts in which incoming payloads are tested for the similarity with the already established normal profile, if a payload is found very different from the normal profile an alert is generated and further actions are taken.

Wang et al. [32] progress over PAYL and proposes the use of higher order n-grams ($n > 1$) to detect malicious events and generate malicious signatures. Anagram relies on storing distinct n-grams observed in normal legitimate traffic in a bloom filter, at the same time, distinct n-grams from known malicious signatures are also stored. This stored data acts as training input for the system, later at detection time, the packet is rated according to the number of unobserved n-grams and the number of malicious n-grams observed.

Nwanze and Summerville [22] propose an approach to detect malicious packets based on payload inspection in which they rely on modeling packets on per service basis taking advantage of the predictable byte frequencies shared between pack-

ets of the same service. The detection happens through comparing the n-gram histogram of incoming packets to the normal histogram of the relevant service.

Other research like citepli2008classifying uses payloads in addition to packet headers to determine and classify the uses of the well-known HTTP protocol.

4.2 Signature Generation Sub-field

The other research subfield under study is concerned with the signature generation of malicious attack whether those attacks are worms, DoS or another type of network attacks.

Thakar et al. [30] propose a semi-automated approach to monitor attacks and generate attack signatures, the method identifies three components: data logging, data analysis and signature extraction where attack data is logged using honeypot framework and classified into suspicious or legitimate based on Support Vector Machine (SVM) [11] taking into account traffic features like source/ destination IP and port and packet content, finally, an attack signature is extracted using Longest Common Substring algorithm [14].

Katkar and Bhirud [16] present an approach to detect new (D)DoS resource consumption attacks and generate a signature for them. The proposed approach utilizes the use of an anomaly detection filter and stores all suspicious traffic in a database. The server is continuously pinged to test if it is under attack or not. When the server is under attack suspicious traffic is analyzed using one pass incremental apriori algorithm [17] to relate traffic and find most frequent traffic based on 9 features including duration, dst bytes, src bytes, service, flag, and others, then those features are used as signatures.

Hwang et al. [15] propose the use of S-IDS in combination with a custom-designed A-IDS and introduces a weighted signature generation method for decting DoS attacks in addition to other types of network attacks. Snort is used as the S-IDS and the designed A-IDS deduces frequent episode rules from frequent events over a time window characterizing them with different parameters. In the final stage, A-IDS captured traffic episodes are analyzed according to their anomaly score and normality score (given by the A-IDS) and signature generation is carried out.

Perdisci et al. [24] present a behavioral malware clustering system targeting malicious HTTP traffic. The proposed approach aims to enable better and more efficient automatic signature generation for HTTP based malware. The basic idea of the presented approach is to monitor the behavior of different HTTP based malware and group different malwares according to their behavior; this behavior can then be modelled to monitor and detect malwares. Clustering is done in an incremental manner

that starts with finding more generic similarities between different malwares like the number of HTTP requests, number of GET/ POST requests then more granularity is applied to further split obtained clusters and finally cluster merging is done to combine very similar clusters to obtain a more generic view.

Kreibich and Crowcroft [19] propose a signature generation mechanism for malicious network traffic captured on a honeypot system (a monitored system deployed on the internet with the intent to lure hackers and malicious traffic for further studies). Inspection is done on both header levels to detect matching connections and on payload level to detect matching patterns. Signature generation is done using Longest Common Substring in two manners which are horizontal detection that matches the nth packet from multiple different connections that target the same destination port and vertical detection that matches maximum bytes of concatenated messages of two different individual connections. If a matching pattern that exceeds a configurable length is found, it is added to the signature pool. This signature pool is continuously monitored to check if one signature is a subset of another to see if the number of generated signatures can be minimized. Generated signatures are then converted to Snort and Bro formats to facilitate later use.

Mohammed et al. [20] focus on automatic signature generation for zero day polymorphic worms that change their payload in every infection attempt. It does so by trapping suspicious traffic obtained through trying to access private servers between two honeynets where attempts to create outbound connections are considered malicious worm-generated traffic and is redirected from one honeynet to the other honeynet and vice versa and thus obtaining various payloads of the same worm which facilitates the signature generation process. For the signature generation variations of the longest common substring algorithm are used.

Finally, Portokalidis et al. [25] present an emulator to generate signatures for exploit zero-day attacks without much focus on packet payloads.

Since signature generation is of key relevance to the work of this thesis, a summary of the key points in the different signature generation methods is presented in table 4.1. A survey of different signature generation techniques is given in [18].

Work Reference	Automation Level	Designed For	Traffic Classification Method	Signature Generation Method
[30]	Semi-Automatic	Intrusion attacks on web services	SVM semi-supervised classifier	Longest common Substring (LCS)
[16]	Automatic	(D)DoS resource consumption attacks	Server live ping	One pass incremental apriori algorithm
[15]	Automatic	Network attacks	Customized A-IDS	Weighted frequent items generated
[24]	Automatic	Clustering HTTP malicious traffic	Incremental feature based clustering	NA
[19]	Automatic	Malicious network traffic	Honeypots	LCS
[20]	Automatic	Polymorphic worms	Honeynets	LCS variations
[25]	Automatic (emulator-based)	Zero-day attacks	Dynamic Taint Analysis [29]	LCS and critical exploit string detection

Table 4.1: Signature generation summary

Following the criteria mentioned in table 4.1, our proposed work could be categorized as automatic signature generation for payload based (D)DoS attacks using

minimum shared signature as a traffic classification method and employing longest common substring [14] for signature generation. Our approach differs from previous work in that it treats packet payloads as strings and clusters them based on their string similarity properties. It also proposes ways to minimize the processing speed and power by considering a carefully selected subset of the attack traffic while still maintaining an accurate representation of the whole attack.

Part II

Signature Generation

Chapter 5

Signature Generation

A signature is a sequence of characters that efficiently identify an attack; it is large enough to encompass different representations of a particular attack and small enough to not be ubiquitous in normal traffic.

The problem of automatic payload based signature generation deals with string processing techniques, it aims to find a string pattern that characterizes malicious attack traffic whether that malicious traffic is a worm, a DoS attack or another type of network attack. In this sense the input data can be processed as one of three formats which are

- Binary Format: This is the most basic data format; it has the advantages of being the fastest format to obtain as it is the data format transferred on communication links and having the least amount of possible values (0 and 1). On the other hand it has the biggest amount of data as every byte is represented in 8 binary bits and consequently, the longest time to process whether to generate a signature or to match a signature.

- Hex Format: This is preferred over the binary format for having less amount of data where every byte is represented in two hex characters and is still fast to get and operate on with only 16 possible values.

- ASCII Format: This format takes the most time to get and is susceptible to losing some information as not all byte combinations can be transferred to meaningful ASCII characters.

The Hex format is chosen to be the primary data format for this work. This is due to its mentioned characteristics and that it lends itself well for packet payload processing.

The main hypothesis in this research is that DoS attacks are considered as an abnormal behavior for the victim network. Also attacks involve a large number of

repetitions for a similar type of packet for example if the attack is a DNS reflection attack then there would be a dominance of DNS packets in the time of the attack and if the attack is an NTP reflection attack then there would be a considerable larger amount of NTP packets at the time of the attack and if its an HTTP GET attack then more than usual HTTP GET request are going to occur during the attack and so on. This unusual distribution of traffic would trigger the A-IDS that there is a malicious activity possibly taking place on the network and it will start storing data including suspicious packets and their source IPs. This stored malicious activity is the input to our signature generation subsystem.

In figure 5.1. An overview of the system deployment is depicted; an example of the general attack scenario is given where the role of each part of the system is explained to give a better understanding of the system dynamics.

The generic scenario goes like this:

1. Malicious traffic is sent to the server and detected by the A-IDS.
2. The A-IDS stores the malicious traffic.
3. Signature Generator reads the attack traffic and generates a signature.
4. The generated signature is then added to the S-IDS attack signature database.
5. The S-IDS is able to filter out the attack traffic if it is still going on or stop the attack if it occurs again. This is how the detection loop is closed.

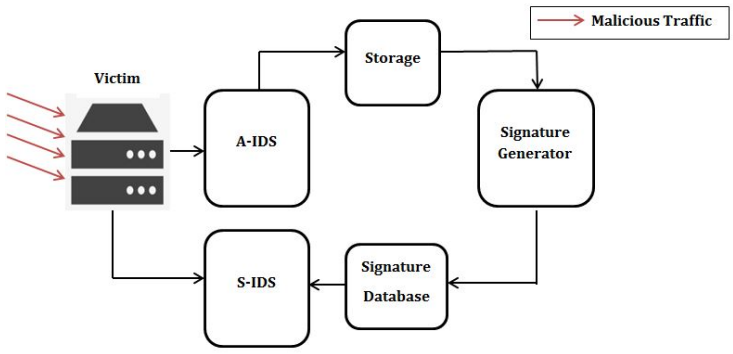


Figure 5.1: General System Architecture

In the upcoming sections two algorithms for signature generation are proposed. Firstly, the generic tasks that both algorithms perform are explained. Secondly, a more in depth explanation for those different tasks and the different possible ways to accomplish a certain task are investigated to reach the most suitable solution for this specific problem. Finally, the proposed algorithms are presented and validated.

5.1 General Approach

To be able to generate a signature, our research identifies major tasks that should be accomplished. Those tasks are:

1. *Attack Filtering*

   A network attack can last for minutes and even hours and can contain millions of packets. In order to be able to work on this enormous amount of data in a timely and space-efficient manner a filtering mechanism should be employed that enable handling a fraction of the trace data while still getting accurate data that represent the majority of the trace. A high level demonstration of attack filtering step is shown in figure 5.2.

Figure 5.2: Attack filtering demo

2. *Payloads Classification*

   After attack filtering, only a fraction of the trace is going to remain; this remaining fraction is supposed to have a concise representation of the original trace. The next step is to be able to classify payloads according to their shared features and have a set of unique patterns grouped together that represent the different patterns in the trace. Related patterns are simply patterns that share a similarity score higher than certain threshold; methods to obtain this similarity are discussed in the following sections. A high level demonstration of payloads classification step is shown in figure 5.3.

Figure 5.3: Payloads classification demo

3. *Signature Generation*

   The next step is to generate a signature for the patterns obtained in step 2, those signatures are now supposed to represent all key patterns from the original trace. To settle on a final signature that could isolate the most dominant traffic in the trace which is the attack, all obtained signatures are matched

against the original trace and the signature that matches the most payloads of the original trace is considered the final signature. A high level demonstration of signature generation step is shown in figure 5.4.

Figure 5.4: Signature generation demo

4. *Use Case*

   The final step is to integrate the generated signatures in a signature based intrusion detection system simulating a practical use case. A high level demonstration of the use case is shown in figure 5.5.

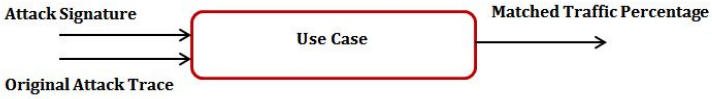


Figure 5.5: Use case demo

5.2 Attack Filtering

At this phase the input is an attack trace that contains a large number of packets. The goal of this phase is to minimize the number of packets that have to be processed. This is done by extracting the key payloads from the trace that represent ideally all or the majority of the variable packets that belong to the attack.

In order to do this we create a hypothesis that is based on the simple definition of a DoS attack that it is a flood of traffic and validate this hypothesis with experiments. Our hypothesis is two-fold and can be summarized as:

1. A smaller percent of source IPs send a large percentage of the attack traffic and their traffic can efficiently represent the attack.

2. Most frequent payloads represent a large percentage of the attack traffic and those payloads can efficiently represent the attack.

Test Setup

To verify our claims an experiment was run on 14 DNS attacks and 12 chargen attacks all having multiple source IPs provided from ddosdb [2]. For the source IPs

approach, the results are represented in a percentage notion in which the percentage of traffic is plotted against the percentage of the source IPs sending this traffic. On the other hand, for the most frequent payloads approach, the most frequent payloads are represented as constant numbers as the frequency of each payload will change in different attacks so a constant representation is more suitable in this case; those frequent payloads are plotted against the percentage of the attack traffic they represent.

5.2.1 Source IPs Approach

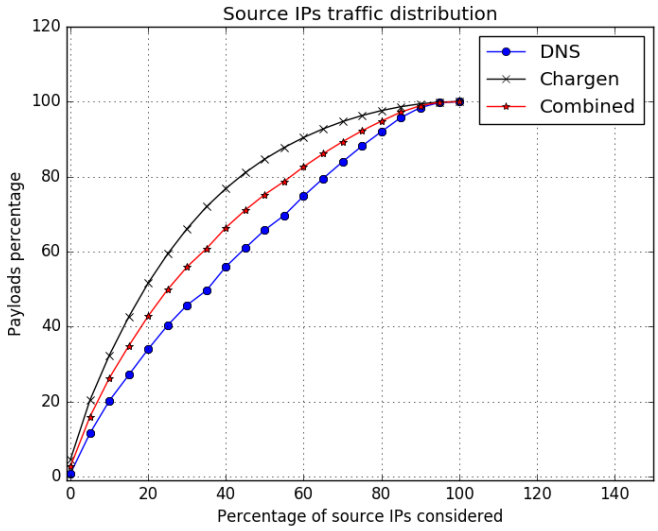


Figure 5.6: Source IPs traffic distribution

From figure 5.6 it can be seen that almost 60% of a DNS attack is sent by 40% of the participating IPs. The traffic percentage gets higher for a Chargen attack in which more than 75% of the attack traffic is sent by 40% of the participating IPs. These observations support our first claim that a smaller range of source IPs send the majority of the traffic during an attack.

5.2.2 Most Frequent Payloads Approach

From figure 5.7 it can be seen that around 30% of a DNS attack is represented by only the 100 most frequent payloads and this percent keeps increasing as more frequent payloads are considered. The percentage is even higher for a chargen attack where around 70% of an attack traffic is only from the most frequent 100 payloads. This support the claim that for DoS attacks the most frequent payloads represent a considerable amount of the traffic.

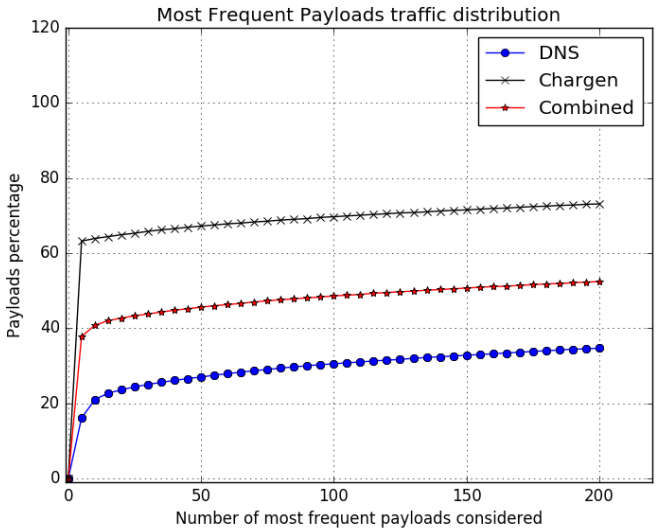


Figure 5.7: Most frequent payloads traffic distribution

5.2.3 Concluding Remarks

In this section, Two approaches were found to minimize the amount of traffic that needs to get analyzed and still get a representation of a large percentage of an attack. It was found that by considereing only the traffic from less than 50% of the source IPs of an attack trace, more than 70% of an attack can be covered on average. Also, it was found that by considering only as low as 200 payloads from an attack, more than 50% of an attack can be covered on average. Through using those findings, processing power and time can be greatly reduced without sacrificing the signature accuracy. However, the question now is what is the exact numbers do those percentages and numbers need to be to extract a correct signature for an attack. This question is further investigated in section 5.4.

5.3 Payloads Classification

This is the phase in which we have a large amount of payloads that possibly represent more than one type of traffic. This traffic can include multiple types of attacks or one type of attack with more than one attack variant or normal and attack traffic. In order to be able to relate payloads, a measure of similarity is chosen to deduce whether two payloads are similar and consequently, part of the same type of traffic (attack) or unrelated where they are of different types of traffic. A simplified example of the expected input is given in figure 5.8. It is followed by an explanation of what the goal of the payload classification task is.

In figure 5.8 every line represents a different payload, the green sequence is the

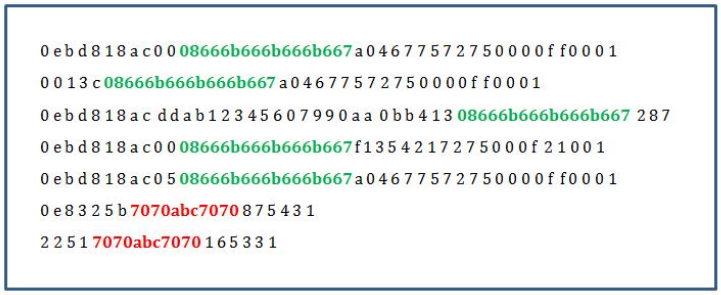


Figure 5.8: Example input to payloads classification

attack signature we aim to automatically generate while the red sequence is another signature but with much lower frequency. In such simple scenario, the trace has two unique patterns one in which the green sequence appears and the other is the one in which the red sequence appears.

The goal of payload classification task is to be able to extract those unique patterns in which the payloads representing each unique pattern are considered similar and dissimilar to those of any other pattern with as high accuracy as possible. Similar payloads are supposed to be variations of the same attack which share the same signature.

5.3.1 Similarity Based Classification

Similarity based classification treats payloads as strings and investigates methods to relate those payloads based on the similarity of their structure and content. The goal of this classification is to cluster related payloads that are part of an attack with the same attack variant differently from other traffic.

A string similarity metric is a function that operates on string sequences and character formulations. It measure similarity or dissimilarity (distance) between two objects facilitating an easier estimate to whether these objects are related or not. String similarity metrics have different applications in information retrieval field, text classification, spam filtering and document clustering.

Since we deal with character sequences, then string similarity metrics are of particular interest to us. The inputs of similarity metrics will be packet payloads and the goal is to be able to relate different payloads and deduce that they are part of the same traffic entity which ideally is one attack encompassing one application protocol with one signature which is for example DNS traffic with a specific DNS request or HTTP GET traffic with a specific HTTP GET resource.

We represent payloads as a vector of 16 values as indexes from 0 to f according to Hexadecimal alphabet in which the value of index i is the number of

occurrences of character i in the payload. To represent this mathematically, let $P = \{p_1, p_2, \cdots, p_n\}$ be a set of payloads and $T = \{t_1, t_2, \cdots, t_m\}$ be the set of distinct terms occurring in P. A term can be one character or a sequence of characters. A payload can then be represented as an m-dimensional vector $\vec{t_p}$. Let $\vec{t_f}(p, t)$ denote the term frequency of the term $t \in T$ in payload $p \in P$. Then the vector representation of a payload p is $\vec{t_p} = (\vec{t_f}(p, t_1), \vec{t_f}(p, t_2), \cdots, \vec{t_f}(p, t_m))$

Test Setup

For every similarity metric a test is carried-out to estimate the efficiency of the similarity metric for such application. A reference test payload is compared with 80 payloads from which 40 are known to be similar since they are from the same attack and contain the same signature and the other 40 are known to be dissimilar since they are from a different attack and do not share much with the test payload. The different payloads were chosen such that they cover as much cases as possible where they have different lengths and the signature occurs at different offsets. The goal of this test is to show which metric provides the best match for DoS payload comparison. The test data was provided from ddosdb [2].

Decision Criteria

We judge different metrics on two main points:

1. The accuracy of the metric. Accuracy is decided upon the number of correct results the metric gives, which is based upon the decision matrix in figure 5.9 and is given by:

$$Accuracy = \frac{NTruePositive + NTrueNegative}{NTotal} \tag{5.1}$$

   N: is the total number of comparisons

   Positive: is two payloads considered similar. Negative: is two payloads considered not similar

2. The time taken to run the metric. Since, this operation is going to be repeated a lot of times then a fast algorithm is highly needed. Time is judged as the time complexity of the metric algorithm in the big-O notation [10] which simplifies the time comparison into how time grows in relation to input sizes; the inputs in this case are the objects of the metric which are the payloads.

Decision Boundary

Each similarity metric is adjusted to generate a number between 0 and 1 where 1 indicates that the comparison objects are identical and 0 indicates that the comparison objects are disjoint. In some metrics this can be done by only normalizing the metric by dividing the old result by the maximum result while for those that resemble

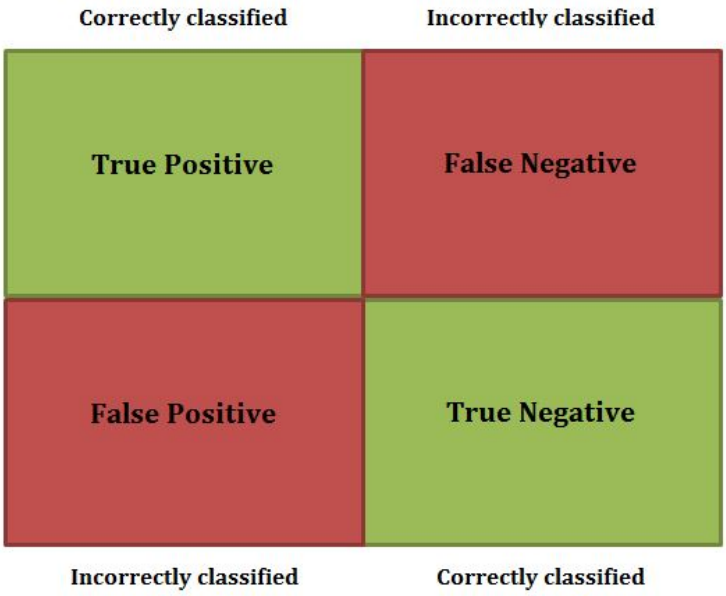


Figure 5.9: Decision Matrix

distance a subtraction operation is performed to get the similarity. For each similarity metric a different optimal threshold is chosen such that it maximizes the accuracy of the metric. The weights of the four quadrants in figure 5.9 are given in table 5.1, those weights are then used to calculate the accuracy of the used metric.

Decision	Weight
True Positive	+1
False Positive	0
True Negative	+1
False Negative	0

Table 5.1: Decision Weights

We eliminate penalties for wrong decisions to simplify accuracy calculations.

Cosine Similarity

This is perhaps the simplest string similarity metric; it represents payloads as vectors of term frequencies as shown in equation 1. The degree of similarity between two payloads is measured as the cosine of the angle between the two vectors which represents the correlation between the two payload vectors. Given two payloads ta, tb, their cosine similarity is

$$CosineSimilarity(t_a, t_b) = \frac{\vec{t_a}.\vec{t_b}}{\|\vec{t_a}\|\|\vec{t_b}\|} \tag{5.2}$$

An important property of the cosine similarity is that its order independent. For example, comparing two payloads like p1= 012ab and p2= ba210 would give a similarity of 1 meaning that cosine the angle between the two payloads is 0 which means that the payloads are identical. Cosine similarity claims that two payloads are identical despite that they are in reverse order.

Cosine similarity algorithm is implemented in $O(max(n, m))$ time complexity where,

n: is the length of the first payload. m: is the length of the second payload

Experimental Results

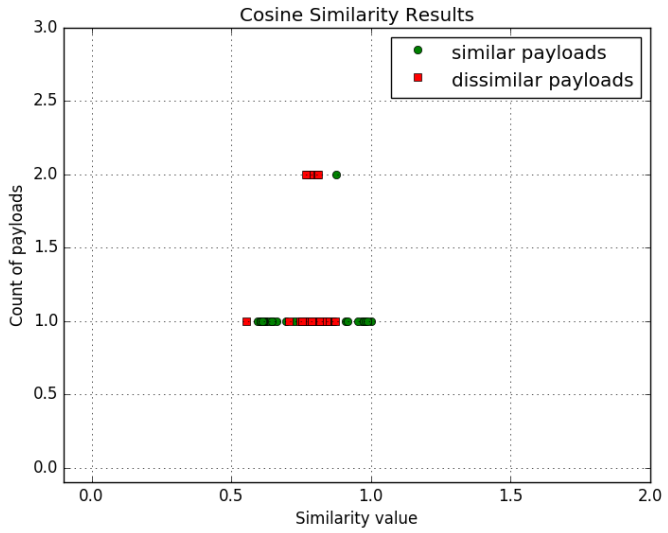


Figure 5.10: Cosine Similarity Results

The results in figure 5.10 show that cosine similarity is not very efficient in such domain. It can be seen that a perfect threshold is not possible as both similar and dissimilar payloads share common regions therefore an optimal threshold that maximizes the accuracy is calculated. The cosine similarity behavior can be attributed to the order independence property of the metric which doesnt lend itself well to a limited alphabet size like that of the hexadecimal characters where most of the alphabet characters are likely present in totally different payloads.

The obtained cosine similarity results are shown in table 5.2. The optimal threshold for the results still yielded a large number of false negatives which brought the accuracy down to 65%.

Optimal Threshold	True Positives	False Positives	True Negatives	False Negatives	Accuracy
0.85	13	1	39	27	65%

Table 5.2: Cosine Similarity Results

Jaccard Index

The Jaccard index measures the intersection of the terms between the given two payloads over the union of the terms in the two payloads. It divides the strings into 1-character long terms and checks how many characters are shared between the two payloads against the total set of present characters. Jaccard index produces 1 for identical payloads and 0 for disjoint payloads.

Given two payloads ta, tb, their Jaccard index is given by:

$$JaccardIndex(t_a, t_b) = \frac{|t_a \cap t_b|}{|t_a \cup t_b|} \tag{5.3}$$

Two important properties of the Jaccard index are

1. It is a present/ not present metric which means that it tests only the fraction of terms that are shared between the two payloads without taking into account their frequencies. In this sense it is less accurate than cosine similarity that considers the frequency of characters.

2. It is order independent which means that it doesnt take into account the order of appearance of the characters and will therefore give a fully identical result to two reversed sequences.

Jaccard index algorithm is implemented in $O(max(n, m))$ time complexity where,
n: is the length of the first payload m: is the length of the second payload

Experimental Results

Jaccard index results shown in figure 5.11 look optimistic until we see the high number of payloads matched at such high similarity value which worsens the metric overall score. This inefficient classification can be expected due to the mentioned properties of the metric especially, the presence/ non presence property combined with the limited hexadecimal alphabet that gives a high chance of finding a lot of the allowed characters in many payloads.

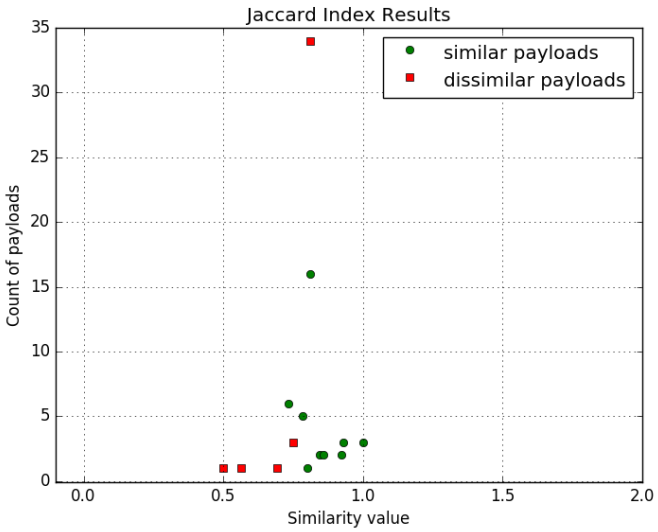


Figure 5.11: Jaccard Index Results

The exact Jaccard results are shown in 5.3. At the optimal threshold, Jaccard index obtained an accuracy result similar to that of cosine similarity.

Optimal Threshold	True Positives	False Positives	True Negatives	False Negatives	Accuracy
0.82	12	0	40	28	65%

Table 5.3: Jaccard Index Results

Levenshtein Distance

Levenshtein distance is an edit-distance string similarity metric. It returns the minimal number of insertions, deletions and replacements needed to transform string a to string b.

The levenshtein distance score was adjusted to return a score between 0 for disjoint payloads and 1 for identical payloads such that,

$$NewScore = 1 - \frac{oldScore}{max(length(payload1), length(payload2))} \tag{5.4}$$

To prove this we take the two cases of having identical and disjoint payloads

- In case the two payloads are identical, the old score will be 0 and the new score will be 1.
- In case the two payloads are disjoint, the old score will be the sum of replacements of each character in the shorter payload and insertions of the remaining characters from the longer payload so the old score will be equal to the length of the longer string and therefore the new score will be 0.

Levenshtein distance algorithm is implemented in $O(n*m)$ time complexity where,
n: is the length of the first payload m: is the length of the second payload

Experimental Results

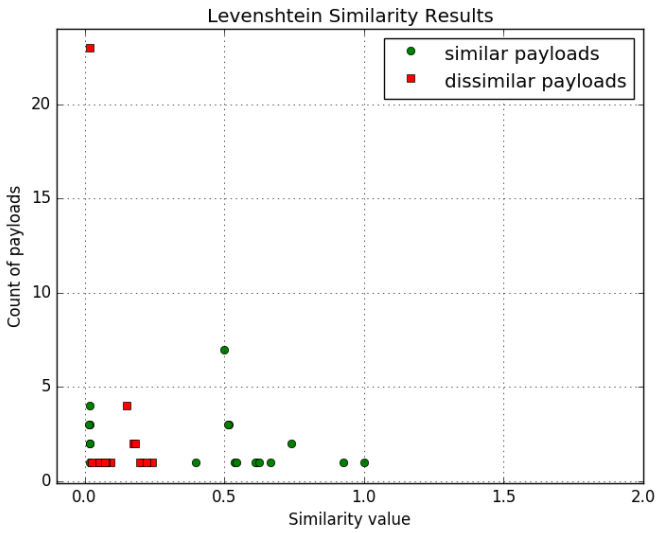


Figure 5.12: Levenshtein similarity Results

As seen in figure 5.12 levenshtein similarity offers better results as more of the dissimilar payloads are having low similarity values and vice versa. This can be because Levenshtein distance implicitly takes into consideration the order and count of characters. However, results are still not optimal to our application.

The exact obtained Levenshtein similarity results are shown in 5.4

Optimal Threshold	True Positives	False Positives	True Negatives	False Negatives	Accuracy
0.25	24	0	40	16	80%

Table 5.4: Levenshtein similarity results

Smith-Waterman Alignment Score

Smith-Waterman algorithm [21] is an algorithm that aims to determine the similar regions between two strings by finding the best local alignment of those two strings. It is commonly used in DNA sequencing and protein sequencing applications. It provides a score at the end for every two sequences based on what is called a scoring scheme. The scoring scheme involves three costs which are

- Match score: gets added when the current aligned two characters in the two strings match.

- Mismatch score: considered a penalty and gets subtracted when the current aligned two characters in the two strings mismatch.

- Gap score: considered a penalty and gets subtracted whenever a character insertion or deletion operation is needed.

Smith-Waterman algorithm is coded in a dynamic programming [10] approach that takes $O(n * m)$ time complexity where,

n: is the length of the first string m: is the length of the second string

The scoring scheme used in Smith-Waterman algorithm was equal weighted for simplicity and given by:

Match Score: +1 Mismatch Penalty: -1 Gap Penalty: -1

The alignment score was normalized to return a score between 0 for disjoint payloads and 1 for identical payloads such that,

$$NewScore = \frac{oldScore}{min(length(payload1), length(payload2))} \tag{5.5}$$

This is because the max alignment score when the two payloads are identical or one is subset from another is equal to the length of the shorter payload, while, when the two payloads are disjoint the score is 0.

Smith-Waterman algorithm is implemented in $O(n * m)$ time complexity where,

n: is the length of the first payload m: is the length of the second payload

Experimental Results

Results for Smith-Waterman algorithm shown in figure 5.13 are optimum. This was highly expected because Smith-Waterman algorithm tries to find the best local alignment of the compared payloads. This in our case would align the signature in the two similar payloads and fail to do so in dissimilar payloads.

The exact obtained Smith-Waterman results are shown in 5.5

Optimal Threshold	True Positives	False Positives	True Negatives	False Negatives	Accuracy
0.22	40	0	40	0	100%

Table 5.5: Smith-Waterman similarity results

Longest Common Subsequence

Although not strictly a similarity metric, it gives a good indication about how similar are the contents of two payloads. The longest common subsequence [9] problem aims to find the length of longest subsequence of sequential or non-sequential characters that are repeated in two strings. The greater the length, the greater

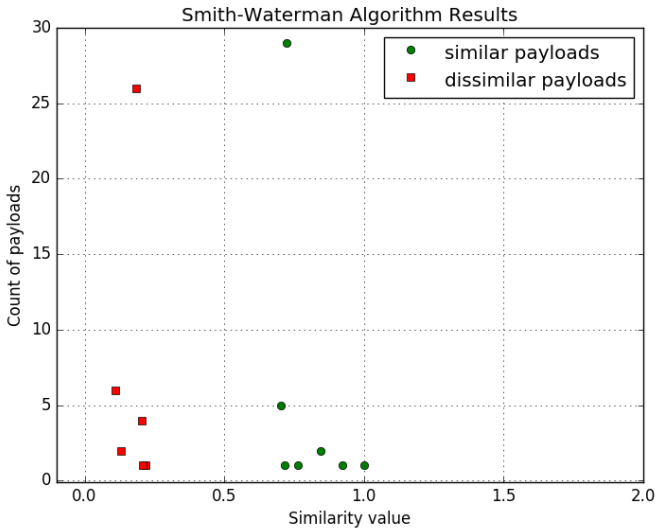


Figure 5.13: Smith-Waterman similarity Results

the similarity of the compared strings. For example, let s_1=012ab44458 and s_2= 0932ab48323 then

LCSubsequence (s_1, s_2) = 02ab48

An important property of the longest common subsequence is that it takes into account the order of appearance of the characters in the strings so unlike other similarity metrics the result of two reversed strings will be one which means that the two strings are dissimilar.

The longest common subsequence score was normalized to return 0 for disjoint payloads and 1 for identical payloads such that,

$$NewScore = \frac{oldScore}{min(length(payload1), length(payload2))} \tag{5.6}$$

This is because the max score that can be returned by the longest common subsequence when the two payloads are identical or one is subset of the other is the length of the shorter payload while when both payloads are disjoint the minimum score is 0.

The longest common subsequence is solved using dynamic programming in an approach of time complexity $O(n * m)$ where:

n: is the length of the first payload m: is the length of the second payload

Experimental Results

The longest common subsequence results shown in figure 5.14 are worse than expected, especially, with the high dissimilar payloads matched at a similarity value of 1. This is probably because of the limited alphabet of hexadecimals which make the presence of an entire payload as a subsequence of another more likely espe-

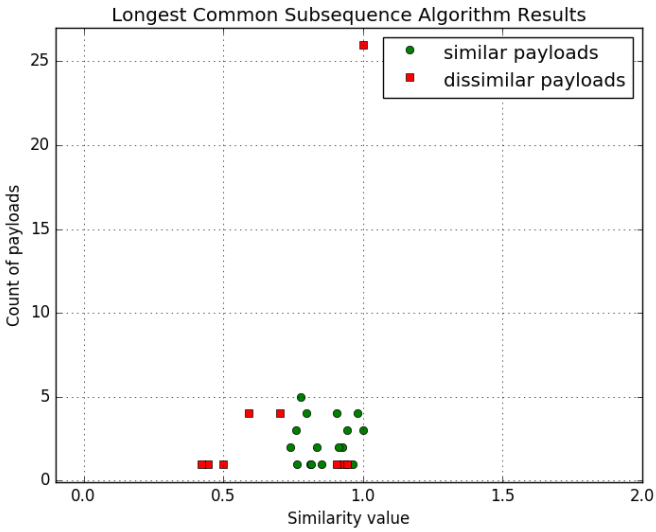


Figure 5.14: Longest common subsequence results

cially, when one payload is considerably shorter than the other.

The obtained longest common subsequence results are shown in 5.6

Optimal Threshold	True Positives	False Positives	True Negatives	False Negatives	Accuracy
0.72	40	28	12	0	65%

Table 5.6: Longest common subsequence results

Longest Common Substring

Although not a straight forward similarity metric, longest common substring can provide a good estimate of how similar payloads are based on the length of the longest substring they share where a substring is a group of sequential characters. Taking the same example given for the longest common subsequence metric , let s_1=012ab44458 and s_2= 0932ab48323 then

LCSubstring (s_1, s_2) = 2ab4

The longest common substring score was normalized to return 0 for disjoint payloads and 1 for identical payloads such that,

$$NewScore = \frac{oldScore}{min(length(payload1), length(payload2))} \tag{5.7}$$

This is because the max score that can be returned by the longest common substring when the two payloads are identical or one is subset of the other is the length of the shorter payload while when both payloads are disjoint the minimum score is 0.

The longest common substring is solved using dynamic programming in an approach of time complexity $O(n * m)$ where: n: is the length of the first payload m: is the length of the second payload

Experimental Results

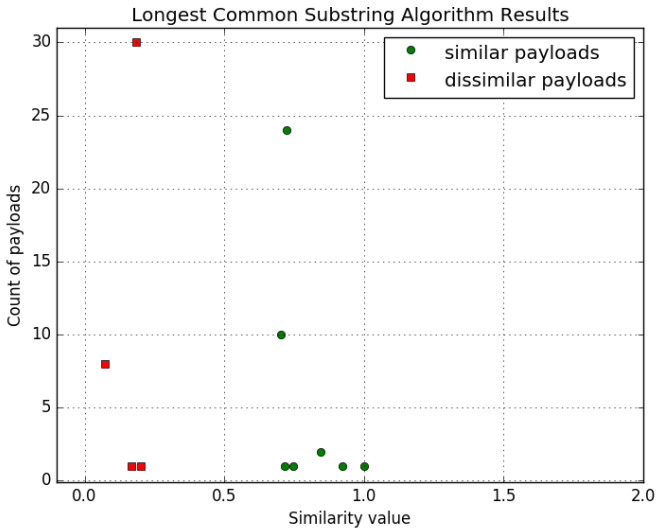


Figure 5.15: Longest common substring results

The longest common substring approach also reached optimal results. This can be explained by the fact that in similar payloads the longest common substring is the signature while this is not the case and the longest common substring is of much shorter length in dissimilar payloads.

The exact obtained longest common substring results are shown in 5.7.

Optimal Threshold	True Positives	False Positives	True Negatives	False Negatives	Accuracy
0.2	40	0	40	0	100%

Table 5.7: Longest common substring results

Due to the accurate results achieved through the longest common substring, another experiment was run on the same dataset to get the lengths of the longest common substrings between the original payload and the similar and dissimilar payloads. The results are shown in figure 5.16

From the shown results, it can be observed that dissimilar payloads share a substring of length around 10 characters and similar payloads share more than 30 characters substrings. Since these results are attack dependent to a certain extent; hard generalized thresholds cannot be used however, a relaxed threshold of around 20 characters should be a fair threshold for differentiating different patterns in an attack.

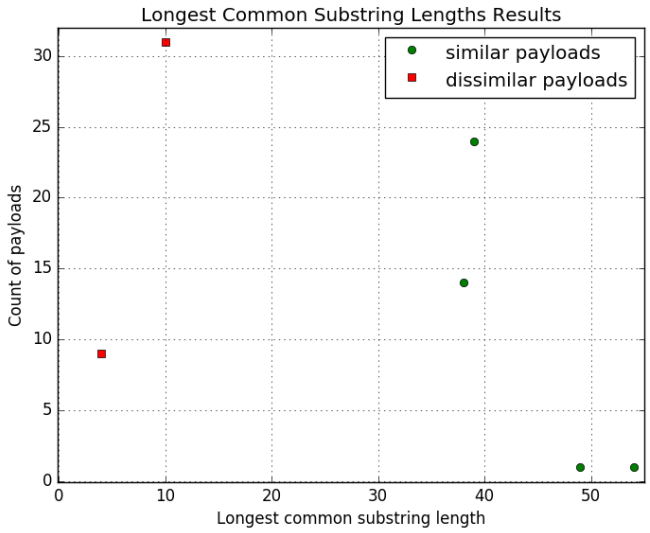


Figure 5.16: Longest common substring lengths

Time Optimization

Using the observation from figure 5.16 that similar payloads share a considerably longer substring than dissimilar payloads, an algorithm could be implemented that looks for configurable minimum shared substring. This algorithm is implemented in an approach of time complexity $O(max(n, m))$ where, n: is the length of the first payload m: is the length of the second payload and is referred to as "Minimum shared substring" algorithm in this work. The algorithm is shown in 1

Data: payload1, payload2, n
1 `/* n is the minimum acceptable signature length */`
Result: boolean: whether payloads are similar or not
2 **foreach** $payload$ *of the two payloads* **do**
3 $ngrams \leftarrow payload$ ngram sequences; `/* ngrams now has all shared substrings of size n */`
4 **end**
5 **foreach** $ngram$ *in one of the payload ngram sets* **do**
6 **if** $ngram$ *in the other payload ngram set* **then**
7 return $True$;
8 `/* The two payloads share a substring of minimum acceptable size and therefore are considered similar */`
9 **end**
10 return $False$;
11 **end**

Algorithm 1: Minimum shared substring algorithm

The inputs to the minimum shared substring algorithm is the two payloads to compare and the minimum accepted signature length which is referred to as n while the output is true or false indicating whether the two payloads are similar or not. In lines $2 \rightarrow 4$ all the n-sized substrings are extracted from both payloads. In lines $5 \rightarrow 10$ we check if any n-sized substring is shared between the two payloads; if one is found the payloads are considered similar and true is returned, otherwise false is returned. One thing to note here is that similar payloads will share a substring of minimum size n. This remark guarantees that the final signature will be at least n characters long and may be longer.

Shared n-gram Approach

In the examined approaches, payloads have been divided based on a 1-gram scheme where only each individual character was considered a term. However, another approach is to deal with sequential characters in a window manner where an n-gram scheme divides a sequence into n-character sequential terms. For example, using a 2-gram with sequence= 012abc69 would result in a set of terms equal to t = 01, 12, 2a, ab, bc, c6, 69 and so on. The number of n-gram terms present in a sequence of length L is given by:

$$ngramTerms = L - n + 1 \tag{5.8}$$

All the previous tests could be repeated using the shared n-gram approach in which different n sizes are chosen. However, since this introduces additional complexity of choosing the best n size and since an acceptable result was reached, those tests were not repeated.

A summary of the similarity metrics results is shown in table 5.8

Similarity Metric	Accuracy	Time Complexity
Cosine Similarity	65%	$O(max(n,m))$
Jaccard Index	65%	$O(max(n,m))$
Levenshtein Distance	80%	$O(n*m)$
Smith-Waterman Algorithm	100%	$O(n*m)$
Longest Common Subsequence	65%	$O(n*m)$
Longest Common Substring	100%	$O(n*m)$
Minimum Shared Substring	100%	$O(max(n,m))$

Table 5.8: Similarity metrics results

5.4 Signature Generation

Signature generation is the process of extracting the common variant that represents the attack. This signature is the most common shared content in the original attack trace.

To be able to generate a signature we build up on the aforementioned chosen methods for attack filtering 5.2 and payloads classification 5.3

Longest common substring algorithm was chosen to extract the signature from different payloads. This is because the longest common substring algorithm can be implemented in a space- time efficient manner. Also, the generated signature can be incorporated to an existing S-IDS database and matched using simple regular expressions.

In this section, two algorithms are presented, the first relies on the source IPs approach as an attack filtering method while the second relies on the most frequent payloads for attack filtering. Both algorithms implement the minimum shared substring method for payloads classification. Figure 5.17 shows a simple demonstration of the two proposed algorithms.

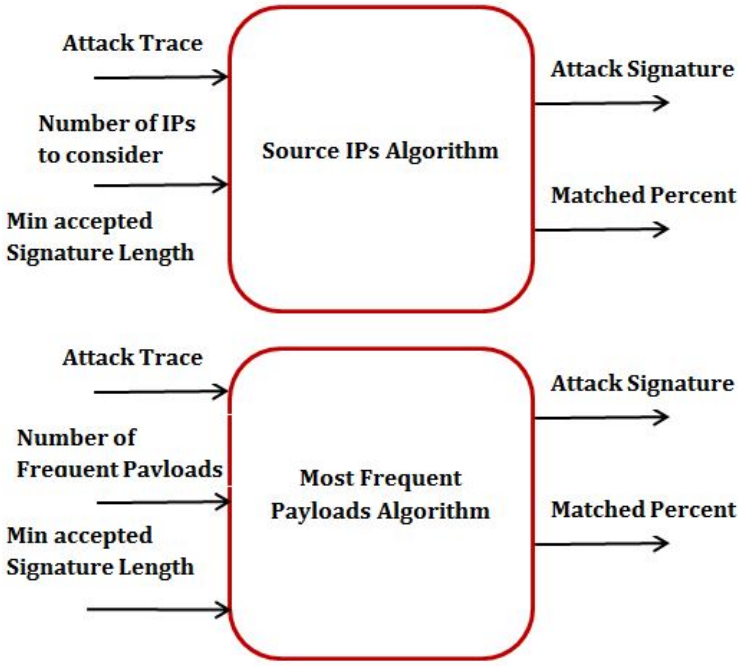


Figure 5.17: Source IPs and frequent payloads approaches demo

Test Setup

To test the implemented algorithms, they were run on 10 DNS attacks and 10 Chargen attacks from ddosdb database [2]. The attacks followed the criteria in which there was a clearly dominant attack in the trace accompanied by lower occurences

of other traffic; that can be other attack variants or normal traffic. The algorithms' goal is to find the signature of the dominant attack. Results are presented in section 5.4.3.

5.4.1 Source IPs Algorithm

```
   Data: Attack trace, Number of IPs to consider, signature threshold
   Result: Attack signature, matching percentage
 1 sort IPs from the most sending to least ;
 2 foreach IP of the top n IPs do
 3 |   add longest common substring(longest payload, shortest payload)
   |    to obtained signatures ;
 4 end
 5 foreach signature of obtained signatures do
 6 |   if similar(signature, other signatures from obtained signatures) then
 7 |   |   add longest common substring(similar signatures) to final
   |   |    signatures ;
 8 |   end
 9 |   else
10 |   |   add unique signature to final signature
11 |   end
12 end
13 foreach signature of final signatures do
14 |   check the percentage it matches in original attack trace ;
15 end
16 return the signature that matches the most and its matching percentage
    ;
```

Algorithm 2: Source IPs algorithm

The inputs to the source IPs algorithm are the attack network trace, the number of source IPs to process and the signature threshold which is the minimum acceptable signature length. While, the outputs are the attack signature and the percentage it matches from the original trace. The first step is to sort the source IPs according to the amount of traffic that they send. In lines $2 \rightarrow 4$ a signature is extracted for the traffic of every IP. This signature is extracted by getting the longest common substring between the shortest and longest payload sent by this IP, if all the payloads of the same IP are of the same length, the first payload is considered the signature for this IP traffic. This method speeds the time considerably as we only need to consider maximum two payloads for every IP. This method is also supported as we expect every IP to send one attack variant and to have this attack variant in all its

traffic. In lines $5 \rightarrow 11$ we find similar IP signatures according to our minimum shared substring algorithm and update the final signatures with the mixed signatures and the unique signatures that have no similar ones. In that way, different IPs participating in the same attack are found and their signatures are mixed so that we get a more accurate signature for this attack traffic. Finally, in lines $13 \rightarrow 16$, the signature that is most frequent in the attack trace payloads is extracted and returned with its matching percentage.

5.4.2 Most Frequent Payloads Algorithm

```
Data: Attack trace, Number of frequent payloads to consider,
      signature threshold
Result: Attack signature, matching percentage
sort payloads from most frequent to least ;
add first payload to unique patterns ;
foreach payload in top n payloads do
    if similar(payload, any pattern in unique patterns) then
        if pattern signature is not substring of payload then
            patternsignature ← longest common substring(pattern
              signature, payload) ;
        end
        else
            do nothing
        end
    end
    else
        add unique payload to unique patterns ;
    end
end
foreach signature of pattern signatures do
    if similar(signature, other signatures from obtained signatures)
     then
        add longest common substring(similar signatures) to final
          signatures ;
    end
    else
        add unique signature to final signature;
    end
end
foreach signature of final signatures do
    check the percentage it matches in original attack trace ;
end
return the signature that matches the most and its matching
  percentage ;
```

Algorithm 3: Most frequent payloads algorithm

The inputs to the most frequent payloads algorithm are the attack trace, the number of most frequent payloads to process and the minimum acceptable signatue

length. While, the outputs are the attack signature and its matching percentage. The first step is to count the frequenct of the payloads and get the required most frequent ones. The next step is to collect the unique patterns in those frequent payloads and get their signatures. In lines $2 \rightarrow 15$ we start by considering the most frequent payload as a unique pattern and then start checking the similarity (according to minimum shared substring algorithm) betweeh the other frequent payloads and the extracted unique patterns (originally contain the most frequent payload only) and if two are found similar, a signature is extracted for them through the longest common substring algorithm. If one is found unique it is added to unique patterns. At the end of this step we have all the unique patterns represented in the most frequent payloads and their signatures. In lines $16 \rightarrow 23$ we do another pass on the obtained signatures for finding similar ones like in the source ips algorithm. Finally, the signature that matches the most of the attack trace if returned accompanied by its matching percentage.

5.4.3 Experimental Results

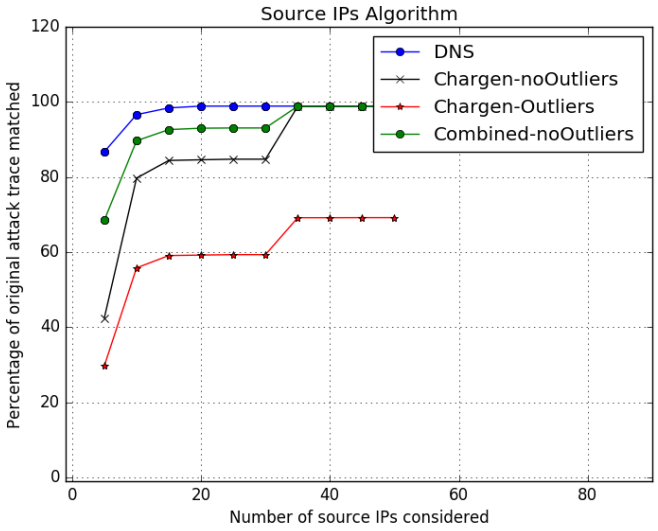


Figure 5.18: Signature generation using source IPs algorithm at 20 minimum signature length

The results for the most sending source IPs algorithm are shown in figure 5.18. It can be observed that for DNS more accurate results are reached with inspecting less IPs. For both DNS and Chargen high percentage reaching 99% of the attack traces could be matched with the generated signatures. The original payloads that weren't matched are either parts of another attack or traffic of a different nature for example legitimate traffic. It is also observed and expected that as more source

IPs traffic is processed, a more accurate signature is generated. This accuracy improvement is because more attack variations get processed when more source IPs are considered until a poit of saturation is reached. At this saturation point processing more IPs traffic adds low to no value.

It is worth noting here that out of the 10 tested chargen attacks, the source IPs algorithm could not extract a signature of length more than or equal to the specified minimum length (20 characters). After further inspection, the reason was found to be that some IPs send payloads of much different lengths in which the shortest and longest payloads share a very short signature (around 2 to 5) characters. Despite that those extremely short payloads are not frequent, they do affect the final signature generated for every IP which affects the whole signature generation process. The impact of those outliers can be seen in the graph at Chargen-Outliers where the average matched percentage is considerably lower than the noOutliers result. Whether those special cases are really outliers or a frequent case needs more testing, however, they give a good example of the limitations of the source IPs algorithm.

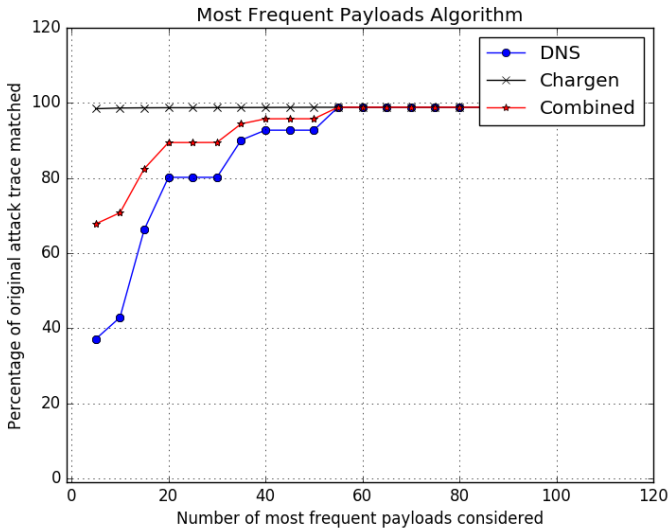


Figure 5.19: Signature generation using most frequent payloads algorithm at 20 minimum signature length

The results of the most frequent payloads algorithm are shown in figure 5.19. It can be observed that for Chargen an accurate signature that represents around 99% of the attack traffic is generated from as early as investigating only the 5 most frequent payloads. For DNS attacks, the case was a little different in which the most accurate signature was reached after inspecting around the most frequent 55 payloads. It is also worth mentioning that as the number of investigated payloads increases the signature matching percentage increases. This direct relation is be-

cause as more payloads get processed, more patterns of the same attack are seen which improves the accuracy of the generated signature. This trend continues until all the frequent patterns have been taken into consideration and investigating more payloads results in little to no improvements.

From the conducted tests it can be concluded that the most frequent payloads algorithm provides more accurate results than the most sending source IPs algorithm. This conclusion is because the same attacks were ran for both algorithms and the most frequent payloads algorithm showed no outliers and could efficiently generate an attack signatures. However, the source IPs algorithm worked well for DNS attacks but could not generate a signature for 3 Chargen attacks.

To further present the results of the most frequent payloads algorithm, the confidence interval was calculated for the tested attacks at 90% confidence level. The results are shown in figure 5.20.

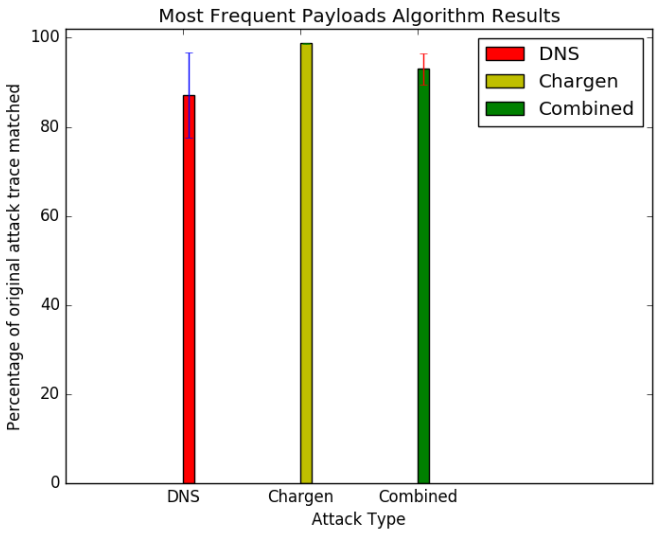


Figure 5.20: Most frequent payloads confidence results

5.5 Use Case

This is the final phase of our signature generation approach. It deals with incorporating the signatures generated in the signature generation step 5.4 into a functional signature based intrusion detection system and analyzing their behavior. Snort [26] was chosen as the IDS for the use case phase. This is because of multiple reasons which are:

1. Snort has been around for a long time (since the end of the 1990s) which makes it stable and well tested.
2. Snort has a wide community behind it and was widely used in academia and industry.
3. Snort is simple to install and use which saves time and effort.

5.5.1 Test Setup

Snort works by having a set of rules defined at specific files [6]. When snort is activated it starts checking those rules against the incoming traffic, if any rule is matched the specified action is triggered. Snort has a simple way of writing rules. The generic snort rule format is shown in figure 5.21.

```
action protocol src_ip src_port -> dst_ip dst_port (rule_specs)
```

Figure 5.21: Snort rule format

Snort rules generally follow the following format:

- **Action:** This is the action to be taken when a rule is matched. For example it can be log to store some info about the packets firiing the rule or drop to drop the traffic firing that rule or others.
- **Protocol:** This is the protocol that snort would look for when analyzing the traffic.
- **src ip and srcport:** Those are the source IPs and ports that snort would look for in the incoming traffic.
- **dst ip and dst port:** Those are the destination IPs and ports that snort would look for in the incoming traffic.
- **Rule specs:** This is the exact rule specificaions. It can include content to look for in packets, message to relate to a certain rule and the rule id.

Snort version 2.9.9.0 is used. In order to check how effective the generated signatures are, twenty Attack traces (the ones tested in signature generation phase 5.4) were read by snort and snort default rules were turned off. Only one rule was added to look for the generated attack signature in the traffic. An example of the rule used is *alert udp any any -¿ any any (content:"—08666b666b666b667a04677572750000ff0001—";msg:"SIGNAT GENERATION ALGORITHM LOGGED ERROR"; sid:1000002;)*. This rule analyzes all UDP packets coming from any source and going to any destination. It looks for the content (signature) in all the packets by doing pattern matching algorithms and if found an alert is fired and logged with the message.

5.5.2 Experimental Results

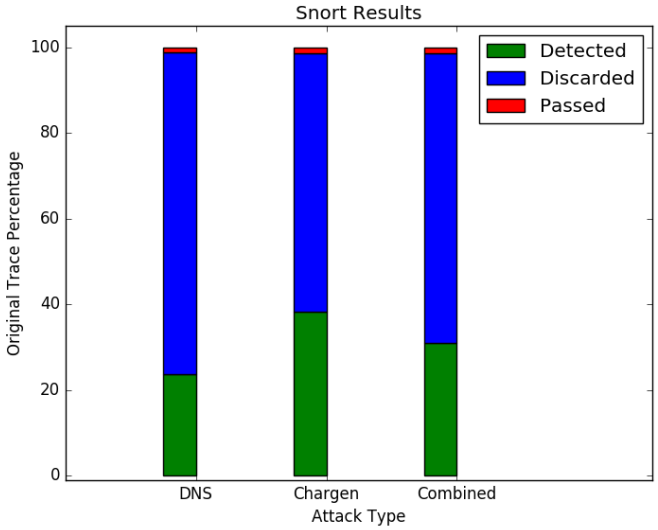


Figure 5.22: Snort results

The results of snort tests are shown in figure 5.22. The results are interesting as they show some unexpected behavior of Snort. It was expected that Snort would match the same percentages found in 5.4.3. The unexpected part was that Snort discarded a considerable amount of the trace packets due to basic encoding integrity flaws that prevented it from decoding the packet. After more inspection these discarded packets were found to be tuncated which is related to our input traces and could be dealt with in real life scenarios. Those parts are seen as blue in the above figure. For the remaining portion of the trace, Snort was able to correctly find the packets that contained the signature and they were correctly logged; those detected packets percentage is shown as green in the above graph. The red part depicts the packets that didn't match the specified signature; those are suspected to be traffic of different behavior for example normal traffic.

Part III

Conclusions and Future Work

Chapter 6

Conclusions and Future Work

In this section, we present our conclusions and future work possibilities.

6.1 Conclusions

The work in this thesis serves as a complete demonstration for automatically generating a signature for payload based DDoS attacks. This work was accomplished by first identifying the types of attacks that are suitable for payload based signature generation. (**RQ1: What types of attacks are likely to share a payload signature?**) a criteria was proposed for judging whether an attack is suitable for payload inspection or not. Among the most common types of attacks in the last few years, DNS-based and Chargen-based attacks are more likely to share payload signatures.

Secondly, the followed approach for signature generation was presented. (**RQ2:How can signatures be extracted from attacks?**) it started by finding methods to enable faster and less computationally intensive processing for attacks by minimizing the amount of traffic processed. Six similarity metrics were then investigated to be able to find relations between the traffic and identify the most frequent pattern within the traffic which is considered the attack. Smith-Waterman and Longest Common Substring algorithms showed the highest accuracy in payload classification. A signature was then generated for the attack pattern identifying as many as possible from the attack packets.

(**RQ3: How representative are the extracted signatures for detecting attacks?**) finally, the generated signatures were added as new rules for Snort IDS and validated by correctly being able to identify and handle the attack packets. Through the presented process, the goal of automatic signature generation was reached.

6.2 Future Work

Automatic signature generation is an interesting continuously evolving problem, especially with the impact and complexity of attacks nowadays. For that reason, more research can be done to further improve the reached results. A more comprehensive study on different attack types can be done to determine the applicability of the payload inspection method on them for example HTTP-based attacks. We focused on similarity metrics for finding relations between different attack packets using 1-gram scheme, this can be extended by using a variable n-gram scheme and inspecting how this changes the results. In addition to that, other techniques like machine learning could be investigated for the packet clustering task.

We expect the proposed approach to be able to generate signatures for multi-vector attacks but in multiple runs. Each run generates the signature for the most dominant attack and then the trace is filtered to exclude the signature matched trace and the algorithm is rerun and so on. It is also possible with minimal changes in the proposed algorithm to generate the most dominant signatures. However, this investigation presents an interesting research which would complement the proposed research.

Finally, it would be beneficial to setup the complete environment proposed in this thesis. This can be done by integrating the proposed algorithm into a system having both signature based and anomaly based intrusion detection systems. This research would help solidify the whole scenario and present a fully integrated tested deployable signature generation system.

Bibliography

[1] Akamai state of the internet security report q1-2017. https://www.akamai.com/us/en/multimedia/documents/state-of-the-internet/q1-2017-state-of-the-internet-security-report.pdf, 2017.

[2] Ddos attacks database. ddosdb.org, 2017.

[3] Top 10 ddos attack trends. https://www.imperva.com/docs/DS_Incapsula_The_Top_10_DDoS_Attack_Trends_ebook.pdf, 2017.

[4] Top ddos attacks 2015. https://blog.radware.com/security/2016/01/top-ddos-attacks-2015/, 2017.

[5] Most popular cyber attacks 2016. https://security.radware.com/ddos-knowledge-center/ddos-attack-types/most-popular-cyber-attacks-2016/, 2017.

[6] Snort manual. http://manual-snort-org.s3-website-us-east-1.amazonaws.com/, 2017.

[7] Rise of ddos attacks. http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-continued-rise-of-ddos-attacks.pdf, 2017.

[8] What is a ddos attack. https://www.verisign.com/en_US/security-services/ddos-protection/what-is-a-ddos-attack/index.xhtml, 2017.

[9] Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*, pages 39–48. IEEE, 2000.

[10] Paul E Black. big-o notation. *Dictionary of Algorithms and Data Structures*, 2007, 2007.

[11] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[12] Kevin R Fall and W Richard Stevens. *TCP/IP illustrated, volume 1: The protocols*. addison-Wesley, 2011.

[13] Pedro Garcia-Teodoro, J Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1):18–28, 2009.

[14] Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge university press, 1997.

[15] Kai Hwang, Min Cai, Ying Chen, and Min Qin. Hybrid intrusion detection with weighted signature generation over anomalous internet episodes. *IEEE Transactions on Dependable and Secure Computing*, 4(1), 2007.

[16] Vijay Katkar and SG Bhirud. Novel dos/ddos attack detection and signature generation. *International Journal of Computer Applications*, 47(10):18–24, 2012.

[17] Vijay Katkar and Rejo Mathew. One pass incremental association rule detection algorithm for network intrusion detection system. *International Journal of Engineering Science and Technology*, 3(4), 2011.

[18] Sanmeet Kaur and Maninder Singh. Automatic attack signature generation systems: A review. *IEEE Security & Privacy*, 11(6):54–61, 2013.

[19] Christian Kreibich and Jon Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *ACM SIGCOMM computer communication review*, 34(1):51–56, 2004.

[20] Mohssen MZE Mohammed, H Anthony Chan, and Neco Ventura. Honeycyber: Automated signature generation for zero-day polymorphic worms. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–6. IEEE, 2008.

[21] Richard Mott. Smith–waterman algorithm. *eLS*, 2005.

[22] Nnamdi Nwanze and Douglas Summerville. Detection of anomalous network packets using lightweight stateless payload inspection. In *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*, pages 911–918. IEEE, 2008.

[23] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23):2435–2463, 1999.

[24] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *NSDI*, volume 10, page 14, 2010.

[25] Georgios Portokalidis, Asia Slowinska, and Herbert Bos. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. In *ACM SIGOPS Operating Systems Review*, volume 40, pages 15–27. ACM, 2006.

[26] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *Lisa*, volume 99, pages 229–238, 1999.

[27] Christian Rossow. Amplification hell: Revisiting network protocols for ddos abuse. In *NDSS*, 2014.

[28] Karen Scarfone and Peter Mell. Guide to intrusion detection and prevention systems (idps). *NIST special publication*, 800(2007):94, 2007.

[29] Edward J Schwartz, Thanassis Avgerinos, and David Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Security and privacy (SP), 2010 IEEE symposium on*, pages 317–331. IEEE, 2010.

[30] Urjita Thakar, Nirmal Dagdee, and Sudarshan Varma. Pattern analysis and signature extraction for intrusion attacks on web services. *International Journal of Network Security & its Applications (IJNSA)*, 2(3), 2010.

[31] Ke Wang and Salvatore J Stolfo. Anomalous payload-based network intrusion detection. In *RAID*, volume 4, pages 203–222. Springer, 2004.

[32] Ke Wang, Janak Parekh, and Salvatore Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Advances in Intrusion Detection*, pages 226–248. Springer, 2006.