MODELLING BEHAVIOURAL DYNAMICS FOR APPLICATION LAYER DISTRIBUTED DENIAL OF SERVICE ATTACK DETECTION

Thesis

Submitted in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

AMIT PRASEED

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA

SURATHKAL, MANGALORE - 575 025

August, 2020

DECLARATION

by the Ph.D. Research Scholar

I hereby declare that the Research Thesis entitled **Modelling Behavioural Dynamics for Application Layer Distributed Denial of Service Attack Detection** which is being submitted to the **National Institute of Technology Karnataka, Surathkal** in partial fulfilment of the requirements for the award of the Degree of **Doctor of Philosophy** in Department of Computer Science and Engineering is a bonafide report of the research work carried out by me. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

Amit Praseed, 165003 CS16F01

Department of Computer Science and Engineering

Place: NITK, Surathkal.

Date: August 7, 2020

CERTIFICATE

This is to certify that the Research Thesis entitled **Modelling Behavioural Dynamics for Application Layer Distributed Denial of Service Attack Detection** submitted by **Amit Praseed** (Register Number: 165003 CS16F01) as the record of the research work carried out by him, is accepted as the Research Thesis submission in partial fulfilment of the requirements for the award of degree of **Doctor of Philosophy**.

Prof. P. Santhi Thilagam
Research Guide

(Signature with Date and Seal)

Chairman - DRPC
(Signature with Date and Seal)

ACKNOWLEDGEMENTS

It is my pleasure to thank the people who have supported and helped me throughout the duration of my research work. First and foremost, I would like to express my sincere gratitude to my research supervisor Dr. P. Santhi Thilagam, Professor, Department of Computer Science and Engineering, for her constant support and encouragement. She has been a constant guiding light throughout the duration of my research. I have learned a lot from her, both from a technical standpoint and in the form of life lessons, which will be of use throughout my life.

I would like to extend my gratitude to the members of my research progress assessment committee, Dr. Alwyn R. Pais and Dr. Vinatha U., for their valuable suggestions throughout the course of my research. I also extend my gratitude to the technical and administrative staff of the Department of Computer Science and Engineering for all the help and co-operation extended to me. They were always ready to lend a hand whenever I faced any issues, and for that I am grateful.

I would like to thank NITK for providing the infrastructure and facilities for the smooth conduct of my research. I also express my gratitude to the Ministry of Electronics and Information Technology (MeitY), Government of India, for funding our research work. The suggestions provided by the Project Review Steering Group (PRSG) have been invaluable in strengthening my research. I would also like to thank Mr. Sivakumar K. for the technical assistance extended to me as part of our project team. I would also like to extend my gratitude to Dr. Deepa G., Dr. Bindu P.V., Mr. D.V.N. Sivakumar, Mrs. Umapriya D., Mr. Srinivas, Nikhil and Pramod for their support and encouragement throughout the course of my research.

I am deeply indebted to my parents for encouraging and supporting me throughout the course of my research and indeed, throughout my life.

Finally, I would like to thank the Almighty for granting me the health, strength and wisdom for carrying out my research work.

Amit Praseed

ABSTRACT

Distributed Denial of Service (DDoS) attacks are one of the oldest and most dangerous attacks against network infrastructure and web applications alike. Traditionally, DDoS attacks were executed using network layer protocols for generating a large volume of requests, thereby exhausting the network bandwidth and leading to a degradation in the quality of Internet services. These attacks, called Network layer DDoS attacks, no longer pose a significant threat due to the availability of cheap bandwidth and the development of sophisticated detection mechanisms against these attacks. With the success of network layer DDoS attacks no longer guaranteed, attackers have slowly started using application layer protocols to launch DDoS attacks. Application Layer DDoS (AL-DDoS) attacks attempt to take down web applications by exhausting server resources such as CPU, database, memory or socket connections by using the features of application layer protocols. Majority of these attacks use the HTTP/1.1 protocol for launching attacks, and in particular there has been a growing trend of using computationally expensive requests to launch DDoS attacks. These attacks are called Asymmetric AL-DDoS attacks and are generally imperceptible owing to the use of legitimate requests and a comparatively low attack volume. Due to these features, simple firewall rules and request inspection techniques are ineffective against these attacks and hence, an analysis of user behaviour is required for detecting these attacks. Most of the existing detection mechanisms focus on building a model of legitimate user behaviour as under the HTTP/1.1 protocol, and then identifying attacks by observing the deviation from the learned model. Existing detection approaches for asymmetric AL-DDoS attacks use indirect representations of actual user behaviour and use complex modelling techniques, which leads to a higher false positive rate (FPR) and longer detection time, which makes them unsuitable for real time use. In addition, most of these models are unable to adapt to changing user behaviour, which leads to the model becoming ineffective in the long run. A review of existing literature suggests that there is a need for a

lightweight, fast and adaptable detection mechanism for asymmetric AL-DDoS attacks that has a very low false positive rate.

The recent standardization of HTTP/2 adds another layer of complexity over asymmetric AL-DDoS detection. HTTP/2 was designed to improve the performance of web servers, and has been greatly successful in reducing the average response time of web servers due to the introduction of features like multiplexing and server push. This has led to more and more web applications migrating to HTTP/2. However, while reducing page load time for clients, HTTP/2 puts additional load on web servers, leading to concerns about these servers being more vulnerable to asymmetric AL-DDoS attacks. In addition, there is no evidence found in existing literature regarding the possibility of multiplexing and server push being misused to launch potentially lethal asymmetric AL-DDoS attacks. This lack of understanding has led to existing mechanisms being unable to handle the HTTP/2 protocol effectively.

In this work, an attempt is made to model the actual behavioural dynamics of legitimate users using an annotated Probabilistic Timed Automata (PTA) along with a suspicion scoring mechanism for differentiating between legitimate and malicious users. This allows the detection mechanism to be extremely fast and have a low FPR. In addition, the model can adapt to changing user behaviour in an incremental manner, which further reduces the FPR. Experiments on public datasets reveal that our proposed approach has a high detection rate and low FPR and adds negligible overhead to the web server, which makes it ideal for real time use.

This work also explores the impact of asymmetric AL-DDoS attacks on HTTP/2 servers. Our experiments demonstrate that an HTTP/2 server is actually more resilient to asymmetric AL-DDoS attacks as compared to an HTTP/1.1 server. However, despite the improved resilience, HTTP/2 servers are vulnerable to a more sophisticated class of attacks. We demonstrate that multiplexing and server push features in HTTP/2 can be misused to launch a sophisticated attack called Multiplexed Asymmetric Attack, that can exhaust server resources much faster and with minimal number of attacking clients. In order to detect these attacks, the PTA-based behavioural model is extended to accommodate HTTP/2-specific features. Our experiments demonstrate that the in-

clusion of these features allows the system to detect Multiplexed Asymmetric Attacks effectively.

There is a considerable degree of similarity between attacking connections in a DDoS attacks due to the use of common attack generation tools and botnets. In the case of an AL-DDoS attack, this similarity manifests itself in the form of repeating sequences of HTTP requests across attacking connections. Knowledge of this similarity allows for the early detection of AL-DDoS attacks, thereby reducing the average detection time of the system and allowing it to operate in real time. A dynamic signature based approach using HTTP request sequences is used in order to facilitate the early detection of AL-DDoS attack as part of the proposed approach. Experimental results indicate that the use of the early detection mechanism leads to a considerable decrease in detection time, and leads to efficient real time use.

Keywords: application layer, ddos, http, http/2, multiplexing, asymmetric, attack, detection

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviations	Expansion
AL-DDoS	Application Layer Distributed Denial of Service
C&C	Command and Control
DoS	Denial of Service
DDoS	Distributed Denial of Service
DNS	Domain Name Service
FPR	False Positive Rate
FSM	Finite State Machine
HTTP	Hyper Text Transfer Protocol
HsMM	Hidden semi Markov Model
ICA	Independent Component Analysis
ICMP	Internet Control Messaging Protocol
PCA	Principal Component Analysis
PTA	Probabilistic Timed Automata
SIP	Session Initiation Protocol
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol

CHAPTER 1

INTRODUCTION

Internet usage across the world has increased tremendously in the past few years. According to reports from the International Telecommunications Union (ITU), over 4 billion people around the world are using the Internet for their day to day activities (Union 2018). This number accounts for more than half of the world population. Businesses and organizations have been quick to capitalize on this trend by extending their presence online, and by making their services available online. E-commerce companies are the front-runners in this category. In 2019, around 14% of global retail sales were performed over the Internet. This number is expected to grow quickly in the coming years, and it is expected that by 2040, around 95% of purchases will be performed online (Statista 2019). However, it is not just e-commerce companies that have capitalized on the Internet boom. Government agencies have also began to extend their services online. The use of Information and Communication Technology (ICT) to support public services and government administration is popularly called E-Governance, and represents a significant change in public administration. The use of web applications to power their services essentially means that organizations and businesses can extend their services to customers 24x7, irrespective of time or space constraints.

However, this open architecture of a web application poses significant challenges as well. Such an open architecture also allows unfettered access to the web application for malicious users. Attacks against web applications have also grown in direct proportion to the boom in Internet usage. Sonicwall (2019) estimates that attacks against

web applications increased by 56% in 2018. The motivation behind these attacks can be broadly classified into three categories - economically motivated, politically motivated, and espionage (Gandhi et al. 2011). Majority of the attacks against web applications are economically motivated, where attackers seek direct or indirect financial gain. Credit card fraud, ATM skimming attacks, information disclosure attacks and various instances of ransomware (O'Gorman and McDonald 2012; Scaife et al. 2016) are examples of economically motivated attacks. Around 30% of cyber attacks are politically motivated and are often signs of protest against political actions, laws, public documents, or outrage against acts related to physical violence (Gandhi et al. 2011). Espionage using the cyber domain is often much more subtle, and encompasses attempts by governments to learn sensitive information related to other countries.

From a technical standpoint, attacks against web applications violate one or more of the fundamental principles of information security - Confidentiality, Integrity and Availability (often called the CIA or AIC triad). Majority of the attacks are designed to compromise either the confidentiality or integrity of the web application. Online banking frauds compromise the integrity and sensitive information disclosure attacks compromise the confidentiality of the web application. However, there are a large number of attacks that attempt to compromise the availability of a web application as well. In other words, the primary aim of these attacks is simply to disrupt the services provided by the web application. This denies legitimate users the opportunity to avail the services provided by the web application, and hence, these attacks are aptly named as Denial of Service (DoS) Attacks.

1.1 DISTRIBUTED DENIAL OF SERVICE (DDOS) ATTACKS

DoS attacks can be executed in a variety of ways - by exploiting vulnerabilities in the target systems, by exploiting vulnerabilities in the underlying communication protocol, or by simply exhausting server resources. A single malicious client is usually unable to cause a Denial of Service unless a vulnerability is exploited. However, these attacks are isolated incidents, and cease to be a cause of concern as soon as the vulnerability is patched. Attacks targetting server resources, on the other hand, cannot be prevented by

any means. They can be executed on any system, no matter how secure, at any point of time. A single malicious client, however, is unlikely to be able to exhaust the resources of a server on its own. More often than not, denial of service attacks are executed using hundreds or thousands of malicious clients joining forces to target a single server and are called Distributed Denial of Service (DDoS) attacks.

In the 18 months from January 2015 to June 2016, Arbor networks tracked around 1,24,000 DDoS attacks every week (Networks 2016). Most of these attacks are executed with social or political motivations, such as expressing dissent or displeasure against a particular organization, government or law. The attacks against Latvian government websites in 2007 and against Iranian government websites in 2009 are prime examples of politically motivated DoS attacks (Nazario 2009). More recent instances include the attacks against the Spanish government in support of Catalan independence (Magazine 2017) and against the governments of USA (Incapsula 2015), Ireland (Silicon 2017), India (Register 2012) and Brazil (Corero 2016) for a variety of reasons. These attacks demonstrate a serious lack of security in a number of government websites, which also hold a large amount of sensitive information.

Banks have also been prime targets of DDoS attacks in the past decade, notably in USA (Networks 2012) and UK (Guardian 2016a). These attacks are particularly disconcerting at a time where the general public is becoming more inclined to purchasing online. Bitcoin websites have also been targeted in the same light as banking websites (Coindesk 2017; Forbes 2014), often calling into question the feasibility of a currency with no physical existence.

DDoS attacks are often considered as targetted attacks, focusing on a single web application. However, that is not always the case and DDoS attacks are capable of disrupting a number of websites at once. This was demonstrated during an attack on the Dyn DNS servers in 2016, which simultaneously took down a number of high profile websites including Twitter and Reddit (Dyn 2016). On a more dangerous level, DDoS attacks can even disrupt Internet services in an entire country as observed during an attack on the Internet backbone in Liberia (Guardian 2016b).

Table 1.1: Impact of DDoS Attacks

Target	**Type of Organization**	**Year**	**Impact**
Github (Register 2018)	Hosting Service Provider	Mar 2018	Github servers were taken offline
Brazilian Sports Ministry Sites (Corero 2016)	Government websites	2016	Sites for the Rio Olympics and associated sites went down
Spanish Constitutional Court Website (Magazine 2017)	Government website	2017	Several websites were taken down or hacked
Bitcoin Gold (Coindesk 2017)	Cryptocurrency server	Oct 2017	Cryptocurrency deals went down
HSBC (Guardian 2016a)	Banking website	Jan 2016	Financial transactions were blocked for a long time
Bank of America, Chase, WellsFargo, PNC (Networks 2012)	US based banks	2012	Financial transactions were unavailable to the general public
Dyn (Dyn 2016)	DNS Provider	Oct 2016	Websites like Twitter and Reddit went down
Valtia (Times 2016)	Heating Systems Provider	Oct 2016	Temperature adjustment failed in a town in Finland
Trafikverket (SCMagazine 2017)	Sweden Transport Administration	Oct 2017	Trains were delayed, reservations portals failed to function
Lonestar Cell MTN (Guardian 2016b)	Internet Provider in Liberia	Nov 2016	Internet was unavailable in parts of the country

Recently, there have been a number of attacks on critical infrastructure such as heating systems in Finland (Times 2016) and transport systems in Sweden (SCMagazine 2017). A study by Corero (Guardian 2017) provides additional cause of concern, as it reveals that more than half of the critical infrastructure organizations in the UK are

blatantly ignoring the risk of DDoS attacks. Table 2.2 gives a summary of some of the critical DDoS attacks in the last decade.

1.2 ANATOMY OF A DDOS ATTACK

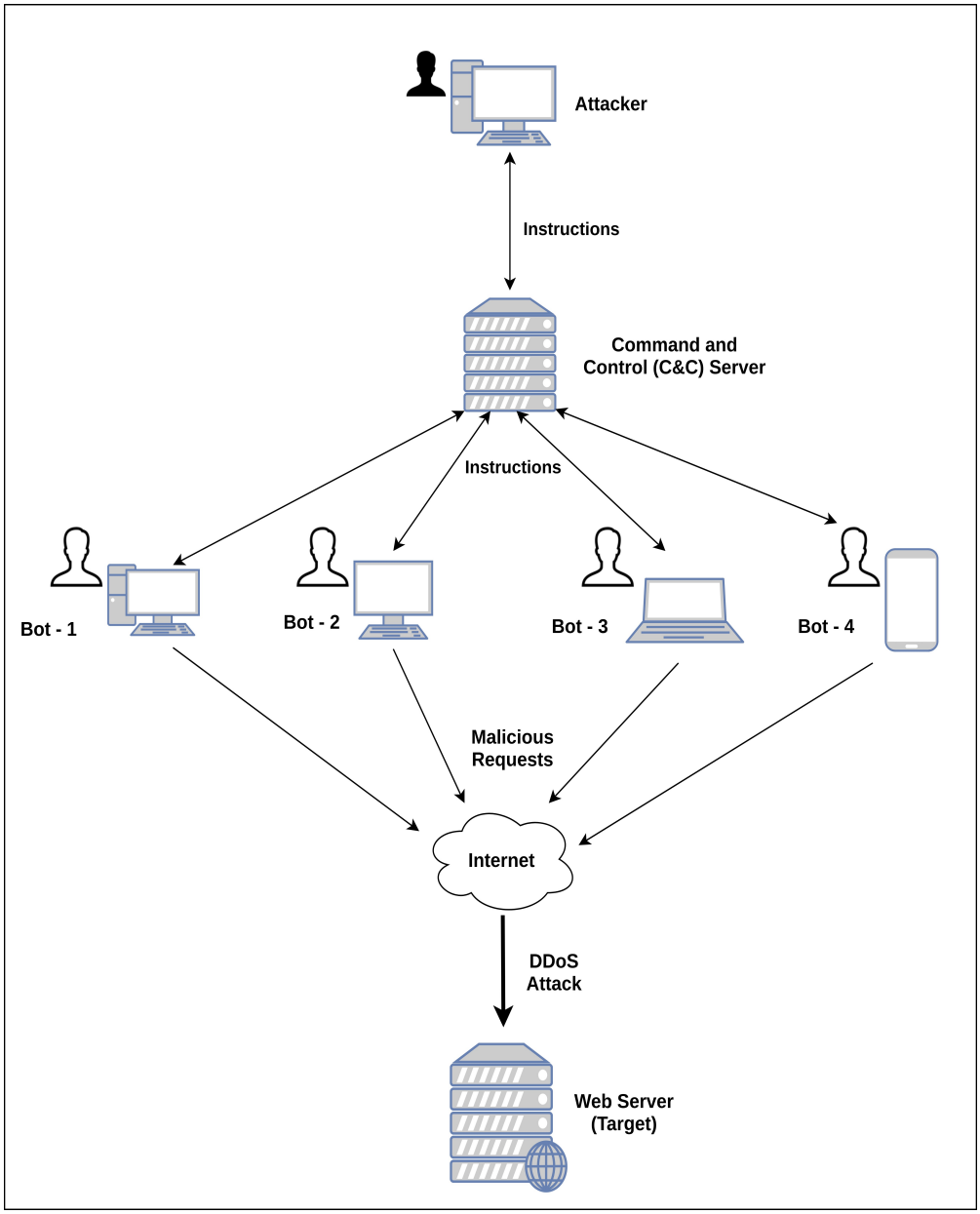


Figure 1.1: Typical Botnet Structure

Most of the time, a DDoS attack is launched from a large number of compromised systems located at physically distinct locations. These systems, called zombies or bots, are often infected with malware and act on the attackers volition. This entire setup is

called a botnet, and has become an indispensable part of modern-day DDoS attacks. The attacker who wants to bring down a target server often never physically participates in the attack. The attacker communicates with the botnet through a Command and Control (C&C) Server. The C&C server periodically communicates with the bots to keep track of their status and also to transfer instructions and attack scripts. The C&C servers used to reside on a physical system under the control of the attacker and could remain alive for years. However, nowadays, C&C servers reside on legitimate cloud servers and use automatic domain generation algorithms to evade detection. Consequently, they have a very short shelf life. The C&C server communicates with the bots which have been infected with rootkits or other malware and use them to launch attacks against a target web server. Often the users of the infected systems have no idea they are unknowingly participating in a DDoS attack.

Two points are worth mentioning in the context of a botnet-based DDoS attack.

1. The success of an attack is determined by a combination of two factors - the size of the botnet and the potency of the attack.

2. Since all zombie clients in a botnet follow the same script, which is supplied by the attacker through the C&C server, the traffic patterns of individual clients involved in a DDoS attacks tend to be extremely similar as well. This similarity could potentially be used to identify DDoS attacks.

1.3 TYPES OF DDOS ATTACKS

In the early days of the Internet, DDoS attacks were executed by using a large volume of network layer packets (UDP or ICMP requests) to choke network layer devices such as routers. These attacks work by exhausting the network bandwidth and choking network layer devices such as routers. This leads to service degradation, and in extreme cases, can lead to service outage. Network layer DDoS attacks pose a significant challenge even today, but their impact has been significantly reduced due to two major factors. Firstly, the proliferation of broadband services and fibre optic technology has led to bandwidth becoming cheaper in general. This allows servers to have larger bandwidths at cheaper rates. Since network layer DDoS attacks work by exhausting network band-

Table 1.2: Comparison of Network Layer and Application Layer DDoS Attacks

	Network Layer DDoS Attack	**Application Layer DDoS Attack**
Layer of Attack	Layer 3	Layer 7
Attack Payload	Network Layer Packets (UDP, ICMP etc.)	Application Layer Requests (HTTP, SOAP etc.)
Type of Request	Malformed or Legitimate	Legitimate
Attack Volume	Large	Low
Attack Target	Network bandwidth and Infrastructure	Server resources such as CPU, database, memory, socket connections etc.

width, they need to generate exceedingly large traffic volumes in order to be effective. Secondly, significant research has been done on detecting network layer DDoS attacks and this is evidenced by the large number of network layer firewalls with DDoS mitigation capabilities available in the market. There has also been a spurt in the number of cloud platforms which offer DDoS mitigation as a service. A combination of these two factors means that it is now becoming increasingly difficult for attackers to successfully launch network layer DDoS attacks.

With the success of network layer DDoS attacks no longer guaranteed, attackers started turning to alternate avenues. In the past decade, there has been a surge in the number of DDoS attacks executed at the application layer (Incapsula 2016, 2017; Kaspersky 2016). These attacks use application layer protocols such as HTTP for launching attacks, and are consequently called Application Layer DDoS (AL-DDoS) attacks. Instead of targetting the network bandwidth, AL-DDoS attacks attempt to exhaust server resources such as CPU or database cycles, memory or socket connections. They are executed using legitimate application layer requests, which makes them extremely difficult to detect. The major differences between network layer and application layer DDoS attacks are encapsulated in Table 1.2.

1.4 FEATURES OF APPLICATION LAYER DDOS ATTACKS

Application layer DDoS attacks have certain special features that make them unique and also make their detection comparatively difficult.

- **Legitimate Requests**: Application layer DDoS attacks are executed using legitimate requests. This means that a detection mechanism cannot determine whether an incoming request stream is malicious or not by simply inspecting the request.

- **Low Attack Volume**: Application layer DDoS attacks can be executed with comparatively low attack volume. This is partly due o the fact that the target of these attacks are server resources like CPU or database cycles, memory or socket connections. Since a single HTTP request can make the server perform more work than network layer packets of the same size, attacks at the application layer can be executed with lower bandwidth. The average attack volume for network layer DDoS attacks is around 14 GBps while the maximum attack volume ever recorded for an application layer DDoS attacks is around 8 GBps. Typical application layer DDoS attacks incur an attack volume of just a few MBps which is unlikely to trigger any existing DDoS detection mechanisms.

- **Targetted Strikes**: While network layer DDoS attacks have a single target which is the network bandwidth, application layer DDoS attacks can target any of the resources at the server side such as CPU or database cycles, memory or socket connections. A detection mechanism against one of these attacks is unlikely to be of any help during the other attacks.

- **Resemblance to Flash Crowds**: Application layer DDoS attack traffic bears a close resemblance to legitimate user traffic. In particular, they closely resemble an Internet phenomenon called a Flash Crowd. A Flash Crowd refers to a sudden spike in the requests coming to a web server due to a noteworthy event or a large sale. These requests come from legitimate users and bring useful revenue to the web application. Since both flash crowds and application layer DDoS attacks are usually characterized by a sudden increase in the number of requests coming from different geographic locations, they are often mistaken for one another. This means that any detection mechanism has to be extremely cautious when blocking a user because blocking out users during a flash crowd leads to an enormous loss of revenue.

1.5 TYPES OF AL-DDOS ATTACKS

There are a large number of protocols available at the application layer, using which attackers can execute DDoS attacks. However, the overwhelming majority of AL-DDoS attacks are executed using the HTTP protocol due to its ubiquitous nature. Attacks using the HTTP protocol can be broadly classified into three categories:

- **Slow DDoS Attacks**: Slow attacks form a class of attacks that exploit a vulnerability in the HTTP protocol that allows requests to be split into multiple pieces. A server is required to hold a connection open till the entire request is received, or a timeout occurs. Attackers can force servers to keep connections open by sending a small chunk of data at specially timed intervals to avoid a timeout. When executed on multiple connections, the server socket pool exhausts, and the server cannot accept any more connections.

- **HTTP Request Flooding**: This attack goes back to the concept of flooding. The attacker sends a large number of requests to the server, thereby tying up its resources. HTTP flooding is one of the most prominent variations of DDoS attacks executed at the application layer.

- **Asymmetric Attacks**: Most servers have a number of functions, not all of which are computationally intensive. Normal users typically switch between computationally intensive tasks and non-intensive tasks. In order to execute an asymmetric attack, the attacker sends a number of computationally intensive tasks for the server to perform, thereby exhausting its resources. There is a variation of the asymmetric attack wherein the attacker connects with the server through multiple sessions, each running a computationally intensive task. This effectively ties up the server from accepting new connections, and cannot close the existing connections with the task running. This attack is called the Repeated One Shot Attack.

1.6 HTTP/2 AND ASSOCIATED CHALLENGES

For decades, HTTP/1.1 has been the standard of web communication. In recent years, however, a number of websites faced performance issues while using HTTP/1.1 due

to the way in which the protocol was designed. HTTP/1.1 was designed in the initial days of the Internet, and was tailor-made for static websites. Modern websites, on the other hand, are dynamic with multiple inline resources embedded within every webpage. These websites faced various performance issues, most notable of which was Head-of-Line (HoL) blocking. The HTTP/1.1 protocol specifies that only a single request from a client will be processed by a server at any point of time. This feature severely affects the performance of dynamic websites which require multiple inline requests (such as images, styling sheets and script files) to be made before a page can be rendered. A computationally intensive or long running request could potentially hold up the entire connection and prevent the page from loading. This is called HoL blocking.

The standardization of HTTP/2 in 2015 marked an important change in the Internet. HTTP/2 was designed to improve the efficiency of the HTTP protocol by reducing the page load time for clients. Some of the significant changes made in the new version of the protocol include the use of a new binary framing system, the introduction of header compression, multiplexing and server push. The introduction of multiplexing allows users to bundle multiple HTTP requests into a single TCP connection, thus reducing network bandwidth considerably. Server Push on the other hand, allows the server to send inline resources to the client without an explicit request. A combination of multiplexing and header compression has led to a drastic reduction in request sizes in HTTP/2, leading to lower bandwidth utilization. These modifications have helped in reducing page loading latency for most clients, particularly in the case of dynamic web applications. de Saxc et al. (2015) observed that the use of multiplexing alone improves the page load time up to 40% in some cases. This improvement in the Quality of Service for clients has led to the fast adoption of HTTP/2 by web servers and browsers. Most of the modern browsers provide support for HTTP/2. Currently HTTP/2 is used by 42.9% of all the websites (Statistics 2020).

While HTTP/2 has been effective in improving the user experience, it has also faced criticism on the following counts:

- Bursty Traffic : HTTP/2 request traffic is fundamentally bursty. For the same request rate, the steady flow of requests in HTTP/1.1 has now been replaced by short bursts of a large number of requests. This necessitates web servers to be over-provisioned to handle much more requests concurrently than was previously required (McCombs 2019) and makes an HTTP/2 server vulnerable to DoS (or DDoS) attacks (Adi et al. 2015).

- Enhanced Request Generation Capabilities for Clients : Multiplexing and Header Compression in HTTP/2 have led to a reduction in the size of individual requests, which allows users to send more requests with the same bandwidth and packet generation capability. This in turn allows attackers to launch DDoS attacks more effectively (Beckett and Sezer 2017a).

- Misuse of Newly Introduced Features : HTTP/2 has introduced a number of features intended to improve user experience and reduce response times. However, most of these features can be misused in order to launch attacks against the web server in the absence of proper validation. For example, HTTP/2 allows users to specify a priority order for requests in order to improve the Quality of Service (QoS). This feature was introduced with a genuine intention of improving user experience, but Imperva identified that this feature can be misused in order to generate a DoS attack at the server (Imperva 2016).

The introduction and rapid adoption of HTTP/2 thus adds a layer of complexity to the detection of asymmetric AL-DDoS attacks. On one hand, there is a need for research to corroborate the observation that HTTP/2 servers are more vulnerable to AL-DDoS attacks. On the other hand, there is further need for research into the newly introduced features in HTTP/2, and whether they can be misused to launch lethal AL-DDoS attacks.

1.7 THE COMPLEXITY OF AL-DDOS DETECTION

1.7.1 DDoS Detection as an Intrusion Detection Problem

AL-DDoS detection is fundamentally an intrusion detection problem and can be carried out using two major approaches - signature based and anomaly based (David and Thomas 2015). Signature based detection approaches look for signatures or attributes of known attacks within a request. Signature based approaches are extremely fast, simple to use and produce few false positives. For network layer DDoS attacks which operate using malformed network layer packets, signature based DDoS detection can be used effectively. However, in the case of network layer floods and AL-DDoS attacks, it is not a viable option due to the absence of a well-defined signature in the attack requests. In such cases, signature based detection approaches are ineffective.

Anomaly detection approaches, on the other hand, build a model of legitimate user behaviour, and then look for deviations from this model to identify attacks. This approach proves to be extremely effective and flexible, but does suffer from certain drawbacks. Anomaly based DDoS detection techniques tend to be slower and more prone to false positives.

In recent years, a new, hybrid approach that combines the positive features of both signature based detection and anomaly based detection have come up. These detection approaches are called dynamic signature based approaches. They work by first employing an anomaly detection mechanism for identifying attacks, and then building a signature to represent the attack. Future incidences of this attack can now be effectively identified using the attack signatures instead of the anomaly detection, which makes the detection process faster.

1.7.2 Anomaly Detection Approaches for AL-DDoS Detection

Existing approaches for AL-DDoS detection follow an anomaly detection approach wherein a model of legitimate user behaviour is constructed, and any deviation from this learned model is deemed to be malicious and treated as an attack. However, legitimate user behaviour is not accurately defined, which has led to researchers adopting their own definitions and features to describe it. A large number of research works use features

such as request rate and request inter-arrival time etc. as indicators of user behaviour. In reality, though, a user does not consciously set his request rate or decide upon his request inter-arrival time. These features are indirect representations of user behaviour and hence cannot effectively detect AL-DDoS attacks. This is clearly indicated by the fact that research works that use indirect representations of user behaviour are capable of detecting HTTP floods, but are unable to detect asymmetric attacks. Research works that are capable of detecting asymmetric attacks model the request sequence of users in some way, notably using a Hidden semi Markov Model (HsMM).

Existing detection approaches for asymmetric attacks use complex modelling techniques and indirect representations of user behaviour which leads to a higher false positive rate (FPR) and longer detection time. Apart from that, they are unable to incorporate gradual changes in user behaviour and require periodic retraining. This adds to the overhead of the system and leads to additional downtime. A lack of adaptability, on the other hand, leads to a higher false positive rate due to the model becoming obsolete over time. There is a need for a lightweight and fast detection approach for asymmetric AL-DDoS attacks which has the ability to incrementally learn at run time.

1.8 THE QUEST FOR EARLY AL-DDOS DETECTION

Early detection of DDoS attacks has been a key research issue since the beginning. However, the concept is often vaguely defined without proper explanation. What constitutes as early detection depends heavily on the network architecture and infrastructure. Often, early detection simply means the ability to detect a DDoS attack before it causes significant damage to the web server or the network infrastructure.

A large amount of work has been done concerning early detection of network layer DDoS attacks. Some of these approaches use more sophisticated features which leads to early detection (Behal et al. 2018), while a few others enable early detection by setting suitable threshold values for attack detection (Xiang et al. 2011). However, majority of the early DDoS detection approaches use a collaborative approach (Chen et al. 2007; Kaushal and Sahni 2016; Yu and Zhou 2008; Yu et al. 2008; Yu Chen and Kai Hwang 2006). A collaborative approach works by pooling the knowledge acquired

by multiple network layer devices and end-users. By the time a network layer DDoS attack reaches an end-user, it has assumed massive proportions, and becomes extremely difficult to mitigate. A better strategy is to identify comparatively smaller surges in network traffic at intermediate devices, and combine that knowledge to decide whether a DDoS attack could be impending. A collaborative approach allows for the excess traffic to be curtailed at the intermediate routers so that the DDoS attack does not reach its full potential.

However, such an approach is unsuitable for the detection of AL-DDoS attacks due to three reasons. First, the traffic volume associated with an AL-DDoS attack is comparable to legitimate user traffic, and hence cannot be used as a measure of attack. Second, intermediate devices like routers cannot inspect application layer requests, and hence AL-DDoS detection cannot be performed at routers. Third, AL-DDoS attacks are tailor-made for a web application. This means that a collaborative approach is unlikely to be of much help in mitigating an AL-DDoS attack. In the absence of a collaborative approach, certain global features could be used to enable the early detection of AL-DDoS attacks at web servers. As discussed previously in Section 1.2, DDoS attacks executed by botnets have a considerable amount of similarity between individual attacking connections. This holds true even in the case of AL-DDoS attacks. If this similarity can be identified and forged into an attack signature, a dynamic signature based approach could be used to block attacking connections at a much earlier stage than usual. However, the signature used to represent an attack must be carefully formed to avoid unnecessary false positives, which would affect the system performance in a negative way.

1.9 MOTIVATION

DDoS attacks are one of the most dangerous attacks against web applications today. The effortless availability of DDoS attack generation tools, the ability to hire botnets at cheap rates (Kandula et al. 2005) and the rise of DDoS-for-hire services (Karami et al. 2016; Santanna et al. 2016) has led to a steep increase in the number of DDoS attacks in recent years. In addition, DDoS attacks have become more sophisticated in recent years. The use of computationally expensive application layer requests to launch

asymmetric AL-DDoS attacks has made most of the existing detection mechanisms obsolete. There is an urgent need for efficient detection mechanisms for asymmetric AL-DDoS attacks.

A fair amount of research has been done on asymmetric AL-DDoS detection in the past decade. Anomaly detection techniques are commonly used for this purpose, by modelling legitimate user behaviour. However, most of the existing detection mechanisms use indirect representations of user behaviour and complex modelling techniques which leads to a higher false positive rate and longer detection time. An abnormally long detection time is unacceptable for asymmetric AL-DDoS attacks, which are capable of causing damage to web servers with relatively few requests. In addition, the use of complex modelling techniques make the model difficult to update, which further increases the false positive rate.

The introduction of HTTP/2 has added another layer of complexity of asymmetric AL-DDoS detection. HTTP/2 has been successful in reducing the page load time for clients through the introduction of features like binary framing, header compression, multiplexing and server push. However, there are concerns that the additional server load that arises due to these features could make an HTTP/2 server more vulnerable to DDoS attacks, especially asymmetric attacks. However, no study has been conducted so far to investigate the behaviour of an HTTP/2 server under an asymmetric attack. Apart from that, there have been no studies about whether the newly introduced features in HTTP/2 can be misused to launch potentially lethal AL-DDoS attacks.

It is of utmost importance that AL-DDoS attacks be detected early as possible due to their ability to compromise the availability of a server with comparatively fewer requests. While there are a large number of research works that deal with early DDoS detection at the network layer, similar research at the application layer is comparatively rare. This is due to the fact that a collaborative detection approach is unlikely to be successful due to the obscure and highly targetted nature of AL-DDoS attacks. However, DDoS attacks display a considerable amount of similarity with each other due to the use of common attack generation tools and botnets. This similarity can be exploited to enable the early detection of AL-DDoS attacks using a dynamic signature

based approach.

This work is primarily concerned with the efficient and fast detection of asymmetric AL-DDoS attacks. Our motivation for carrying out this study is threefold. First, given the potency of asymmetric AL-DDoS attacks, there is an urgent need for a fast, lightweight and adaptable detection mechanism which has a low false positive rate. Second, there is a need to extend the envelope of asymmetric attacks to cover attacks executed using the HTTP/2 protocol as well, related to which no study has been conducted till now. There is also a need to extend the detection mechanisms for asymmetric attacks to be able to handle the unique challenges presented by the HTTP/2 protocol. Thirdly, there is a need for mechanisms that enable the early detection of AL-DDoS attacks in general and asymmetric attacks in particular.

1.10 ORGANIZATION OF THE THESIS

The rest of the thesis is organized as follows. Chapter 2 provides a taxonomy of AL-DDoS attacks executed using the HTTP protocol along with a detailed survey of existing research work that deals with the detection of HTTP based AL-DDoS attacks. An introduction into the HTTP/2 protocol along with the associated security challenges are also provided. In addition, the chapter provides a list of research challenges that exist in the domain. Chapter 3 describes the research problem that is the focus of this thesis. Chapter 4 describes how asymmetric AL-DDoS attacks are executed using the HTTP/1.1 protocol, and describes our proposed approach for detecting these attacks efficiently. The chapter also gives the experimental analysis of our approach in terms of efficiency and running time. Chapter 5 discusses how asymmetric attacks affect the servers running the HTTP/2 protocol. The chapter also discusses how the new features in HTTP/2 exacerbate the problem of asymmetric attacks, and how our proposed approach can be extended to detect these attacks. An experimental evaluation of our approach is also presented. Chapter 6 discusses how the similarity between AL-DDoS traffic can be used to enable the early detection of attacks. The chapter describes our proposed approach for designing an Early Detection Module (EDM) for AL-DDoS attacks. Experimental results regarding the detection time with and without the EDM is

also presented. Finally, chapter 7 presents a summary of the research work presented in this thesis and suggests some future research directions.

CHAPTER 2

LITERATURE REVIEW

Majority of the literature related to DDoS attacks is concerned with attacks at the network layer. This is due to the fact that network layer DDoS still continue to be a popular avenue of attack, despite the numerous defenses available. However, given adequate resources, majority of the network layer DDoS attacks are relatively easy to defend against. Application Layer DDoS attacks, on the other hand, pose a more significant challenge. These attacks started gaining public attention only in the past decade. In fact, the first mention of application layer DDoS attacks only appeared in literature in 2009 (Ranjan et al. 2009). Since then, there have been a large number of research works which attempt to understand and propose defense mechanisms against these attacks. The sophisticated nature of these attacks, coupled with the variety of attacks that can be executed at the application layer, often makes detection difficult.

The application layer has comparatively more protocols that the network or transport layers, which increases the attack surface considerably. Out of the fifty odd protocols at the application layer, four of the most popular protocols are HTTP, SOAP, DNS and SIP. Of these, HTTP is the most ubiquitous, and facilitates communication between end users and web servers. Needless to say, the entire notion of the Internet is unfathomable without the HTTP protocol. The SOAP protocol performs a similar task, except that it facilitates communication between web servers. DNS is an equally important protocol, which helps in the resolution of domain names to IP addresses. SIP is a relative newcomer in the list of application layer protocols and facilitates the exchange of voice,

video and messaging applications over the Internet. Even though DDoS attacks have been reported using all of these protocols, a closer inspection reveals that majority of the AL-DDoS attacks are carried out using the HTTP protocol (Kaspersky 2016). For this reason, our primary focus in this work relates to DDoS attacks using the HTTP protocol.

HTTP/1.1 has been the undisputed standard of web communication since its inception in 1997. Consequently, most of the research work related to the HTTP protocol consider the HTTP/1.1 version of the protocol. In 2015, a more efficient and fast version of the HTTP protocol, named HTTP/2 was standardized, and has slowly started replacing its predecessor. Currently, around 42.9% of web servers support HTTP/2. Research into DDoS attacks using the HTTP/2 protocol is still in its infancy. However, it must be noted that most of the DDoS attacks that can be executed using the HTTP/1.1 protocol can also be executed using the HTTP/2 protocol. In addition, some of the newer features in HTTP/2 can be used to launch DDoS attacks specific to the HTTP/2 protocol as well.

The rest of this chapter is organized as follows. Section 2.1 presents a taxonomy of AL-DDoS attacks using the HTTP protocol. Section 2.2 presents a detailed survey of the different approaches that are commonly used for detecting AL-DDoS attacks. Section 2.3 presents an introduction to the HTTP/2 protocol and discusses the security challenges that arise due to its introduction. Section 2.4 presents a list of research challenges that has to be addressed in order to develop effective detection mechanisms for AL-DDoS detection.

2.1 TAXONOMY OF APPLICATION LAYER DDOS ATTACKS USING THE HTTP/1.1 PROTOCOL

Attacks at the application layer are many, varied and ever-changing. The existence of a multitude of application layer protocols, combined with the myriad number of ways in which these protocols can be used lead to a large variety of DDoS attacks at the application layer. In this section, we present a taxonomy of AL-DDoS attacks executed using the HTTP protocol. For building this taxonomy, the nature of exploitation was chosen

as the major criterion for dividing AL-DDoS attacks into different classes. Based on the nature of exploitation, attacks at the application layer can be grouped into three classes - attacks which exploit application vulnerabilities, attacks which exploit protocol features and attacks which exploit system features.

2.1.1 AL-DDoS Attacks by Exploiting Application Vulnerabilities

Acunetix (2019) reports that around 87% of web applications have medium level security vulnerabilities and over 40% suffer from severe security vulnerabilities. These vulnerabilities can arise due to a number of reasons, such as the use of vulnerable software components, the use/reuse of vulnerable algorithms without patching or due to programmer negligence. Web applications are rarely designed and built from scratch. Web developers always use existing software components or algorithms without any modifications. The use of components or algorithms with security vulnerabilities makes the entire web application insecure. As an example, earlier versions of the Apache server suffered from a vulnerability due to which an HTTP message with a large overlapping range header caused a memory exhaustion and crashed the server Apache (2011). This attack is no longer feasible, because newer versions of the server have patched this vulnerability. HashDoS (CCC 2011) was an attack that exploited the use of vulnerable hashing algorithms in web application servers that use hashing to organize POST input parameters. In both the best and average cases, a good hashing algorithm works in $O(1)$ complexity, but deteriorates to $O(n)$ when collisions occur. Attackers can successfully cause collisions in a large number of scripting languages by supplying crafted inputs with a large number of POST parameters. This can cause the CPU to spend a large amount of time working to resolve collisions and cause a denial of service situation (Crosby and Wallach 2003).

Even when all the code for a module are written by web application developers from scratch, there are chances of vulnerabilities creeping in due to programmer negligence. This usually manifests in the form of missing or imperfect security checks, and can lead to a variety of attacks, including injection attacks (Boyd and Keromytis 2004; Deepa et al. 2017) and access control violations (Deepa et al. 2018).

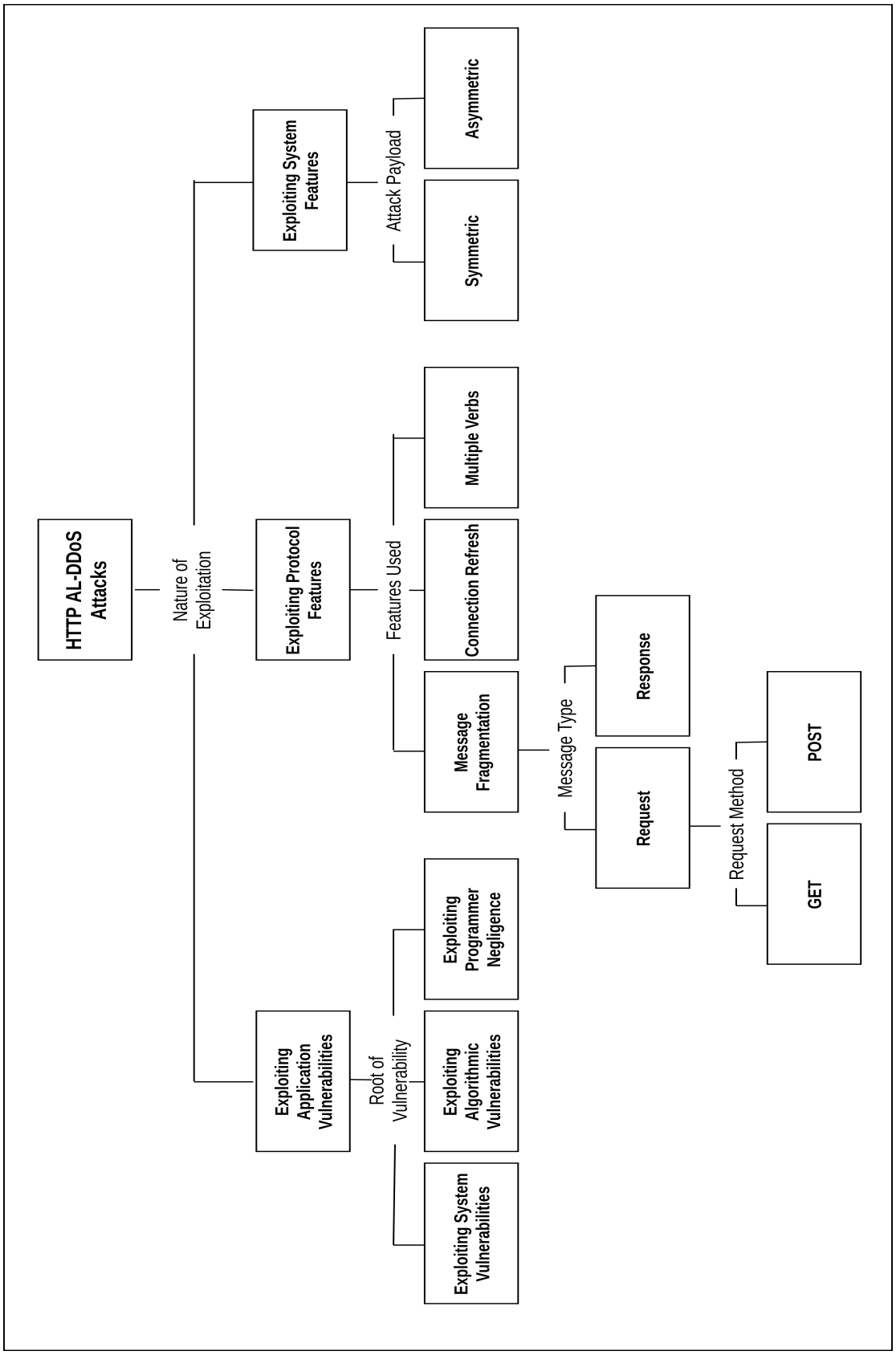


Figure 2.1: Taxonomy of AL-DDoS Attacks using the HTTP/1.1 Protocol

2.1.2 Exploiting Protocol Features

Protocols are designed to facilitate efficient communication between different parties regardless of differences in bandwidth or computing power. Attackers often misuse protocol features in order to launch attacks against web applications. HTTP was designed to facilitate communication between a human user (using a web browser) and a web server. It is a connection oriented protocol based on TCP, which means a TCP connection should be established before communication can proceed. This connection is maintained till the end of communication. Different features of HTTP have often been abused by attackers to launch attacks.

Message Fragmentation: HTTP allows its users to fragment an HTTP message across multiple packets in an attempt to be accessible even for users who possess limited bandwidth. An attacker who fragments his HTTP messages into extremely small packets can keep the connection open for an arbitrarily long time. Since web applications have a predefined limit on socket connections they can maintain simultaneously, an attacker who manages to keep multiple connections open for an infinitely long time can effectively force the server to decline legitimate connections. These class of attacks are called Slow DDoS attacks, and can be executed using both HTTP requests and responses.

The attacks that make use of an HTTP request are called Slow Write attacks. They are usually carried out by fragmenting an HTTP requests into small fragments and sending them slowly to the web server. One of the most famous Slow DDoS tools is called Slowloris, and was used in several politically motivated DDoS attacks after the 2009 elections in Iran.

In order to maintain the connection indefinitely, an attacker has to create the illusion that the HTTP request hasn't been completed. In reality, this is relatively simple to replicate. HTTP uses a carriage return line feed (CRLF) to denote next line and it uses two CRLF characters to denote a blank line. A blank line is used to denote the end of headers, and consequently, the end of an HTTP GET request as shown below:

```
GET /index.php HTTP/1.1[CRLF]
Pragma: no-cache[CRLF]
Cache-Control: no-cache[CRLF]
Host: testphp.vulnweb.com[CRLF]
Connection: Keep-alive[CRLF]
Accept-Encoding: gzip, deflate[CRLF]
User-Agent: Mozilla/5.0
(Windows NT 6.1; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/28.0.1500.63 Safari/537.36[CRLF]
Accept: */*[CRLF][CRLF]
```

An attacker could simply omit sending this blank line and continue sending fragments of headers in order to keep the connection open. The attacker needs to send fragments of the request periodically in order to avoid a connection timeout. This value is usually server dependent, and could be of the order of a few minutes. Once the attacker deduces the timeout value and sends request fragments at regular intervals to avoid the timeout, the connection can, theoretically, be maintained indefinitely. If executed over multiple connections the server runs out of socket connections for legitimate users.

A GET request is not a particularly enticing option for executing a slow write attack due to its size limitation. The absence of a request body creates an upper bound on the size of a GET request, and hence, puts an upper limit on the amount of time the connection can be maintained by an attacker. A POST request, on the other hand, has technically no size limitations. The HTTP header specifies a field called $Content-Length$ which tells the server how long the message is going to be. The attacker sets the POST request to have an arbitrarily large value of $Content-Length$ and then proceeds to send data in small fragments. The server is forced to maintain the connection until either the connection times out or the entire message is received. After a while the server is unable to accept any new incoming connections.

Slow DDoS attacks can also be launched on the HTTP response and are called Slow Read Attacks. The attacker sends a GET or POST request to the server and waits for a response by advertising a small network window. The web server assumes the user is on a low bandwidth connection and proceeds to send the HTTP response in small fragments. This ties up the connection until the entire data is received. If the attacker can establish multiple connections of this nature, it exhausts the socket connections on the server. Such an attack is much more effective because an HTTP response is often many orders of magnitude larger than a request, and can hence maintain a connection for a much longer duration.

Connection Refresh: The effect of maintaining a connection for an undefined amount of time can be achieved by using an HTTP header field called PRAGMA (Dantas et al. 2014). The HTTP PRAGMA header field tells the HTTP server and any intermediate caches that the user wants a fresh copy of the requested resource. This ensures that the request is not satisfied by any of the caches but by the web server itself, ensuring that the server has to expend processing power. Additionally, whenever an HTTP PRAGMA is issued the timeout for the connection is reset. Use of the PRAGMA field can keep connections open for an indeterminate amount of time, ensuring that socket resources get tied up. It is worth mentioning that the PRAGMA field has no visible purpose in HTTP/1.1, but is still allowed to maintain compatibility.

Multiple Verbs: HTTP request methods initiate action at the server side, and for this reason they are sometimes called HTTP verbs. Traditionally, a single request consists of a single action or verb. In such a scenario, it is rudimentary that the more workload the attackers want to dump on the server, the more number of requests they have to send. But HTTP has a somewhat lesser known feature which allows the users to pack multiple verbs into a single HTTP request. This means that attackers can compress multiple verbs into a single request and send to the server. The server is forced to perform all the tasks that are requested, which is much more than a normal request. The advantage this presents to the attackers is that the attack volume can be cut down to a large extent, thus helping them evade detection (DDoS-Guard 2014; Zargar et al. 2013).

2.1.3 Exploiting System Features

The startling aspect of DDoS attacks is that they can be executed against any web server, and do not need to exploit any vulnerability in the server. While the presence of vulnerabilities certainly makes it easier to launch DDoS attacks (as explained in Section 2.1.1), it is not a necessary condition. DDoS attacks can work by simply sending a large enough number of requests to exhaust the resources at the server, after which the server becomes unable to handle legitimate requests. These resources could be CPU, database, memory or socket connections. Attacks exploiting system features are extremely common, and form the bulk of documented DDoS attacks. Based on the attack payload used to launch the attacks, they can be classified into symmetric and asymmetric DDoS attacks.

Symmetric DDoS Attacks: A DDoS attack is said to be symmetric if the attacker expends as much effort in generating the attack requests as the server expends in handling them. Like network layer DDoS attacks, symmetric DDoS attacks also work by sending a large number of attack requests to the target web server, so that it is unable to process legitimate client requests. Attacks at the application layer do not need to throttle the server bandwidth to cause a denial of service situation. A single HTTP request makes the server perform more work than a packet at the network layer. So the server resources become the new bottleneck in this situation and get exhausted much before the bandwidth of the server gets throttled.

HTTP flooding (Cloudflare 2014) is the most common application layer DDoS attack which utilizes the HTTP protocol. This is largely due to the ease with which this attack can be executed. The simplest way of executing this attack is to repeatedly send requests to any one URL on the target web application. More often than not, it is the home page or login page that is attacked. However, a repeated stream of requests to the same URL can be easily identified and blacklisted by the server administrator. So the next line of attack is to continuously send requests to random URLs in the web application. This works in the same way as the earlier attack, but cannot be detected that easily. Tools like Low Orbit Ion Cannon(LOIC) or other stress testing tools can easily

be used to launch an HTTP flood. The fact that these tools can be easily downloaded off the internet makes it much more dangerous.

Asymmetric DDoS Attacks: All the symmetric attacks against the victims also consume a significant amount of resources for the attacker. The objective of the attacker is always to maximize the damage to the victim while minimizing the resources that have to be employed for the attack. Asymmetric attacks use specially crafted requests which consume more resources at the server side in order to launch attacks. This will bring down the server faster, using fewer requests and at the same time reduces traffic volume which in turn evades detection.

As an example, consider a web server that connects to a database server. The web server gives the users the option to search for jobs based on some search queries. To make the search better, the server employs pattern matching so that the database returns all jobs which have the words the user wants in any scenario. The internal query would look similar to the following:

```
SELECT TOP 10 JobPK, JobDescription
FROM JOBS as j
WHERE JobDescription LIKE '%ASP%'
ORDER BY JobPK
```

This query will typically return the result in under a second. Note that the search term the user enters is "ASP". However, if the user enters a more complex search term, the query gets complicated. Consider a slight variation of this query as given below

```
SELECT TOP 10 JobPK, JobDescription
FROM JOBS as j
WHERE JobDescription LIKE
'%_[^|?$%"*[(Z*m1_=]-%RT$)|[{34}\?_]||%TY-3(*.␣>?_![]_%'
ORDER BY JobPK
```

This query performs a complex comparison using regular expressions on the tuples and

is likely to tie up the database for some time, possibly minutes.

We refer to the class of requests that do not put excessive load on the server as low workload requests, and the requests which do have to perform significant computation as high workload requests (In reality though, there is no such distinction, and classifying requests based on the workload is purely situational). Assuming that a high workload request performs twice the computation that a low workload request does, it would take just half the amount of requests to exhaust a server. The attackers greatly benefit from sending high workload requests to the web server because it can bring the server down with fewer resources. This attack is termed as an asymmetric workload attack.

2.2 DEFENDING AGAINST DDOS ATTACKS USING THE HTTP/1.1 PROTOCOL

The increased sophistication that goes into executing an application layer DDoS attack makes it comparatively difficult to defend against these attacks. In general, there are two approaches to defend against these attacks - blocking automated requests using user puzzles, or to employ a detection mechanism to identify and block malicious users.

2.2.1 Blocking DDoS Attacks using User Puzzles

The primary cause of a denial of service attack is the ability of attackers to generate automated requests through the use of botnets or other tools. The most basic line of defense against any DDoS attack is to restrict automated requests from entering the system. This is usually achieved through the use of user puzzles. A user puzzle is any challenge that can be completed quite easily by a human user, but is considerably difficult to complete for an automated system. Simple examples of user puzzles are CAPTCHAs (Completely Automated Public Turing test to tell Computers and Humans Apart) and AYAHs (Are You A Human). Although these puzzles can be broken by bots through the use of suitable image processing algorithms (Mori and Malik 2003; Sivakorn et al. 2016; Yan and El Ahmad 2007) , this simple line of defense against DDoS attacks still works very well. This is because a majority of the DDoS attacks are simple attacks executed using available tools and do not possess the computing power required to crack these challenges.

Associated with this defensive tactic is the question of which users should be served with the challenge-response mechanism. The easiest solution is to deploy the challenge-response mechanism for all the users who want to access your system. This is, however, a relatively inefficient solution. User puzzles significantly reduce the user experience when visiting a website. Users do not want to be burdened by having to solve a puzzle every time they visit a website. The next step would be to deliver the puzzles to only a fraction of the users. SENTRY (Zhang et al. 2016) uses a moderator module which supplies a challenge to a fraction of the requests it encounters. There are provisions within the defense mechanism to sample requests based on the associated workload. Additionally, the complexity of the challenge is also proportional to the workload of the request. However, the users who will be supplied with the puzzle are completely random. Another line of thought is that it is better to serve a challenge only to users who appear to be suspicious. This raises the additional question of how to determine suspicious users. Sivabalan and Radcliffe (2013) proposed a way of detecting suspicious users based on the pages viewed and their viewing times. As long as the server load remained within limits, no user is served with a puzzle in their solution. When the server load crosses a threshold, they served suspicious users with AYAHs. This presents a calibration mechanism for their signature generation because if a user with a suspicious signature successfully solves the AYAH, the user is vindicated along with the other users who have a similar signature. On the other hand, repeated failures to solve the AYAH results in blocking of the users.

While puzzle based mechanisms like CAPTCHAs and AYAHs do work in preventing denial of service attacks, they also reduce the user experience. Hence, most of the research focus is on other approaches to solve the problem.

2.2.2 Detecting Application Layer DDoS Attacks using the HTTP Protocol

Detecting application layer DDoS attacks presents a significant challenge because of their low traffic volume and the use of legitimate requests. Rather, the complex interrelationships among a number of requests need to be modeled and studied to successfully identify an attack at the application layer. In general, attack detection can proceed

in three different ways:

1. Request Tracking : Request tracking refers to schemes which monitor how many requests (or responses) have been received (or sent out) and possibly identifying correlations between them. Such mechanisms have been employed considerably in the detection of slow DDoS attacks, and attacks that rely on an underlying UDP connection.

2. Analyzing Request Stream Dynamics : Request dynamics provide a statistical analysis of a data stream, rather than finding meaning in the stream of requests. This encompasses the use of features like number and type of requests, request rate, source IP distribution etc. These mechanisms find extensive use in detecting HTTP flooding attacks.

3. Analyzing Request Stream Semantics : These detection mechanisms try to recognize and model features which represent how a user accesses the web application. A normal user does not consciously know about or manipulate his request rate or his session rate. These users are only concerned about the different resources or web pages that they need to access and the order in which they need to access them. This approach, therefore, analyzes the different pages the users have requested and the sequence in which users request web pages. In other words, these detection mechanisms attempt to find an underlying meaning behind the incoming request stream. Detection mechanisms that focus on semantics are used extensively in detecting HTTP flooding attacks, as well as HTTP asymmetric attacks.

2.2.3 Defending against HTTP Protocol Vulnerabilities

Slow DDoS attacks are the major class of attacks that exploit the HTTP protocol. Cambiaso et al. (2012) created a detailed taxonomy of slow DDoS attacks. Their classification however, clubs asymmetric attacks and attacks like HashDoS attacks into the category of slow DDoS. The rationale behind that classification was that the request rate in these attacks are much less than that in normal flooding.

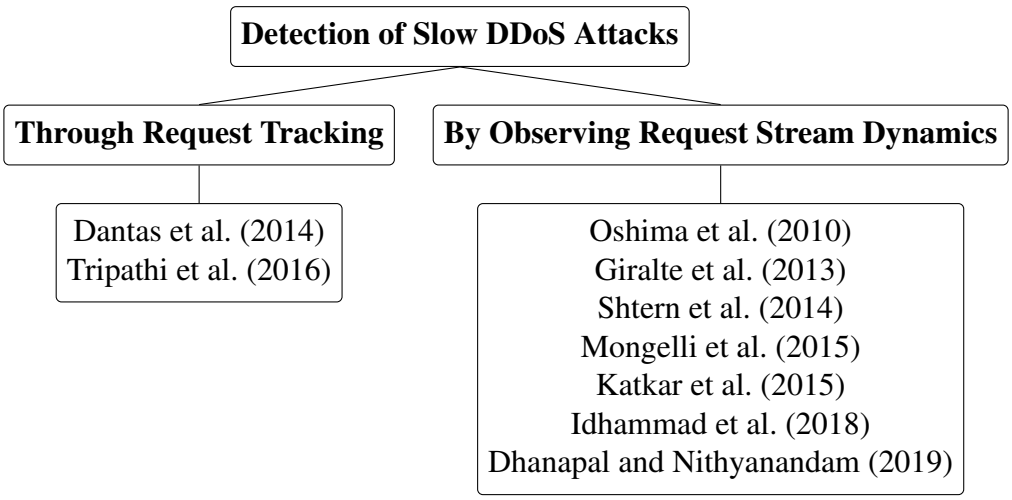


Figure 2.2: Detection Mechanisms for Slow DDoS Attacks

2.2.3.1 Preventing Slow DDoS Attacks

There are comparatively fewer research works done on detecting and defending against slow DDoS attacks. However, these attacks can be mitigated by following some preventive mechanisms such as lowering the timeout value of the server, installing suitable Apache security modules, setting up proper IPTables or IDS rules (Moustis and Kotzanikolaou 2013). Park et al. (2014) conducted simulations that validate the effectiveness of slow DDoS attacks. They observed that the timeout feature in the web application is effective in reducing the magnitude of the attack but cannot stop the attack from happening.

2.2.3.2 Detecting Slow DDoS Attacks

Slow DDoS attacks are usually detected by employing a request tracking mechanism to keep track of the incomplete requests in the system at any point in time. Some works have also used an analysis of request rate (request dynamics) to identify malicious attackers.

Detection Mechanisms which employ Request Tracking Tripathi et al. (2016) proposed that each connection be associated with a vector denoting the percentage of complete or incomplete GET and POST requests. At any instant of time, if the percentages go beyond a learned threshold, the connection is detected as suspicious. They proposed

the use of Hellinger distance to perform the distance calculation. Dantas et al. (2014) proposed maintaining a record of the number of bytes received for each request in each connection on the server. In this case, if the server runs out of connections, it can randomly choose to either drop the incoming connection or to drop an existing connection taking into consideration the number of received and sent bytes. Their approach defends against slow GET, POST and PRAGMA attacks.

Detection Mechanisms which Analyze the Request Rate Shtern et al. (2014) proposed a defense mechanism against slow DDoS attacks based on Software Defined Infrastructure (SDI). SDI is a setup where the network infrastructure is virtualized and the connections and routing tables can be modified on the fly using software controls. This provides a great deal of flexibility. Their approach was to direct suspicious connections to a ”shark tank” where they would be analyzed further. However, their approach works only if the infrastructure is software defined and cannot be used in other cases. The work of Giralte et al. (2013) is one of the few works that seem to be able to detect both HTTP attacks (symmetric and asymmetric) along with Slowloris attacks. Mongelli et al. (2015) performed Fourier transform analysis on the incoming packet stream to identify slow DDoS attacks. Katkar et al. (2015) was able to identify slow read and post attacks using a Naive Bayes classifier. Oshima et al. (2010) and Idhammad et al. (2018) proposed a method of using request entropy to identify slow DDoS attacks. Dhanapal and Nithyanandam (2019) proposed a method for detecting slow DDoS attacks in a cloud environment by monitoring different features of the requests including the request rate and window size of incoming client requests.

Multiple verb HTTP attacks are not featured prominently in literature because the attack is not well documented. However, it seems plausible that deep packet inspection can evade these attacks.

Slow DDoS attacks are stealthy DDoS attacks that can take down a server with minimal resources. However, defending against these attacks poses a familiar double edged sword. Reducing timeout values and limiting the number of connections continue being the basic line of defense against slow DDoS attacks, but these measures can

lead to legitimate users with low bandwidths being forced to relinquishing their connections prematurely. Learning general user behavior continues to be the best defensive option but even with this option, false positives continue to come up. A summary of the detection mechanisms employed against slow DDoS attacks is featured in Figure 2.2.

2.2.4 Defending against HTTP Flooding Attacks

HTTP flooding attacks are one of the most common AL-DDoS attacks and are usually executed by sending a large number of HTTP requests to a web server. Even though the requests used to launch the attack are legitimate, there is considerable difference between the way in which requests flow within a legitimate connection, and an attack stream. This forms the cornerstone of detecting HTTP flooding attacks.

2.2.4.1 Detection Mechanisms based on Request Dynamics

Since a spike in request rate is one of the biggest tell-tale signs of a DDoS attack, one line of research focuses on predicting the expected request rate at each instant and scrutinizing the incoming request stream using this knowledge. However, request rate alone is often not a good indicator of an attack, because sophisticated attacks can maintain the illusion of a normal request rate while launching attacks. In such cases, statistics like request entropy can be used.

Traffic Estimation: Wen et al. (2010) uses traffic estimation as the basis for detecting attacks. Their system estimates the expected traffic from historical data using a Kalman filter. If a significant deviation is witnessed from the expected traffic value, this indicates either an attack or a flash crowd. The source IP distribution provides additional evidence to determine if the flood is actually an attack or not. Ni et al. (2013) used an Adaptive Auto Regressive (AAR) Model to model network traffic. The estimated value is smoothed with a Kalman filter. They introduced a feature called HRPI (HTTP Requests Per IP) as the classification variable. The classification is performed using a Support Vector Machine (SVM).

Request Statistics: There are a large number of statistical features that have been used in detecting HTTP floods apart from just the request rate. Some of these features

include the request timestamp, IP address, header fields, user agents, number of 200 OK responses, number of error responses and so on. Most research works use a combination of these features for detecting attacks. Yadav and Selvakumar (2015) used Principal Component Analysis (PCA) and logistic regression on statistical features to identify attacks. In another work, (Yadav and Subramanian 2016), the same authors applied deep learning on these same statistical features. The authors employed a stacked auto-encoder to extract latent features from the basic statistical features and used logistic regression on these features. As similar work using logistic regression was undertaken by Aljuhani et al. (2019) for detecting HTTP flooding attacks by monitoring request arrival times. Johnson et al. (Johnson Singh et al. 2016; Singh and De 2017) used a multilayer perceptron (MLP) with the weights trained by a genetic algorithm. In the work of Johnson Singh et al. (2016) the training features were HTTP GET count, entropy and variance, while in Singh and De (2017) the features used were number of HTTP count, number of the IP addresses, constant mapping function and fixed frame length. Chwalinski et. al. (Chwalinski et al. 2013a,b) used the number of requests per resource for each user as the deciding feature and performed K-means clustering to group users into different clusters. They applied likelihood analysis and Bayes factors to determine if an incoming connection can be attributed to an existing cluster or not. A connection that cannot be attributed to any cluster is deemed as malicious. Oo et al. (2016) extracted features like number of packets, number of bytes, average packet size, packet rate, byte rate, time-interval variance and packet-size variance from connections. If all of these features fall within limits then the connection is most likely benign and is accepted. If all of these values fall outside acceptable limits, the connection is most probably malicious and is blocked. In other cases, where some of these features fall within the specified range and others do not, they employed an HsMM (Hidden semi Markov Model) to efficiently model and classify the connection. Lee et al. (2011) used Principal Component Analysis (PCA) to reduce the feature dimension and then performed clustering.

Entropy is a much used measure to determine whether a stream of requests is malicious or not. A normal user sends requests which are varying in size, speed and intent.

A stream of attack requests on the other hand will be more uniform and have similar requests repeated at regular intervals. As a result, an attack stream will have lower entropy than normal user requests. Devi and Yogesh (2012a) used this information along with trust values of users to detect attacks. Zhou et al. (2014) also used model entropy but their work was meant for identifying attack streams in backbone web traffic. Zhao et al. (2018) used two measures - Entropy of URL per IP (EUPI) and Entropy of IP per URL (EIPU) for identifying flooding attacks. During a random flooding attack or a fixed URL flooding attack, EUPI would increase, thus indicating an attack. EIPU helped in distinguishing attacks and flash crowds.

2.2.4.2 Detection Mechanisms based on Request Semantics

Request semantics can be further refined to consist of two classes. Mechanisms that rely on the composition of requests in a request stream, without any consideration to the sequence are much easier to use because of the reduction in data that needs to be analyzed. Mechanisms that rely on request sequence on the other hand tend to be more complex but on the whole tend to be more accurate because they can model normal human behaviour much more accurately.

Request Composition: Devi and Yogesh (2012b) constructed an access matrix using features like HTTP request rate, HTTP session rate, server documents that are accessed and duration of user's access. Singular Value Decomposition (SVD) is applied followed by Independent Component Analysis (ICA) for dimensionality reduction. The incoming connection is assigned a suspicion score based on how much deviation it exhibits from the output of the ICA module. Instead of blocking the connection if it does not match the learned model, this approach schedules the request accordingly. A request which matches with the learned model will have a low suspicion score and consequently will be scheduled for execution faster. A request with a high suspicion score will be scheduled much later. Beitollahi and Deconinck (2014) follows a similar approach. The system constructs the CDF (Cumulative Distribution Function) for each of the observed features in the normal user sessions. For an incoming connection, it assigns a suspicion score for each feature based on how likely it is to have the observed

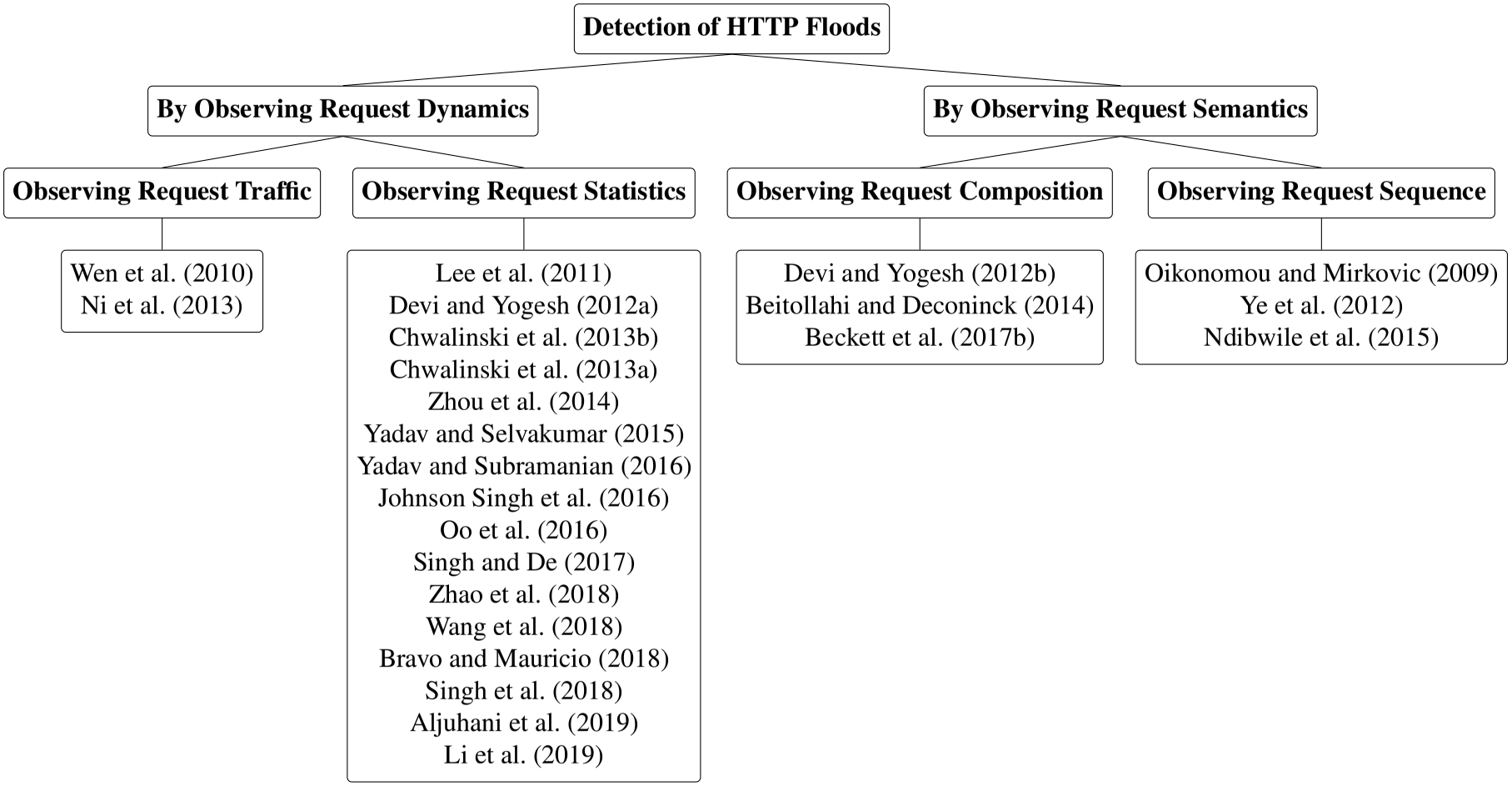


Figure 2.3: Detection Mechanisms for HTTP Flooding Attacks

value according to the CDF. The final suspicion score is the sum of the individual suspicion scores for all the features. Beckett et al. (2017a) presented a novel approach to detect DDoS attacks targetting database systems by observing features like the number of databases opened or closed, total and average query time etc. They used a decision tree classifier to identify malicious users based on these features.

Request Sequence: Ye et al. (2012) uses average transition probability and page popularity as features for clustering. They used Euclidean distance and Ward's linkage to match an incoming connection into a cluster. Ndibwile et al. (2015) proposed the use of three servers - a bait server, a decoy server and a real server - for the detection of attacks. They proposed that all traffic be directed to the bait server initially. From there, proven benign traffic is directed to the real server while suspicious traffic is routed to the decoy server. At the decoy server, traffic is authenticated using decision trees trained using known attack generation tools. Oikonomou and Mirkovic (2009) used dynamics, semantics and decoys to weed out attackers. Dynamics refers to features like number of sessions, average pause between sessions, average number of requests per session, and average request inter arrival rate per session. Decision trees were used for classifying the incoming connections. Semantics is modeled by creating a probability graph of the website. The probability of a path is defined as the average of the probabilities of all the edges in the path. Finally, they used decoys which are invisible links or images embedded in the web page. These links are invisible to normal users and hence do not show up in the traces for normal users. However, the links are still parsed by bots and can be used to identify them.

Figure 2.3 depicts a summary of the detection mechanisms that can be used for HTTP flooding attacks.

HTTP flooding attacks are the most common application layer DDoS attacks. It is often possible to identify a flooding attack by observing the characteristics of the request stream. This is the reason most of the application layer firewalls can provide a great level of defense against HTTP floods. However, systems are far from immune to flooding attacks. Even at the application layer, attack volume is on the rise for flooding

attacks. Though still many orders of magnitude less than the network flooding volume, the high volume HTTP floods can clog the firewalls, thus creating a new bottleneck.

2.2.5 Defending Against Asymmetric HTTP Attacks

Compared to HTTP flooding attacks, asymmetric attacks are much harder to detect. This is because of the fact that HTTP floods are marked by a sharp change in many features which can lead to its identification. Asymmetric attacks on the other hand can be executed without raising too many red flags and hence evades detection to a large extent. However, there are certain warning signs that can signal an asymmetric attack in process. Every web application has a set of normal users who browse the web application in a certain way. This means that regular users often follow certain paths in the website more often than others. In other words, certain request sequences are more likely than others. Apart from that, normal users take time to browse the web application. Between every pair of requests issued, there is an associated time gap, often called ”think time” because this is when the user processes the web response and decides what to do next. These two features are often not followed in the case of a request sequence submitted by a bot. This forms the basis of identifying an asymmetric attack.

In other words, request dynamics are unlikely to be of any use in detecting asymmetric attacks. All existing detection mechanisms use request semantics in some form. A summary of the detection mechanisms for asymmetric AL-DDoS attacks is given in Figure 2.4.

2.2.5.1 Detection Mechanisms based on Request Composition

Ranjan et al. (2009) was one of the earliest works to take into account the workload profile of a user into account for detecting DDoS attacks. The idea was that normal users are not likely to send high workload requests continuously to the server. Their request profiles would contain interleaved low, medium and high workload requests, while the request sequence of an attacker would have a stream of high and medium workload requests. They analyzed session and request inter arrival distribution, along with client workload profile to assign suspicion scores to users. Based on the deviation of each

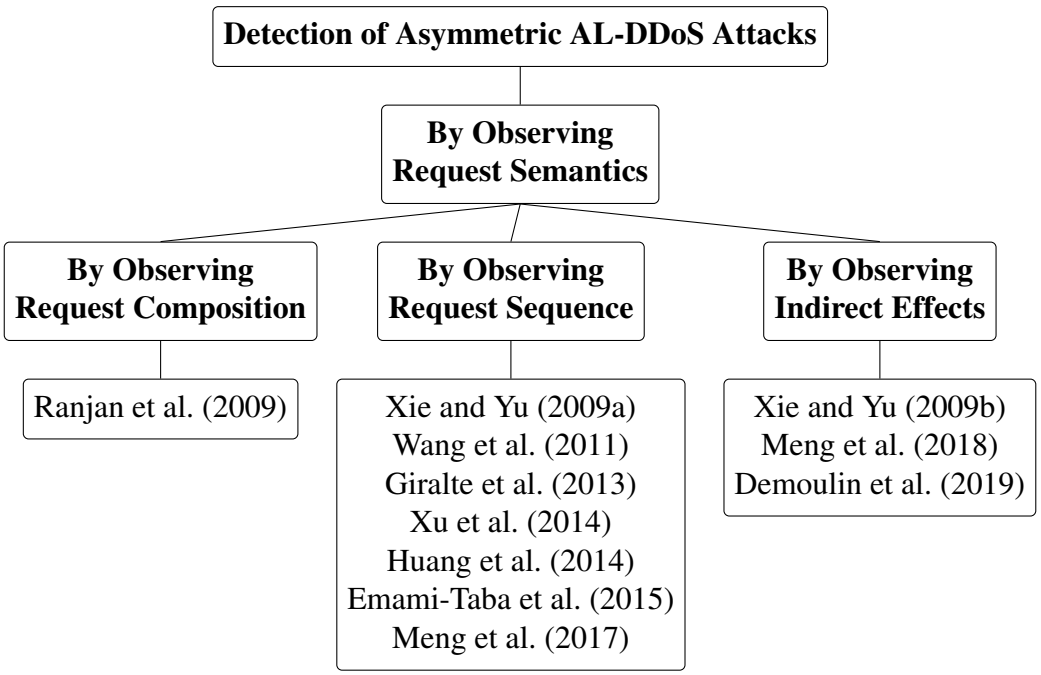


Figure 2.4: Detection Mechanisms for Asymmetric AL-DDoS Attacks

feature from the legitimate user profile, they assigned suspicion scores to individual users. The connections were scheduled according to the suspicion scores.

2.2.5.2 Detection Mechanisms based on Request Sequence

Most of the existing works use attack-free user traces to learn how a user accesses the website. Based on this learned model, they calculate the normality of the observed request sequence, which denotes the probability that the incoming user sequence was generated by a legitimate user. Xu et al. (2014) modeled the user behavior in a web application as a probabilistic graph. For each incoming connection, they observed the stream of requests and predict the future request sequence. The similarity between the predicted and observed stream of future requests is used to identify malicious users. Giralte et al. (2013) presented a three stage detection mechanism which utilizes statistical features, request sequence and request sequence similarity for detecting attacks.

The defense and offense involved in defending against DDoS attacks is strikingly similar to a game between the attacker and the web administrator. Emami-Taba et al. (2015) used Game Theory to develop a set of payoff tables to model the attack scenario and used the min-max algorithm to identify attackers.

Xie and Yu (2009a) were the first to use an HsMM for the purpose of detecting denial of service attacks at the application layer. Their work constructed the HsMM from system traces and approximated the think time associated with a page as the number of inline requests the page makes. This can be seen as an approximation of page loading time. They constructed a normal distribution of the likelihoods of the observed sequences which is called Original Likelihood Distribution (OLD). The amount of deviation from the OLD is taken to be the abnormality of an observed request sequence made by a user. A similar approach was also taken by Meng et al. (2017). Huang et al. (2014) also modeled page popularity of a website using an HsMM. They clustered the available data sets into clusters before using the features to construct an HsMM to reduce the dimensions of the data.

2.2.5.3 Detection Mechanisms by Observing Indirect Effects

There is another approach to detect an application layer attack which relies not on observing the request stream directly, but rather observing the effect of the request stream. Every website has a set of ”hot” pages, i. e. pages which are visited more often than others. Generalizing this observation, every page in a website has a probability with which a user accesses it, which denotes how popular the page is. An attack, which does not follow normal user patterns, tends to disrupt this page popularity in haphazard ways. This provides an indirect indication of attack.

Wang et al. (2011) assumes each user accessing the website has an a priori click ratio which denoted the popularity of the web pages. The idea is that when a web application is under attack, the click ratio deviates from the a priori one, and this is used as the means of identification. This work uses large deviation theory to identify how probable a deviation from the a priori click ratio is. They also modeled the click ratio by means of an HsMM and employed Large Deviation theory to measure the probability of deviation. Another work by Xie and Yu (2009b) constructed an HsMM modeling the document popularity of a web site. However, they observed that algorithms for building and operating HsMM become considerably complex when using high dimensional data. Hence to reduce the dimensionality of the input data they used traditional dimensionality reduction algorithms of PCA and ICA (Independent Component Analysis).

Table 2.1: Summary of AL-DDoS Detection Mechanisms

Attack	Research Work	Detection Mechanism
Asymmetric AL-DDoS Attack	Meng et al. (2017)	Monitoring Request Sequence
	Emami-Taba et al. (2015)	
	Huang et al. (2014)	
	Xu et al. (2014)	
	Giralte et al. (2013)	
	Wang et al. (2011)	
	Xie and Yu (2009a)	
	Demoulin et al. (2019)	Observing Indirect Effects
	Meng et al. (2018)	
	Xie and Yu (2009b)	
	Ranjan et al. (2009)	Monitoring Request Composition
HTTP Flooding Attacks	Ndibwile et al. (2015)	Monitoring Request Sequence
	Ye et al. (2012)	
	Oikonomou and Mirkovic (2009)	
	Beckett et al. (2017b)	Monitoring Request Composition
	Beitollahi and Deconinck (2014)	
	Devi and Yogesh (2012b)	
	Li et al. (2019)	Monitoring Request Statistics
	Singh et al. (2018)	
	Bravo and Mauricio (2018)	
	Wang et al. (2018)	
	Zhao et al. (2018)	
	Singh and De (2017)	
	Oo et al. (2016)	
	Johnson Singh et al. (2016)	
	Yadav and Subramanian (2016)	
	Yadav and Selvakumar (2015)	
	Zhou et al. (2014)	
	Chwalinski et al. (2013a)	
	Chwalinski et al. (2013b)	
	Devi and Yogesh (2012a)	
	Lee et al. (2011)	
	Ni et al. (2013)	Monitoring Request Rate
	Wen et al. (2010)	
Slow DDoS Attacks	Dhanapal and Nithyanandam (2019)	
	Idhammad et al. (2018)	
	Katkar et al. (2015)	
	Mongelli et al. (2015)	
	Shtern et al. (2014)	
	Giralte et al. (2013)	
	Oshima et al. (2010)	
	Tripathi et al. (2016)	Request Tracking
	Dantas et al. (2014)	

Meng et al. (2018) and Demoulin et al. (2019) both observed the increase in resource usage due to incoming requests as an indication of asymmetric attacks. However, their work requires kernel modification in order to identify the spikes in workload, and is hence more difficult to implement.

HsMM proves to be extremely efficient in detecting asymmetric as well as flooding attacks, but they do have some drawbacks. The detection principle involves calculating the probability of the observed sequence at each instant of time. This algorithm however, is complex and thus the overall system load will be more. In some cases, simple statistical calculations prove more efficient than using an HsMM.

Asymmetric HTTP DDoS attacks are extremely severe because of their similarity to normal human behaviour. Defenses relying on request rate and request statistics will fail to detect these attacks, and by extension they make existing web application firewalls obsolete. Identifying these attacks is a complex procedure which involves learning normal user behaviour and weeding out connections that show unnatural behaviour. However, most of the techniques used for detecting these attacks rely on an HsMM. While being highly effective, an HsMM has a large computation cost associated with it, which makes it a questionable choice for run time detection. It may also happen that the defense system becomes a bottleneck that attackers can exploit.

A summary of the detection mechanisms that are commonly employed for detecting the different classes of AL-DDoS attacks is summarized in Table 2.1. It can be seen that asymmetric AL-DDoS attacks are the most potent form of AL-DDoS attacks and require sophisticated modelling of user behavioural dynamics to enable their detection.

2.3 HTTP/2 AND ASSOCIATED SECURITY CONCERNS

HTTP was designed for simple, static web applications in the initial days of the internet and today's complex, dynamic web applications faced performance issues when using HTTP/1.1. A major drawback of HTTP/1.1 was Head-of-Line (HoL) Blocking, which means that an HTTP request can only be processed once the preceding request belonging to the same connection has been processed. This can lead to undue waiting time in the case of large web applications with multiple images embedded in a single web page.

HTTP/2, on the other hand, was designed with performance in mind, and has several features meant to reduce page loading times in web applications. A brief discussion of the new features in the HTTP/2 protocol is presented in the following paragraphs (Belshe et al. 2015).

- Binary Encoding : Unlike HTTP/1.1 which is a purely textual protocol, HTTP/2 is a completely binary protocol. Binary protocols are compact, less prone to errors and more efficient to parse (Group 1999).

- Multiplexing on a Single TCP Connection : HTTP/2 allows multiple messages to be sent through a single TCP connection and processed simultaneously at the server. This allows content-heavy web pages to be loaded faster by sending multiple inline requests in a single connection.

- Streams and Frames : A bi-directional flow of bytes in HTTP/2 is called a stream, and is identified by a stream ID. HTTP messages (requests or responses) flow along these streams. Each message is further broken down into frames. There are ten different types of frames defined by the HTTP/2 protocol, each of which has a different purpose.

- Stream Prioritization : HTTP/2 allows users to specify a particular order or request processing by specifying priorities for streams. A user can also explicitly specify dependencies between streams to ensure the correct order of processing.

- Header Compression : HTTP/2 specifies that headers must be compressed using the HPACK compression format to reduce overhead. The format uses static Huffman codes to reduce individual transfer sizes and also requires that the client and server both maintain a list of previously seen headers. This way, further communication from the same client can simply reference the previous headers and only explicitly specify the new information. Since majority of the header information remains constant throughout a conversation between a server and client, this reduces transfer overheads to a large extent.

- Server Push : The most interesting aspect of HTTP/2 is that it allows the server to push resources to a client without an explicit request. This is done using the PUSH_PROMISE frame. This is especially useful when the client is requesting for a large web page with multiple images or scripts embedded in it.

2.3.1 HTTP/2 Security

HTTP/2 still retains all the semantics of its predecessor, and hence most of the attacks that could be launched using the HTTP/1.1 protocol can also be launched using the HTTP/2 protocol. Research into HTTP/2 security has proceeded in two main directions. On one hand, research has focused on identifying whether HTTP/2 remains vulnerable to the attacks that affected HTTP/1.1 or not. The other line of research focuses on examining the newly standardized features of HTTP/2 for vulnerabilities. A summary of the research works addressing different facets of HTTP/2 security are given in Table 2.2.

2.3.1.1 Legacy Attacks on HTTP/2

DDoS attacks are one of the most popular attacks against web applications. It is reasonable that once HTTP/2 was released, researchers immediately began to study how the new version of the protocol behaved during a DDoS attack.

Slow DDoS Attacks: An investigation by Imperva (2016) and Tripathi and Hubballi (2018) identified a slow DDoS vulnerability lingering from the previous version of the protocol. A slow read DDoS attack could be executed on an HTTP/2 server by using the WINDOW_UPDATE frame with the actual window update value set to zero. A slow read attack using HTTP/2 was possible on Jetty, Apache, IIS and Nginx servers. Tripathi and Hubballi (2018) uncovered five different types of slow DDoS attacks that were possible by manipulating certain settings in the control frames. They also proposed a statistical mechanism to detect these attacks.

HTTP Flooding Attacks: Adi et al. (2015) observed that sending a stream of WINDOW_UPDATE frames with different stream IDs drives CPU utilization over 50%. Beckett and Sezer (2017a) found out that an HTTP/2 server was more vulnerable to an

Table 2.2: Comparison of Existing Research Works on HTTP/2 Security

Research Work	**Feature Exploited**	**Attacks Discussed**	**Limitations**
Imperva (2016), Suresh et al. (2018) and Hu et al. (2018)	HTTP/2 multiplexing, stream dependency and header compression	Reusing of streams in HTTP/2 leads to server crash; Presence of a cycle in stream dependencies leads to increased CPU usage; Abnormally large header values can lead to memory exhaustion when decompressed	Attacks worked only on specific servers; These vulnerabilities have been patched since
Imperva (2016)	HTTP connection maintenance and timeout	Slow Read DDoS attack	Setting proper timeout values for servers can mitigate the effect to some extent
Tripathi and Hubballi (2018)	HTTP connection maintenance and timeout	Five different types of Slow DDoS attacks using complete or incomplete headers	Setting proper timeout values for servers can mitigate the effect to some extent
Adi et al. (2015)	HTTP/2 CPU usage	HTTP/2 control frame flooding	Requires a large number of frames to launch the attack
Beckett and Sezer (2017a)	Server Bandwidth	HTTP/2 data frame flooding	Did not examine if the improvements in the server allow it to handle the load
Beckett and Sezer (2017b)	Server Bandwidth	Amplification DDoS attack on an HTTP/2 - HTTP/1.1 gateway	Did not present an evaluation of the impact of these attacks on the servers
Patni et al. (2017)	SSL encryption	Man-in-the-Middle attack on HTTP/2 traffic through a combination of DNS cache poisoning, Phishing and TLS certificate forging	Requires extensive planning and preparation to execute the attack

application layer flood primarily because the smaller request sizes in HTTP/2 means that an attacker could now generate more number of requests with far less bandwidth, and hence overpower an HTTP/2 server more effectively.

Amplification Attacks: Beckett and Sezer (2017b) further explored the issue of an amplification attack when there is an HTTP/2 gateway proxy which connects to a HTTP/1.1 back-end web server. The idea behind this attack is that in the HTTP/2 protocol, headers are compressed and hence bandwidth and packet size are reduced. However when these requests cross over a gateway that connects to an HTTP/1.1 server, the headers must be decompressed and packets must be converted to HTTP/1.1 format. This would increase the bandwidth considerably and cause the links leading to the server to be flooded.

Man-in-the-Middle Attacks: Patni et al. (2017) devised and demonstrated a Man-in-the-Middle (MITM) attack against HTTP/2 through a combination of DNS cache poisoning, Phishing and TLS certificate forging. They also suggest some methods to mitigate such an attack.

2.3.1.2 Attacks Exploiting New Features

Research into the newly introduced features in HTTP/2 was carried out by Imperva (2016), and later by Suresh et al. (2018) and Hu et al. (2018). Three major vulnerabilities were identified in several implementations.

Stream Reuse Vulnerability: According to the HTTP/2 specifications, a stream is meant to transport an HTTP request and its response. Once closed, the stream is not supposed to be reused or reopened. The reopening and reuse of a stream leads to a Blue Screen of Death (BSOD) on the IIS server. Such an error points to a process encroaching on the memory allocated for another process, which in some cases could lead to execution of arbitrary code.

Dependency Cycle Vulnerability: HTTP/2 allows users to specify inter-dependencies between streams to designate the processing order. If this dependency chain becomes arbitrarily long or contains a cycle, it can lead to high CPU usage and possibly a DoS

attack. This vulnerability was found in Apache and nghttp2 servers.

HPACK Bomb: The HPACK Bomb vulnerability is concerned with header compression and stems from the fact that there is no restriction on the header size in HTTP/2 other than the fact that it should be less than the size of the dynamic header table at the server side. An attacker can hence use one request to fill up the entire table by using a header of the same size as the table. Subsequent requests can refer to the same header multiple times which causes the server to allocate memory to decompress the headers. This can quickly lead to memory exhaustion. This vulnerability was discovered in nghttp2 servers. The timely identification of these vulnerabilities by Imperva allowed all the major servers to fix these vulnerabilities before they could be exploited. A common feature in most of these attacks have been that they have been executed by using the features in ways that they were not intended to be used. We propose that multiplexing and server push can also be misused to launch sophisticated DDoS attacks against an HTTP/2 server, and can bring down servers without the need for a large botnet.

2.4 RESEARCH DIRECTIONS AND CHALLENGES

Majority of the research in AL-DDoS detection has focused on HTTP flooding attacks using the protocol, and there have been comparatively few research works on asymmetric AL-DDoS attacks. Given the potency of asymmetric AL-DDoS and the speed with which they can damage web servers, there is a need for further research to improve upon the existing detection mechanisms.

- Most of the existing detection mechanisms for asymmetric AL-DDoS attacks use complex modelling techniques and indirect representations of user behaviour, which leads to larger false positive rates and longer detection times. The use of more efficient models can help in reducing the detection time by a large extent. In addition, the use of better features which directly correlate with observed user behaviour could help in reducing the false positive rate considerably.

- The problem of concept drift, which is the gradual change in user behaviour over a period of time, has not been addressed in most of the detection mechanisms. A

change in legitimate user behaviour causes the learned model to become obsolete over time, leading to a higher false positive rate. There is a need for incrementally learning detection mechanisms which can effectively sense concept drift and update the model accordingly.

- Asymmetric AL-DDoS attacks can damage web servers much faster than normal attacks, and hence there is a need to detect these attacks as early as possible. The problem of early detection has been addressed in the context of netwrok layer DDoS attacks, but hasn't been explored in the context of AL-DDoS attacks.

- Research on AL-DDoS attacks is primarily focused on the HTTP/1.1 protocol. There have been relatively few research works that examine the possibility of asymmetric AL-DDoS attacks using the HTTP/2 protocol. With HTTP/2 set to replace HTTP/1.1 in the coming years, there is a need for research into how the HTTP/2 protocol responds to AL-DDoS attacks. In addition, there is a need to study whether the newly introduced features in HTTP/2 can be misused to launch AL-DDoS attacks.

2.5 SUMMARY

In this chapter, a comprehensive review of existing literature related to AL-DDoS attacks was presented. AL-DDoS attacks can be executed in a variety of ways, and a proper understanding of the ways in which they are executed proves to be useful in detecting these attacks. A taxonomy of AL-DDoS attacks using the HTTP protocol was presented in this chapter, along with a review of the existing techniques used for detecting these attacks. The literature review reveals that asymmetric AL-DDoS attacks are perched on top of the hierarchy of AL-DDoS attacks, and require sophisticated modelling techniques and detection mechanisms for their efficient identification. Existing mechanisms for the detection of asymmetric AL-DDoS attacks suffer from certain drawbacks such as high computational complexity and longer detection times and higher number of false positives. The introduction of HTTP/2 also adds further complexity to the detection of these attacks, and there is a need for further research to develop efficient detection mechanisms for these attacks.

CHAPTER 3

PROBLEM DESCRIPTION

The primary aim of this research work is concerned with the efficient detection of asymmetric AL-DDoS attacks on web applications. In the past decade, there have been some significant research works that address the problem of asymmetric AL-DDoS detection (Ranjan et al. 2009; Xie and Yu 2009a,b; Xu et al. 2014). All of these research works follow an anomaly detection approach by first modelling legitimate user behaviour, and then observing for deviations from this learned model in order to identify attacks. Thus, the problem of asymmetric AL-DDoS attack detection can be described in two phases and can be formally stated as follows:

- Learning Phase: Given a set of application specific server logs and user access logs I_+ as input, a set of features $F = \{f_1, f_2...f_k\}$ representing legitimate user behaviour must be extracted, and formulated into a model M.
- Detection Phase: Given a sequence of requests $r_1, r_2...r_n$, a function D has to be developed such that $D(r_1, r_2...r_n : M)$ gives an indication of the degree to which the sequence of requests is anomalous.

An efficient detection mechanism also possesses the following constraints associated with the selection of the parameters F, M and D. The set of features F must be capable of representing the behavioural dynamics of legitimate users, and the choice of F will affect the performance of the detection mechanism. The model M must be lightweight and capable of incremental update to avoid periodic retraining. The detection mecha-

nism D operates alongside the web server it is designed to protect, and uses part of its resources. Since the resources are constrained and are essential for proper execution of the web server, the detection mechanism D must be computationally inexpensive, preferably with a linear computational complexity.

The introduction of the HTTP/2 protocol does not alter the fundamental premise of attack detection, but modifies the behavioural dynamics of users slightly. The introduction of multiplexing in HTTP/2 means that multiple requests can be sent simultaneously through a single TCP connection. A request under the HTTP/2 protocol is no longer a solitary entity, but a set of related entities which can be represented as a request set $(r_1, r_2...r_m)$. The feature set F, model M and detection mechanism D must therefore be capable of representing this information, and using this information for attack detection. In particular, the request set F must include features which represent the relationship between the requests within a request set. In turn, the model M must also be updated to include this association between requests in a request set and the detection mechanism D must use this information in the attack detection phase. Under the HTTP/2 protocol, the detection phase can be formally stated as: Given a sequence of requests $(r_{11}, r_{12}...r_{1m}), ...(r_{n1}, r_{n2}...r_{nm})$, a function D has to be developed such that $D((r_{11}, r_{12}...r_{1m}), ...(r_{n1}, r_{n2}...r_{nm}) : M)$ gives an indication of the degree to which the sequence of requests is anomalous.

In addition, the decision length of the detection mechanism N is defined as the smallest value of n for which a request sequence $r_1, r_2...r_n$ can be identified as malicious by D. An efficient detection mechanism must have a sufficiently small value of decision length to enable early detection of attacks.

To summarize, the primary goal of this research work is to propose an efficient detection mechanism for asymmetric AL-DDoS attacks. The work can be subdivided into the following objectives.

1. Constructing an incrementally updating model in order to capture the behavioural dynamics of legitimate users along with an approach for identifying malicious clients

2. Exploring the possibility of asymmetric AL-DDoS attacks against HTTP/2 servers, and to enhance the model to accommodate HTTP/2 specific features in order to detect these attacks

3. Designing an Early Detection Module (EDM) for AL-DDoS attacks using dynamic signatures based on HTTP request patterns

CHAPTER 4

ASYMMETRIC AL-DDOS ATTACKS ON HTTP/1.1 SERVERS

4.1 INTRODUCTION

Asymmetric AL-DDoS attacks on web servers running the HTTP/1.1 protocol were first described in the works of Ranjan et al. (2009). They demonstrated that the use of computationally intensive or high workload requests to launch DDoS attacks can take down target servers using fewer requests than regular HTTP flooding attacks. Since then, there have been a number of research works that aim at detecting asymmetric AL-DDoS attacks on web applications. Majority of these research works used an anomaly detection approach by building a model of legitimate user behaviour, and then observing for deviations from the learned model in order to detect attacks. However, most of these detection mechanisms use indirect representations of user behaviour and use complex models to represent user behaviour. This leads to a higher false positive rate and longer detection time. In addition, these mechanisms are often unable to adapt to changing user behaviour and require periodic retraining, which leads to additional overhead. Thus, there is a need for a fast and lightweight detection mechanism that has a low false positive rate and is capable of incremental learning.

In this chapter, we describe our approach to build an efficient detection mechanism for asymmetric AL-DDoS attacks. The rest of the chapter is organized as follows: Section 4.2 describes the essential requirements of an asymmetric AL-DDoS detection mechanism. Section 4.3 introduces the concept of request workload, which is funda-

mental to asymmetric AL-DDoS attacks. Section 4.4 presents the methodology commonly used for generating asymmetric AL-DDoS attacks against HTTP/1.1 servers, and Section 4.5 describes our proposed approach for the detection of these attacks.

4.2 REQUIREMENTS OF A DETECTION MECHANISM

The detection of asymmetric attacks presents a unique and difficult challenge. Any detection mechanism must be able to detect attacks as soon as possible, i.e. they must have a very low detection latency. In addition, asymmetric AL-DDoS detection mechanisms must possess certain unique characteristics that are not necessarily part of other DDoS detection mechanisms. In particular, we present four desirable features for any asymmetric AL-DDoS detection mechanism.

- Lightweight: Any detection mechanism is supposed to operate alongside a web server, and hence has to be lightweight. Otherwise, the detection mechanism by itself provides a bottleneck that attackers can exploit.

- Judgement: A good detection mechanism must be able to judge whether an incoming request stream is malicious or not with a very low false positive rate. This is particularly important in the case of flash crowds, which are often mistaken for AL-DDoS attacks and blocked.

- Adaptability: A good detection mechanism must be able to adapt to changing user behaviour. This often involves periodic updates to the underlying model in order to accommodate for changing user behaviour, thus avoiding the need for retraining.

- Prioritized Detection: It is crucial that asymmetric AL-DDoS be detected faster compared to flooding attacks, because they are capable of damaging servers with much fewer requests. Thus, a good detection mechanism must be able to distinguish between asymmetric attacks and flooding attacks, and must be able to detect the former much faster.

4.3 WORKLOAD PROFILING

The very idea of asymmetric AL-DDoS attacks revolves around the concept of request workload. In simple terms, the workload associated with a request R is an indication of the amount of resources required by a web server to execute the request. These resources could be CPU or database cycles, memory or socket connections. From a more technical standpoint, the workload associated with a request R can be denoted by a n-tuple $W = (v_1, v_2, ...v_n)$ where v_i denotes the resource usage of resource r_i when executing request R. W is also referred to as the workload dimension of request R. Majority of the asymmetric AL-DDoS attacks target the CPU or database, and hence the workload can be expressed in two dimensions. Requests in the web application are grouped into different workload classes depending on their resource requirements. A simplistic classification is to group resources into two classes - requests that require more resource usage are called high workload requests, and others are called low workload requests. A more detailed classification could include k workload classes instead of just two.

In the case of a single resource (say, CPU usage), the task of workload classification is trivial. However, when the workload dimension is more than one, classifying requests into k workload classes requires a more involved effort. Mishra et al. (2010) considered the problem as a special case of task classification, and used k-means clustering to group the requests into k workload classes.

4.3.1 An Approximation of Request Workload

Asymmetric AL-DDoS attacks are executed by using high workload requests. However, an attacker has no access to the workload dimensions of a request since they have no access to the server usage statistics. In such a case, response time for requests can be taken as an approximation of request workload (Ranjan et al. 2009). Intuitively, a request that has a higher workload will take more time to execute at the server and will have a higher response time. However, the calculation of response time is a sensitive task that can easily be corrupted by network delays and varying system loads. Thus it is necessary to sample the response time values at varying time intervals, under different

system loads before using it for request workload classification.

4.3.2 System and User Workload Profiles

In the context of request workload, we define two terms - system workload profile and user workload profile.

- System Workload Profile: Let us consider a web application designed to handle m different types of requests $R = \{r_1, r_2, ..., r_m\}$. These requests introduce different levels of computation which can be approximated to k different classes of workload, $W = \{w_1, w_2, ..., w_k\}, w_1 < w_2 < ... < w_k, k \leq m$ into which each on the m requests can be mapped to. The System Workload Profile for a web application is essentially a function $f : R \rightarrow W$ so that $f(r) = w$ gives the workload class w that request r belongs to. A sample workload profile generated for a test application Mutillidae is given in Figure 4.1. The X-axis represents the different states, identified by a state ID. There were 141 distinct states in the web application, including images and and CSS files. The y axis represents the response times in milliseconds, which serves as an indicator of request workload.

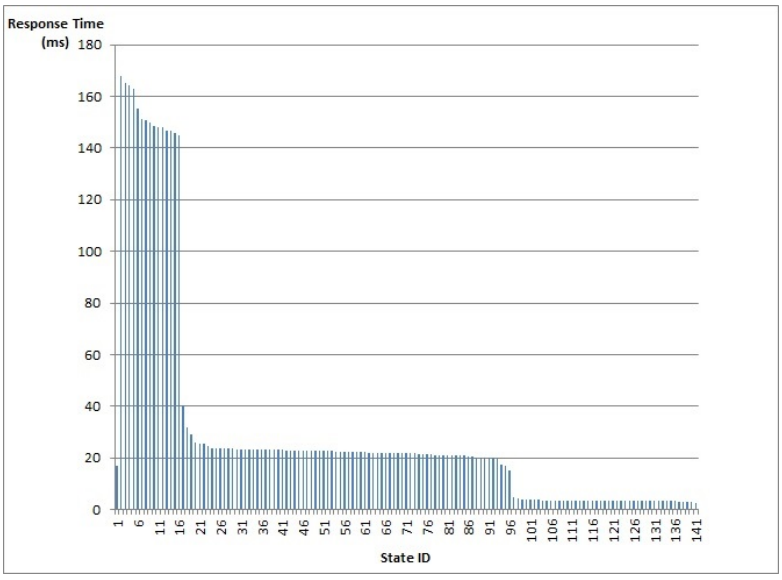


Figure 4.1: Workload Profile for Mutillidae

- User Workload Profile: A user interacts with a web application by issuing a sequence of requests. The workload profile for a user issuing a sequence of n

requests $(r_1, r_2...r_n)$ is an n-tuple $(w_1, w_2, ...w_n)$ where $w_i = f(r_i)$.

4.4 ATTACK GENERATION ON HTTP/1.1 SERVERS

There are a large number of tools available for generating DDoS attacks (Behal and Kumar 2017). Most of these tools are capable of generating a variety of attacks at the network layer, including UDP and ICMP floods. A few attack tools are even capable of generating TCP or HTTP floods at the application layer, but there is a dearth of tools which are capable of generating asymmetric attacks. This is probably due to the additional reconnaissance effort required to launch these attacks, and partly explains why these attacks are not as popular and prevalent as other attacks. Due to this reason, attack scripts had to be generated manually to launch asymmetric AL-DDoS attacks. A summary of the popular attack generation tools and the attacks that can be launched by them are given in Table 4.1.

Table 4.1: Popular HTTP DDoS Attack Generation Tools

Attack Generation Tool	**Attacks Generated**
Low Orbit Ion Cannon (LOIC)	TCP, UDP or HTTP floods
HULK	Random HTTP floods
DDoSim	TCP or HTTP floods
R-U-D-Y (aRe yoU Dead Yet)	Slow POST
Slowloris	Slow GET
Tor's Hammer	Slow POST
Pyloris	Slow DDoS attacks

4.4.1 Methodology

Generating asymmetric attacks against web applications proceeds in four stages - web application scanning, identification of critical states, generating an attack plan and launching the attack. A block diagram describing the attack generation process is given in Figure 4.2.

- Web Application Scanning : The first stage in the attack generation process is scanning or profiling a web application. This is done to create a system workload profile of the web application, and to identify high workload states which can be targetted under an asymmetric attack.

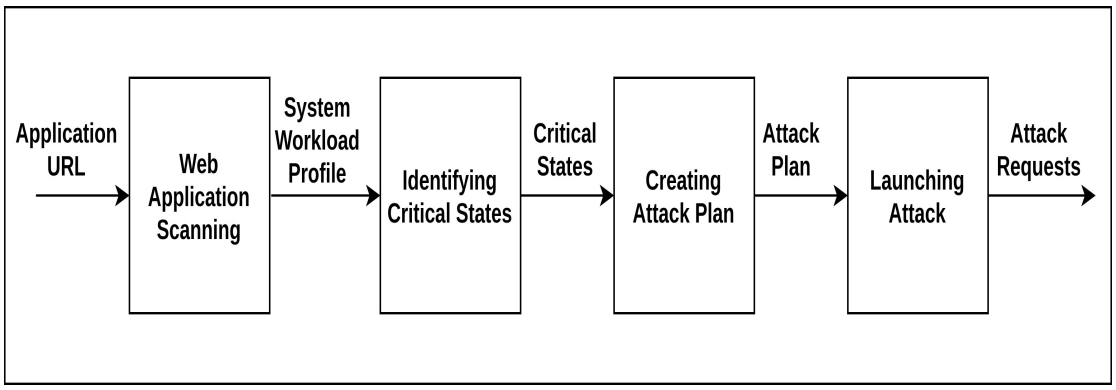


Figure 4.2: Block Diagram for Attack Generation

- Identifying Critical States : From the generated system workload profile for the web application, the requests with the highest resource usage are identified. These requests constitute the Critical States, and are used to generate attacks against the web application.

- Generating an Attack Plan : All DDoS attacks are executed using a predefined script, which sends requests to the target server in a predefined, repeating fashion. Once the system workload profile is generated for a web application, the attacker has information about the web application that can be used to launch an attack. An attacker first chooses the type of attack to be launched and chooses the appropriate attack request from the workload profile of the web application. Three types of attack are used in this work.

  - Random Flood : A Random Flood attack disregards the workload related information in the workload profile of the web application and only takes into consideration the different URLs in the target web application. The requests chosen for launching the attack are random samples from R i.e. $r\prime \in \{r|r \in R\}$.

  - Asymmetric Attack : An asymmetric workload attack utilizes the request workload profile of the web application and chooses the most computationally expensive requests in the web application. In other words, the request $r\prime \in \{r|r \in R \, and \, f(r) = w_k\}$.

  - Stealthy Asymmetric Attack : A more stealthy variation of the asymmetric

workload attack would be to choose requests from the top l workload classes instead of just the top one class. In this case, the request $r\prime \in \{r|r \in R \; and \; f(r) \in \{w_k, w_{k-1}, ..., w_{k-l+1}\}\}$. It can be noted that when $l = k$, this variation of the attack disintegrates to a Random HTTP Flood.

- Launching the Attack : The final stage of generating a DDoS attack is launching the attack itself. The attack must be launched with multiple TCP connections to the target server so that these requests are executed in parallel at the server side.

4.4.2 Experimental Study

4.4.2.1 Experimental Setup

The target web application was hosted on 3.7 GHz Intel Xeon system with 16 cores and 64 GB RAM running Ubuntu 16. Three open source web applications - Opencart, Magento Prestashop - were used for testing. All three test applications are open source e-commerce applications developed in PHP and MySQL. They are used in a number of commercial establishments. All three web applications have around 400 unique URLs and offer a suitable representation of real-world web applications.

Algorithm 1 Algorithm for generating the Request Workload Profile of a Web Application

Input: Set of Seed URLs for the Web Application, $SeedList$
Output: Request Workload Profile of Web Application

1: $URL\_List \leftarrow \phi$
2: **for all** $url \in SeedList$ **do**
3: $\quad Visit \leftarrow url$
4: **end for**
5: **for all** $url \in Visit$ **do**
6: $\quad Workload\_Params \leftarrow Request(url)$
7: $\quad URL\_List \leftarrow URL\_List \cup [url, Workload\_Params]$
8: $\quad Visit \leftarrow Visit \setminus url$
9: $\quad$ **for all** $neighbour\_url \; accessible \; from \; url$ **do**
10: $\quad\quad Visit \leftarrow Visit \cup neighbour\_url$
11: $\quad$ **end for**
12: **end for**
13: $WorkloadProfile \leftarrow CLUSTER(URL\_List, k)$

As described in Section 4.4.1, the first step in generating asymmetric attacks on web applications is to generate a request workload profile of the web application. Algo-

rithm 1 describes the steps involved in web application profiling. A set of Seed URLs, denoted by $SeedList$, is provided as input for the profiler. Typically, this includes a single URL pointing to the home page of the web application. Using the set of seed URLs as input, a crawler generates a list of URLs in the web application, denoted by $Visit$. For every URL in $Visit$, the profiler sends a request to the web application for that URL. As the web application processes the request, the parameters corresponding to $Workload\_Params$ are measured and stored in URL_List. The parameters extracted depends on the situation. For example, a penetration tester with access to server resources could measure the response time, CPU usage and memory usage of the server while executing the request. Thus, $Workload\_Params$ could constitute a triple $(ResponseTime, CPU\_Usage, Memory)$ in such a case. An attacker would normally be unable to access server resources, and would be forced to used response time as an indicator of request workload (Ranjan et al. 2009). In such a case, $Workload\_Params$ would consist of a single value denoting the response time of the request. Once all the URLs in $Visit$ are processed, a clustering algorithm is run on URL_List to group all the URLs in the web application into k clusters. These clusters correspond to different workload classes. In our experiments, we have used only response time as the indicator of request workload, and have used $k = 2$. This divides all URLs in the web application into two classes - low workload and high workload. This generates the request workload profile of the web application.

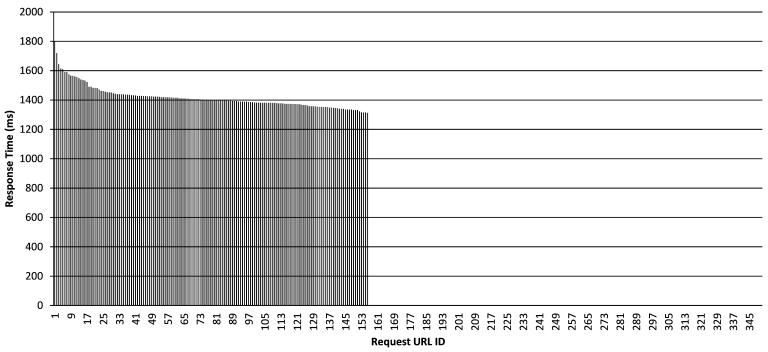


Figure 4.3: Workload Profile for Opencart

Figure 4.3 shows the request workload profile generated for Opencart web application. It can be observed that out of the 350 URLs in the web application, around

155 URLs incurred a sufficiently high response time exceeding 1200 ms. The other URLs required a very low response time, most of which could be completed in under 10 ms. This gives us two workload classes - high workload and low workload. Using the methodology mentioned in Section 4.4.1, three types of attacks were generated - Random HTTP flooding attack, stealthy asymmetric attack and asymmetric attack - at varying request rates. The attack was generated using shell scripts which used the cURL utility to generate and send requests in a specific sequence. The CPU usage of the system when under all three attacks were monitored.

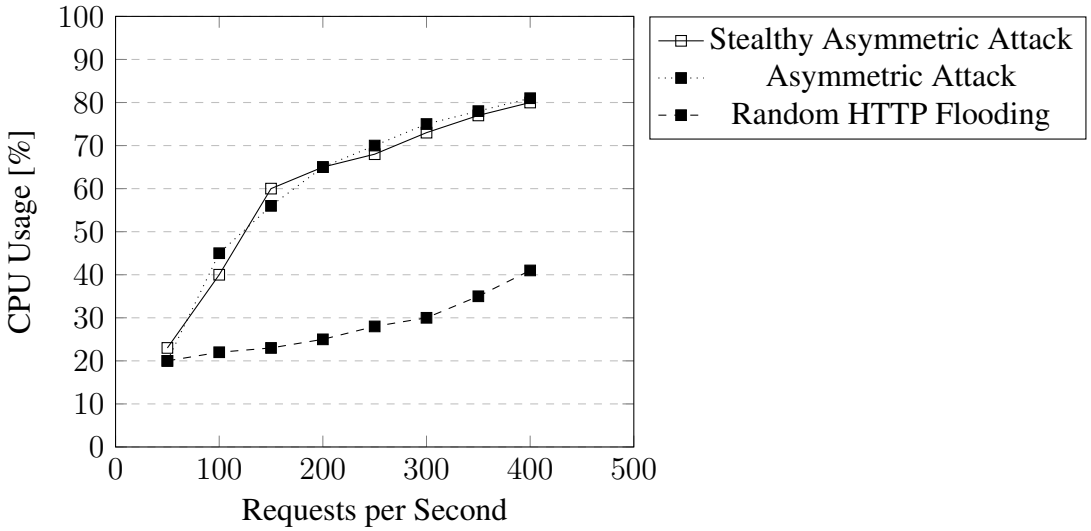


Figure 4.4: Comparison of HTTP Flooding and Asymmetric Attacks on HTTP/1.1 Server

4.4.2.2 Results and Discussion

Figure 4.4 shows how the CPU usage of the target system varies under all three attack plans. The Random HTTP flood only raises the CPU usage of the target system to around 40% for a request rate of 400 requests per second, while both the Stealthy Asymmetric attack and Asymmetric Attack are capable of pushing the CPU usage past 80% for the same request rate. This clearly demonstrates the advantage that attackers possess by using asymmetric attacks in place of randomized floods, and also demonstrates the need to detect these attacks quickly and efficiently.

4.5 DETECTING ASYMMETRIC AL-DDOS ATTACKS ON HTTP/1.1 SERVERS

Our proposed approach for detecting asymmetric AL-DDoS attacks operates in two phases - learning and detection. In the learning phase, the system uses available server logs and access traces to learn and model the behavioural dynamics of legitimate users. In our approach, we use an annotated Probabilistic Timed Automata (PTA) to model user behaviour. During the detection phase, the system intercepts and processes every incoming connection to identify if their behaviour deviates from the learned user behaviour model. Significant and prolonged deviations from the learned model are considered as malicious behaviour, and the connection is blocked. Periodically during the detection phase, the system also uses legitimate run time traces to update the learned model. This is done so that the need for periodic retraining does not arise. Figure 4.5 illustrates the working of the proposed system.

4.5.1 Learning Phase

The learning phase builds a model which describes the behavioural dynamics of users. The features used to describe user behaviour, and the actual model used play a crucial role in the efficiency of detection.

4.5.1.1 Features used to Model User Behaviour

Majority of the existing research works on AL-DDoS detection use indirect representations of user behaviour such as request rate, request inter-arrival duration, session inter-arrival duration, request and response size etc. to model user behaviour and operate on the assumption that an AL-DDoS attack is defined by a sharp increase in the number of requests received by a web application in a short duration. This can be observed by significant deviations in statistical attributes of the request stream which forms the basis for the detection mechanism. However, Ranjan et al. (2009) demonstrated that the use of high workload requests could allow malicious users to launch attacks without any significant deviations in request rate, request inter-arrival duration or other statistical features. Therefore, such detection mechanisms based on statistical attributes of the request stream are unable to detect asymmetric attacks, and a more accurate set of features which effectively describes legitimate user behaviour is required.

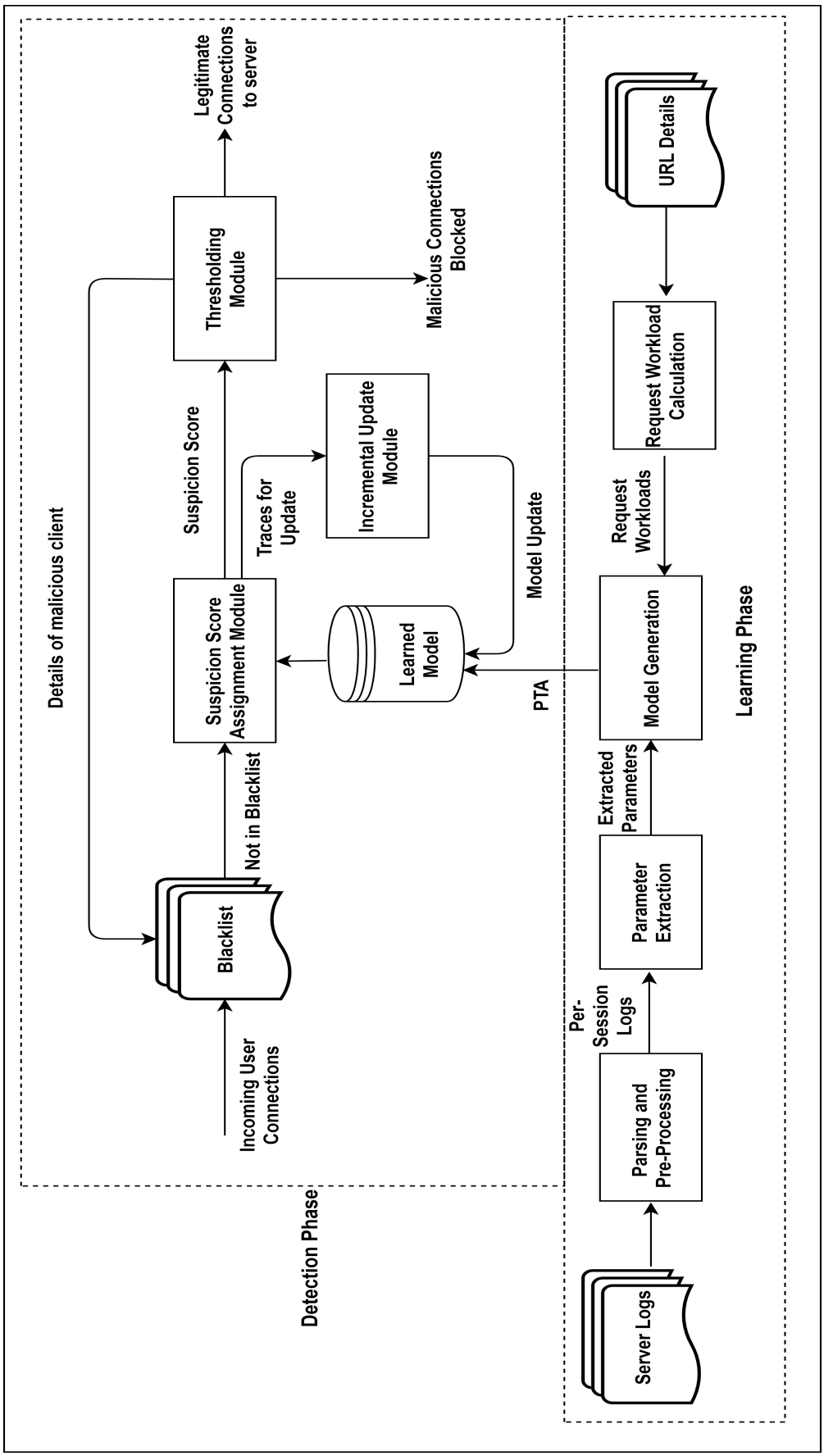


Figure 4.5: Block Diagram of the Proposed Approach

In order to arrive at such a set of features, it is necessary to identify the defining characteristics of legitimate user behaviour. In particular, we identify three defining characteristics:

- Users access some pages and follow certain paths more often than others. Thus there is a hidden purpose behind a legitimate user accessing a web application, which is completely absent in the case of an attacker (Wang et al. 2011; Xie and Yu 2009b).

- Users take time browsing the results of a request before issuing a subsequent request. This once again points to the fact that users have some purpose behind browsing the web application, and that is to obtain a service. In order to obtain that service, a user has to issue a series of requests and every request is dependent on the response to the previous request. Thus, it is imperative that users would take time to browse the results of a request before issuing a subsequent request (Hashemian et al. 2012; Van Hoorn et al. 2008).

- The request workload profile of legitimate users is uneven. Legitimate users browse the web application by issuing a sequence of requests alternating between high, medium and low workload states. On the other hand, an attacker intends to damage a web application as soon as possible, and is likely to issue a sequence of high and medium workload requests repeatedly (Xu et al. 2014).

4.5.1.2 Model Description

Finite State Machines (FSM) have been used to model web applications for a variety of applications, ranging from workload generation, stress testing and identifying vulnerabilities. A web application closely resembles an FSM in the sense that the different URLs in a web application can easily be mapped to states in an FSM and user requests (or hyperlinks) from one page to another can easily represent transitions. As a result, FSMs have been used to model web applications for workload generation, stress testing (Andrews et al. 2005; He et al. 2007; Thirumaran et al. 2011; Zuzak et al. 2011a,b) and vulnerability detection (Deepa et al. 2018).

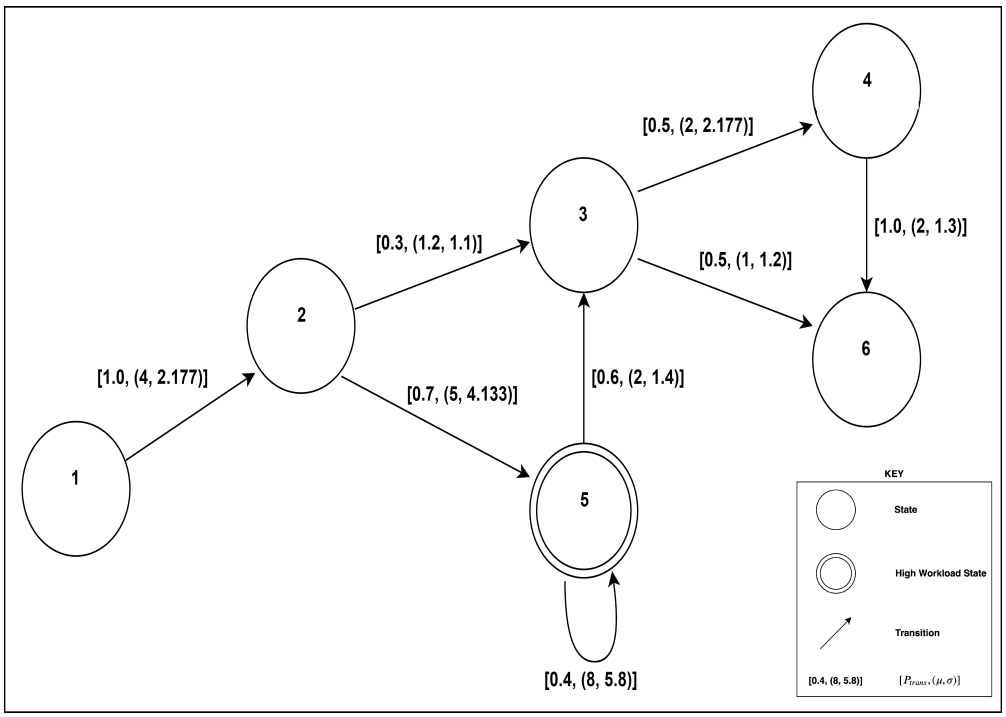


Figure 4.6: Diagrammatic Representation of the annotated PTA

Probabilistic Timed Automata (PTA) are an extension of FSMs which are capable of modelling probabilistic user behaviour. A PTA is a collection of states and transitions, like an FSM, but with some additional features. Every transition in a PTA has an associated weight which denotes the probability of that transition. Every state in a PTA also has a time value associated with it, which represents the average time for which the system remains in that state before making a transition. With the added features, a PTA can be used to model user access patterns in a web application better than an FSM. PTAs have also been used extensively in modelling and testing web applications (Abbors et al. 2013; Gao et al. 2011; Ghezzi et al. 2014).

In this work we model the access patterns in a web application using an annotated Probabilistic Timed Automaton. The URLs or web pages in the web application are represented by states in our model, and the requests between pages are represented by transitions. As expected with a PTA, every transition has a weight associated with it, which represents the probability of users making that transition. Every transition also has a ”think time” associated with it which represents the average amount of time users spend it the previous state before making that transition. Apart from these two

parameters, we also associate a workload measure with every state in the model which represents the computation that the server requires for executing that particular request.

Daniele Beauquier introduced a general definition of a Probabilistic Timed Automaton Beauquier (2003) with multiple clocks and multiple actions that can be performed in a state. When modelling a web application, the complexity of the model can be reduced by considering only a single clock maintained by the web server. Also the actions performed in a state are deterministic and atomic, meaning there is no need to hold on the assumption of multiple actions per state. In our work, we use a simplified model of a PTA with a few annotations.

The annotated Probabilistic Timed Automaton used in our work is a septuple $A = (S, S_{init}, C, W, E, F, \rho, \Phi)$ such that

- S is the set of states in our model, where each state represents a base URL in the web application.
- S_{init} denotes the initial state(s) of the automaton, which is usually the login page of the web application. However, in architectures where there is no concept of separate user logins, every state of the automaton can be considered to be an initial state.
- C is the clock which keeps track of transition timings in our model.
- $W : S \rightarrow \mathbb{N}$ is a priority function which assigns a priority to every state in the model. The priority value is a representation of the computation performed by the web application in that state.
- $E \subset S \times S$ represents the set of transitions in our model. Each edge is a tuple $(s, s\prime)$ such that $s, s\prime \in S$ represent the source and destination states of the transition.
- $F \subseteq S$ is the final state of the automaton. Intuitively, the logout page is the final state of the automaton. However, there are an abundance of scenarios where users simply abandon sessions instead of logging out. To address this situation we consider every state of the automaton as a final state.

- $\rho : E \rightarrow [0, 1]$ is the transition function which maps every transition $(s, s\prime)$ in the model to a probability, which represents the probability of a transition from state s to $s\prime$ in the web application.

- $\Phi : E \rightarrow guard(C)$ is a function mapping every transition $(s, s\prime)$ in the model to a timing constraint within $guard(C)$. $guard(C)$ specifies a set of probability distributions, and Φ maps every transition from s to $s\prime$ to a probability distribution of think times for that transition. A probability distribution is represented by its mean and standard deviation as a tuple (μ, σ).

Other applications involving a PTA usually maintain a trap state as part of the model. Any unauthorized transition, which deviates from the established behaviour, lands the user in a trap state from which they can never perform another transaction. In the case of a web application, however, transitions that deviate from the norm do happen quite often. For example, a user who issues a search query in an e-commerce site usually takes some time to browse the results before issuing a new search query. Hence, we would be right to model such a behaviour using a PTA. However, in some cases, a user may issue a search query, then immediately realizes he input the wrong keywords, or maybe misspelled a word. In such a case, the user might not wait and browse the results of the issued query. The user would issue a subsequent query as soon as possible, which leads to two search queries very close to each other, which deviates from the model. Landing the user in a trap state will increase the false positive rate of our approach considerably, hence we follow an alternate approach, where we assign scores to a user based on how deviant the user behaviour is, when compared to the learned model. The user is blocked only if their scores cross a threshold value, which allows users some relaxation when it comes to deviant browsing behaviour.

A diagrammatic representation of the annotated PTA used in our approach is given in Figure 4.6. Every state in our model is associated with a unique state ID, which is an integer value that is allocated based on a running counter. The use of a state ID allows a more efficient representation of web applications by including dynamic web applications which may use the same URL for multiple web pages. High workload

states are denoted explicitly in the diagram using double circles. Transitions from one state to another are represented by arrows from the source state to the destination state. For every transition, a nested tuple $[P_{trans}, (\mu, \sigma)]$ is given where P_{trans} represents the probability of the transition, and μ and σ represent the mean and standard deviation of the think time distribution for that transition.

4.5.1.3 Suspicion Score Assignment for Detecting Anomalous Clients

When a user logs into the web application for the first time, they are assigned a Suspicion Score value of zero indicating the highest level of trust. As time progresses, their scores keep changing based on how they access the web application.

The three main features that are used in the suspicion score assignment are

- Transition Probability : The higher the probability of the transition that the user is making, the lower the probability that the user is an attacker, and consequently, the user is assigned a lower suspicion score.

- Think Time : Attackers typically try to send requests one after the other without any pause in order to take down a system faster. Based on this fact, if there is a limited time delay between incoming requests as compared to the think time values in the learned model, the access pattern is highly suspicious.

- Workload of the State : Asymmetric DDoS attacks are executed primarily by sending requests to high workload states in the web application. Since such attacks can exhaust server resources much faster than other application layer DDoS attacks, it is necessary to detect these attacks faster. In order to do this, we incorporate a measure of workload into the suspicion score assignment.

A legitimate transition is usually associated with a high transition probability (P_{trans}) and a legitimate think time value (P_{think}) when compared to the think time value for that transition. Since these two events are independent, the probability that an access is legitimate is given by

$$P(LegitimateAccess) = P_{trans} * P_{think} \tag{4.1}$$

Using Equation 4.1, the probability that a user access is anomalous can be given by

$$P(AnomalousAccess) = 1 - P(LegitimateAccess) \quad (4.2)$$

$$= 1 - P_{trans} * P_{think} \quad (4.3)$$

The probability of the transition can be directly obtained from the PTA. The PTA maintains the mean value of the think time (μ) for transitions and their associated standard deviations (σ). If the incoming transition occurs with a think time of t, then a probability that this think time is legitimate can be derived from Chebyshev's Theorem.

$$P_{think} = \left(\frac{\sigma}{\mu - t}\right)^2 \quad (4.4)$$

To ensure that Asymmetric DDoS attacks can be detected faster, we introduce a factor that represents the workload of the state that the user is attempting to access, which is represented by W. Thus, the increment in suspicion score for every request that the user makes is given by

$$SS_{increment} = W * (1 - P_{trans} * P_{think}) \quad (4.5)$$

4.5.1.4 Threshold Determination

The suspicion scores assigned to users indicate the degree of anomalous behaviour displayed by them over a period of time. Threshold determination essentially consists of selecting a value of suspicion score θ such that users who have suspicion score values less than θ can be considered to be legitimate users, and users with values greater than or equal to θ can be considered to be malicious. However, during the learning phase of anomaly detection, the system only has access to legitimate user traces, and hence it can only identify a set of suspicion score values which are all below the threshold θ. Thus, selecting the value of θ is essentially a one-class classification problem where an upper bound of legitimate suspicion score values must be decided based on the observed values.

In most of the cases, the distribution of suspicion score values provides some indication of the underlying distribution and can aid in the selection of the threshold. In the case of a Gaussian distribution of suspicion score values, a number of threshold selection mechanisms are available. However, in the more general case, Extreme Value

Analysis (EVA) can be employed to determine the threshold (Schneider et al. 2019). EVA proposes that for many loss distributions, the distribution of maximum values of a sample can be generalized to a common distribution called Extreme Value Distribution (EVD). This is called the Fisher-Tippet Theorem and forms the basis of EVA. Since threshold determination is essentially concerned only with the maximum values, EVA can be safely employed to determine the threshold.

4.5.1.5 Working of the Learning Phase

During the learning phase, the system builds a model of legitimate user behaviour in the form of an annotated PTA as described in Section 4.5.1.2. The parameters needed to build such a model are extracted from two sources - server logs which describe the workload associated with individual requests, and user access logs which describe user behavioural dynamics. Algorithm 2 describes the process of learning in the form of pseudo code.

Access patterns for multiple users are jumbled within the server access logs. In order to extract user access patterns, they are separated according to individual client accesses and individual session accesses. Client accesses can be identified by using the client IP addresses. For identifying different sessions from the same user, a session time-out value is used. For example, two requests from the same client are considered to be from different sessions if their corresponding timestamps differ by more than the session time-out value. In our case, we have set the time-out value to be 10 minutes as is common in an Apache server. This per-session access logs (I_+) are passed as input to the Learning Phase for constructing the model. The learning phase starts with an uninitialized PTA, and the Parameter Extraction module extracts the required parameters to build the PTA. When a previously unseen URL is encountered in the access logs, a new state is created in set S to represent it. A state is always associated with a workload value which denotes the amount of computation that the state generates at the server side. This information is extracted from the server logs with the help of the $ExtractPriority()$ function. The priority of individual requests can also be pre-computed and stored using the approach described by Mishra et al. (2010).

Algorithm 2 Learning Phase

Input: I_+, a training set of per-session request sequences
Output: Annotated Probabilistic Timed Automata (PTA)

Initialize $pta = (S, s_{init}, C, W, E, F, \rho) = \phi$
while LEARNING **do**
  $I = ReadLine(I_+)$
  $curr\_state \leftarrow ExtractState(I)$
  if $curr\_state$ is initial state **then**
    $s_{init} \leftarrow curr\_state$
  end if
  if $curr\_state$ is final state **then**
    $F \leftarrow F \cup curr\_state$
  end if
  if $curr\_state \notin S$ **then**
    $S \leftarrow S \cup curr\_state$
    $w \leftarrow ExtractPriority(curr\_state)$
    $W \leftarrow W \cup (curr\_state, w)$
  end if
  if $(prev\_state, curr\_state) \notin E$ **then**
    $think\_time \leftarrow ExtractThinkTime(curr\_state, prev\_state)$
    $t \leftarrow CreateTransition(prev\_state, curr\_state, think\_time)$
    $E \leftarrow E \cup t$
  else
    $t \leftarrow ExtractTransition(prev\_state, curr\_state)$
    $UpdateTransition(\rho, t)$
  end if
  $prev\_state \leftarrow curr\_state$
end while
$\theta \leftarrow GetThreshold(pta)$

A transition is defined as a jump from $prev\_state$ to $curr\_state$. If such a transition is previously unknown, a transition associated with the access is created and added to E. In order to create a transition, a function $CreateTransition()$ is called which computes two parameters. The first one is the transition probability which is maintained in ρ. In any per-session access logs, if there is an entry for a request to a URL i followed immediately by a request to URL j, there is a transition from state i to state j in the model of the web application. The probability of the transition can be given by

$$t_{ij} = \frac{n_{ij}}{\sum_{k \in S} n_{ik}} \tag{4.6}$$

where n_{ik} represents the number of times there is a transition from state i to state k in the per-session access logs. The second parameter that needs to be extracted relates to the think time for the users for a particular transition. Every time a transition from state i to state j occurs in a the per-session logs, the associated think time can be extracted by simply subtracting the associated timestamps. However, the number of instances of a single transition, and consequently the think times, could be very large. To represent think time for a transition in a compact way, we represent the think time using the mean (μ) and standard deviation (σ) of the distribution. In case a transition describing the access pattern already exists, the parameters are updated by the $UpdateTransition()$ function.

This process is repeated until the set I_+ is exhausted. Once the learning is done, it is necessary to extract a threshold value for the suspicion score values, which is done by the $GetThreshold()$ function. The multiple ways in which a threshold can be identified are described in Section 4.5.1.4. Once the parameters and threshold are computed, they are stored in the database for future use. This effectively concludes the learning phase.

4.5.2 Detection Phase

During the detection phase, the system intercepts every request coming to the web application. Whenever a new user enters the system, they are assigned a Suspicion Score of zero. Henceforth, every access made by the user to the web application causes their Suspicion Score to change cumulatively based on whether the access is perceived to be legitimate or malicious according to the learned model. If the suspicion score associated

Algorithm 3 Detection Phase

Input: Request R, PTA, Threshold θ
Output: Decision (blocked or allowed)

$updatetraces \leftarrow \phi$
while $True$ **do**
  $Input\ new\ requestR$
  $curr\_state \leftarrow ExtractState(R)$
  $w \leftarrow ExtractPriority(s)$
  $think\_time \leftarrow ExtractThinkTime(prev\_state, curr\_state)$
  if $(prev\_state, curr\_state) \in E$ **then**
    $t \leftarrow ExtractTransition(prev\_state, curr\_state)$
    $p_{trans} \leftarrow \rho(t)$
    $p_{think} \leftarrow GetThinkTimeProbability(t, think\_time)$
  else
    $p_{trans} \leftarrow 0.0$
    $p_{think} \leftarrow 0.0$
  end if
  $ss \leftarrow (1 - p_{trans} * p_{think}).w$
  $SS \leftarrow SS + ss$
  if $SS \geq \theta$ **then**
    $The\ connection\ may\ be\ filtered$
    $updatetraces \leftarrow updatetraces - R$
  else
    $The\ connection\ can\ be\ forwarded\ to\ the\ server$
  end if
  if $SS \leq \theta_{update}$ **then**
    $updatetraces \leftarrow updatetraces \cup R$
  end if
  $prev\_state \leftarrow curr\_state$
  if $Update\ Condition\ is\ met$ **then**
    $Update(pta, updatetraces)$
  end if
end while

with a user crosses a predetermined threshold value θ, the user is considered malicious and blocked. The incorporation of a suspicion score mechanism allows a slight deviation from the learned model to occur. This is essential because although legitimate users follow certain prescribed pathways while browsing the web application, they occasionally make mistakes and deviate from the model. If this minor deviation is considered malicious, it would lead to a huge increase in the false positive rate of the system. Only repeated deviations need to be considered malicious and blocked. The calculation of suspicion score involves three parameters that are easily identified from the PTA - transition probability, think time probability and state priority. The calculation of suspicion score is performed as described in Section 4.5.1.3.

At suitable points during the detection phase, the system also uses traces from legitimate users to update the user model built during the learning phase. This is done in order to avoid the need to periodically retrain the model to account for concept drift. The traces selected for the update are from users who have suspicion scores below a threshold θ_{update}, $\theta_{update} << \theta$. This allows only legitimate traces to update the model, thereby maintaining the integrity of the model. This is done as described in Section 4.5.2.1. Algorithm 3 presents a pseudo code for the detection phase.

4.5.2.1 Incremental Update

Concept Drift is a fundamental challenge in modelling user behaviour in web applications (Stevanovic and Vlajic 2014). User access patterns change over time in a web application. The popularity of web pages keeps changing over time and this change influences access patterns to a large extent. This gradual change in user access patterns renders models obsolete after a point of time. When this happens, the model has to be retrained with the new access patterns. Retraining a model takes a long time and requires considerable amount of computation. Another major point to note is that not all access patterns become obsolete, but still retraining requires abandoning all the information learned and learning new information, even though quite a lot of the information may be the same as earlier. We propose an incremental learning approach using testing data, where the system uses user traces obtained using testing in order to periodically

update the model of the web application. This can be done at regular intervals, or when the load on the system is low.

In order to update the model, we fix a suspicion score threshold, Θ_{update}, such that $\Theta_{update} < \Theta$, below which all users are assumed to be more or less legitimate. These traces are used to build a model of the web application during runtime. Let us assume that for a transition $(s, s\prime, \delta)$, we obtain a transition probability $p_{ij}\prime$ during runtime using testing traces. Also, let p_{ij} be the transition probability of the same transition in the current model. In that case, after the next update,

$$p_{ij}(new) = \alpha * p_{ij}(old) + (1 - \alpha) * p_{ij}\prime \tag{4.7}$$

where α is an anchor value which denotes how much the system should hold on to the initial values learned. This value can be adjusted by the web application developer or system administrator, and can be used to create a stronger model of the web application. For example, in case the model was developed with a sufficiently large set of user traces, it is prudent to assume that user behaviour would have been captured relatively correctly by the model. In such a case, a high anchor value can be used. On the other hand, if relatively few user traces were available during model generation (perhaps because the application is relatively new), the access patterns captured cannot possibly represent the web application well. In this case, the system has to rely more on the traces obtained during runtime, and a low anchor value can be set. From a different perspective, if the web application developer or administrator is confident the user access patterns are unlikely to change (perhaps in a static site), a relatively high anchor value can be used. In case the web application is highly dynamic and user access behaviour changes frequently, a low anchor value can be set.

In the absence of incremental update, the system has a comparatively larger false positive rate, which is detrimental to performance. In order to reduce this FPR, it is necessary to perform incremental update using real time traces at regular intervals. The frequency of update is a critical factor that can impact the performance of the system considerably. Two factors must be considered while determining the update frequency - system load and performance of the detection mechanism. Updating the model frequently using fewer traces allows the update to complete faster, putting less burden on

the system. However, doing so could allow some malicious traces to enter the system and possibly corrupt the system. Updating the model after a considerable amount of time allows more traces to be accumulated, and discourages malicious traces from entering the system. However, this makes the update time consuming and also leads to a slight increase in FPR. This trade-off must be taken into consideration while selecting an update frequency for the system.

4.5.3 Experimental Study

4.5.3.1 Datasets Used

In order to test the efficiency of the system we have used two popular datasets which describe user access behaviour - SDSC-HTTP and CLARKNET-HTTP. SDSC-HTTP contains a day's worth of all HTTP requests to the SDSC WWW server located at the San Diego Supercomputer Center in San Diego, California, and contains 28,338 lines of access logs from 3555 clients. CLARKNET-HTTP is a much larger dataset, and contains two week's worth of all HTTP requests to the ClarkNet WWW server. It has 12,95,853 line of access logs from 77,431 clients and provides a suitable testing dataset. Each line in these access logs describes a request made by a client to the web application, and contain the same information - client address, date and time of access, request URL, status code and request size. Both of these datasets have been used extensively for evaluating AL-DDoS detection mechanisms (Chwalinski et al. 2013a,b).

4.5.3.2 Training and Testing Data

The datasets used were split into two sets - training and testing using an 80-20 split. The training set is used as input to the Learning Phase and is used to build the model of the web application. However, the testing set only contains legitimate traces, and there is a need to generate AL-DDoS attack traces in order to test our detection mechanism. A total of five different kinds of attacks were generated. The first attack generated is a random flood, which randomly sends requests to a web server. The second attack generated is the Single URL flood, which is the most common flooding attack. Instead of randomly selecting from the available URLs, it sends repeated requests to a single URL. This is one of the most common forms of DDoS attack in practice. Attacks A3

and A4 replicate these same attacks (A1 and A2) but with asymmetric attacks. Attack A3 is a random asymmetric attack and A4 is a single URL asymmetric attack. Attack A5 is another popular attack variation wherein the attacker follows a predefined script. This attack, which we call a Botnet based attack due to its use in botnets, creates an attack script by randomly selecting a set of URLs. Once the script is generated, then the attacking connections repeatedly send requests to all the URLs in the script in the same order. Thus, the sequence of requests generated keeps repeating itself at regular intervals. The details of the attacks generated are given in Table 4.2. The generated attack traces were intermixed into the testing dataset and used for testing the proposed approach.

4.5.3.3 Experimental Setup

A prototype of the proposed system was developed using the Go programming language. The prototype of the system was tested on a Dell Optiplex machine with 16 GB RAM running Ubuntu 16.04. The training set was used as input to the Learning Phase to build a model of legitimate user behaviour in the form of an annotated PTA. During the detection phase, the testing set was used to test the performance of the system.

Table 4.2: Types of Attacks Generated for Testing

Attack Code	Attack Type	Request Workload	No. of URLs (Single/Multiple)	Request Sequence
A1	Random Flood	Any	Multiple	Random
A2	Single URL Flood	Any	Single	Fixed
A3	Random Asymmetric	High	Multiple	Random
A4	Single URL Asymmetric	High	Single	Fixed
A5	Botnet based Attack	Any	Multiple	Fixed

4.5.3.4 Results and Discussion

Validity of the Proposed Approach: The proposed identifies AL-DDoS attacks by assigning suspicion scores to users accessing the web application. This assignment is based on the behavioural dynamics of users, and the amount of deviation from the learned model as described in Section 4.5.1.3. Figure 4.7 depicts a histogram of suspicion score values for both legitimate and attack connections. It can be observed that

the suspicion scores for legitimate users tends to cluster at lower suspicion score values, while the scores for attackers are more evenly spread out with generally higher suspicion score values. This clearly shows that the proposed approach can be used effectively for detecting asymmetric AL-DDoS attacks.

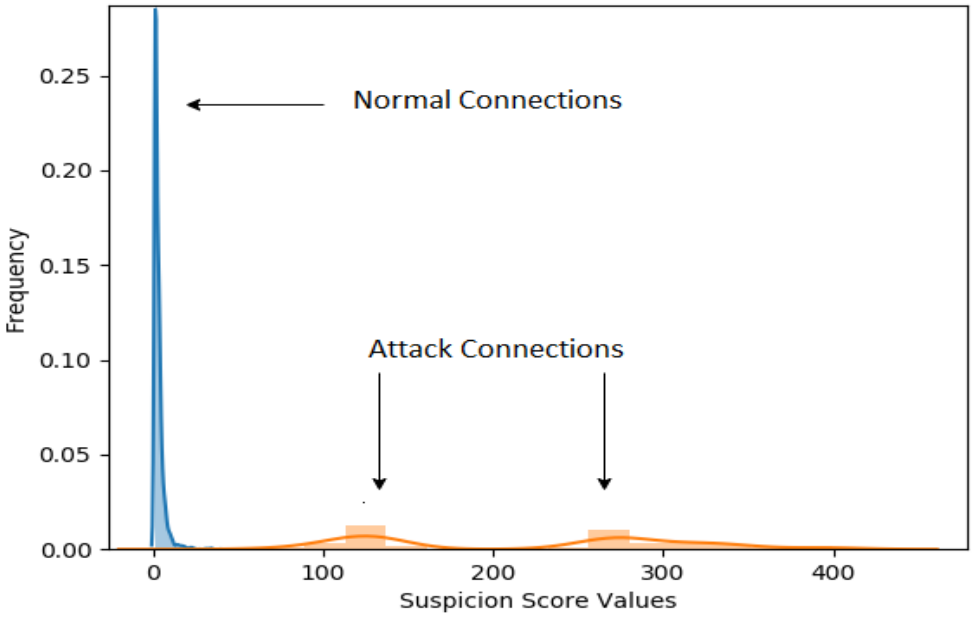


Figure 4.7: Suspicion Score Distribution

Table 4.3: Performance Overview of Attack Detection Module

Parameter	**SDSC-HTTP**	**CLARKNET-HTTP**
Detection Rate	99.8%	99.7%
False Positive Rate (FPR)	0	0.008%
False Negative Rate (FNR)	0.0015%	0.003%
Precision	100%	99.93%
F1 Measure	0.9989	0.9981

Performance of the Detection Mechanism: Table 4.3 gives the overall performance of our approach on different datasets. It can be observed that the proposed approach gives a very high value of precision and recall, with very low false positive value. Table 4.4 gives a more detailed view of the performance, and describes how well the tool performed in detecting individual attacks. While most of the attacks could be detected with an accuracy of 100%, there were a few cases where the detection mechanism was

Table 4.4: Attack-wise Performance of Attack Detection Module

Attack Type	**Detection Rate (%)**	
	SDSC	**CLARKNET**
Random URL Flood	100	100
Single URL Flood	98	91
Random Asymmetric Attack	100	100
Single URL Asymmetric	98.17	100
Bot Attack	100	100

unable to detect the incidence of an attack. This is largely due to the incomplete set of traces available for modelling legitimate user behaviour. The completeness of the model and the efficiency of the detection mechanism rely to a large extent on the size and quality of the input logs. Unlike existing detection mechanisms, our proposed approach assigns suspicion scores to incoming connections based on only the attributes of that particular connection, not on global statistics. This makes our approach robust in differentiating between flash crowds and AL-DDoS attacks.

In order to avoid overfitting, the experiments were repeated multiple times using repeated random sub-sampling cross validation. We observed that the detection mechanism displays a similar performance on all random sub-samples, which indicates that the detection mechanism learns the behavioural dynamics of users efficiently, and that the proposed model does not overfit.

Complexity: Algorithm 3 describes the detection algorithm used in the proposed approach. This algorithm is meant to operate at real time, and describes the real time complexity of the proposed approach. It can be observed that to compute the suspicion score for a sequence of N requests, our approach requires only a single pass over the sequence and hence the complexity of the proposed approach is $O(N)$. Existing approaches which use an HsMM have a computational complexity of $O(N^2)$. The use of a modification of the forward algorithm called the M-Algorithm can reduce this complexity to $O(MN), M < N$, but still the algorithm is computationally expensive for real time use. Thus, our proposed approach clearly outperforms existing detection mechanisms for asymmetric AL-DDoS attacks.

This is further illustrated in Figure 4.8, which compares the execution time for an

HsMM based approach and our proposed approach in a semi log plot. It can be clearly observed that as the sequence length increases, our approach clearly performs better than existing approaches, which further demonstrates that our system is much more suitable for real time use. The actual difference in execution time is considerable - for a sequence length of 50, an HsMM based approach takes around 60 ms while our proposed approach takes under 2 ms for execution.

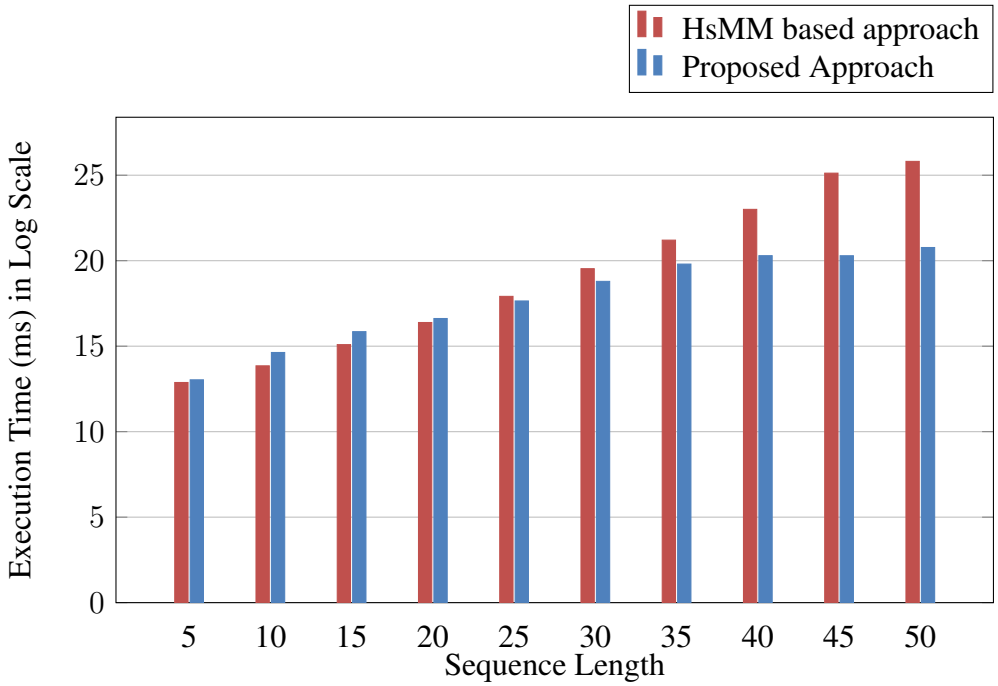
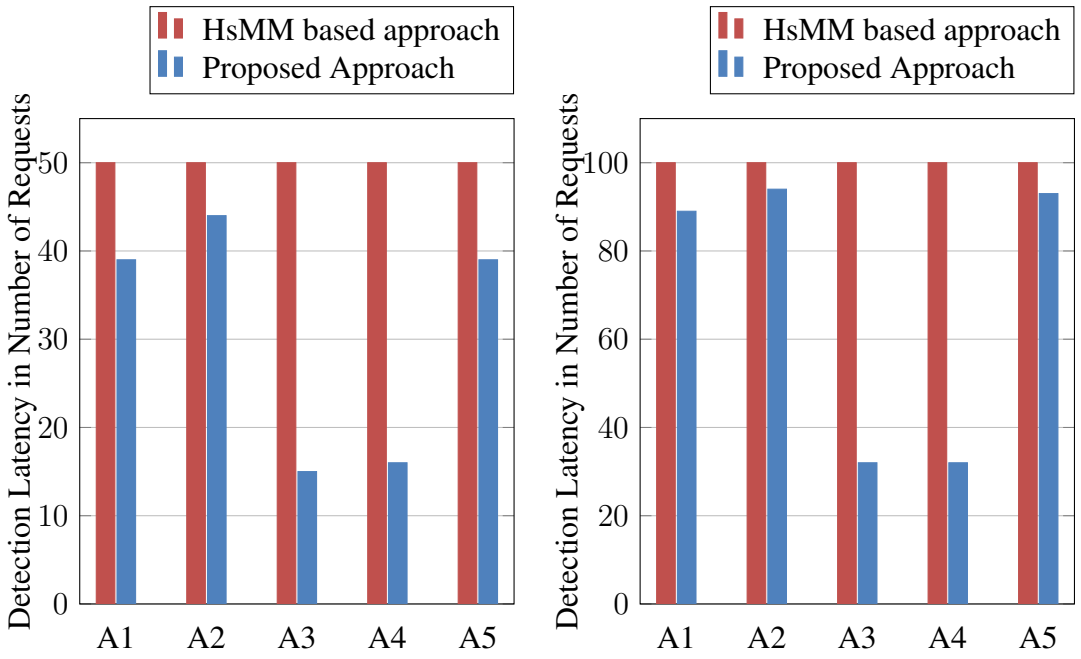


Figure 4.8: Execution Time

Prioritized Detection: Figure 4.9 gives a representation of the time taken for detecting the different classes of attacks as compared with the existing HsMM based approaches. Approaches based on HsMM have a fixed decision length which denotes the number of requests that need to be inspected in order to make a decision. This decision length is constant for all classes of attacks. This essentially means that there is no concept of prioritized detection, and that asymmetric attacks are detected with the same latency as flooding attacks. Figure 4.9 clearly shows that our proposed approach performs better than existing HsMM based approaches in detecting attacks faster. It can also be seen that asymmetric attacks (attacks A3 and A4) are detected much quicker than HTTP flooding attacks. This is particularly due to the inclusion of the request



(a) SDSC (b) CLARKNET

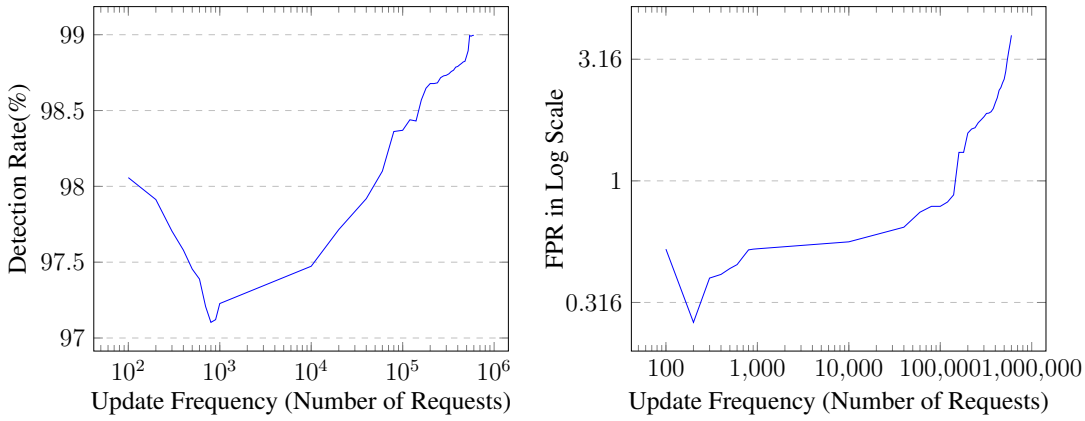
Figure 4.9: Detection Latency

workload parameter in the calculation of suspicion score. This reduced detection latency is particularly important in the case of asymmetric attacks, because they can take down a system with just a few requests.

Adaptability: The ability to learn new user behaviour and update the model at run time is crucial for asymmetric AL-DDoS detection mechanisms. In our approach, the effectiveness of the update is primarily governed by two factors - the frequency of update and the value of the update threshold. In order to test the performance of our incremental update capacity of our model, we separated our dataset into two parts - training and testing. A model of user behaviour was constructed using the training data set. Then, we selected states which were absent in the training data set and generated attack logs using the same approach as described in Section 4.5.3.2. We then combined the testing logs with the attack logs to create a new dataset, which we refer to as the $update - test$ dataset. We then set the update threshold to be half of the attack threshold. We monitored the performance of our detection mechanism both with and without update. We observed that even without any update our detection mechanism has a very high detection rate. This is because any unknown state is treated to be an at-

tack in the absence of threshold and the suspicion score is increased as such. However, this means that any legitimate access with previously unknown states is also treated as anomalous. This is indicated by a comparatively large false positive rate in the absense of any incremental update.

We then tested the performance of our system with periodic updates. It was observed that the update interval plays a crucial role in the performance of the system. If the update interval is set to a small value (say, every 1000 requests), the false positive rate drops considerably, but is also accompanied by a decrease in the detection rate. This is due to the fact that certain attack traces will also be identified as legitimate in the short interval, and be treated as legitimate values. As the update interval increases, the false positive rate remains low, and the detection rate also continues to increase. The impact of update frequency on detection rate and false positive rate is shown in Figures 4.10a and 4.10b respectively.

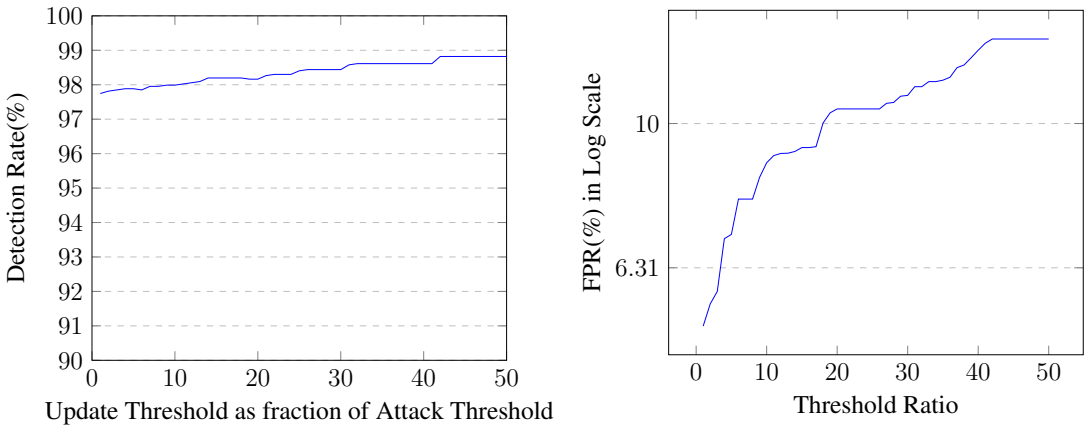


(a) Detection Rate

(b) False Positive Rate

Figure 4.10: Effect of Update Frequency

Figures 4.11a and 4.11b show the impact of the update threshold on the detection rate and FPR respectively. Setting a high update threshold allows some malicious traces to be included in the update which leads to a lower detection rate. However, it also shows a lower FPR due to the impact of the legitimate traces used for updating the model. As the threshold is lowered, fewer traces are used in the update (which indirectly means fewer malicious traces are included), and hence the FPR increases. At the same time, the detection rate also rises. This behaviour is similar to that of the impact of



(a) Detection Rate

(b) False Positive Rate

Figure 4.11: Effect of Update Threshold

update frequency on the detection rate and FPR.

4.6 SUMMARY

Asymmetric AL-DDoS attacks pose a very serious threat to web applications running the HTTP/1.1 protocol, and the detection of these attacks is a daunting task. In this chapter, we demonstrate the potency of asymmetric AL-DDoS attacks as compared to random HTTP flooding attacks, and also illustrate the need for detecting these attacks efficiently. We also present certain features that must be possessed by any detection mechanism for these attacks. In this chapter, we describe out attempt to model the actual behavioural dynamics of legitimate users using a simple annotated Probabilistic Timed Automata (PTA) along with a suspicion scoring mechanism for differentiating between legitimate and malicious users. This allows the detection mechanism to be extremely fast and have a low FPR. In addition, the model can incrementally learn from run-time traces, which makes it adaptable and reduces the FPR further. Experiments on public datasets reveal that our proposed approach has a high detection rate and low FPR and adds negligible overhead to the web server, which makes it ideal for real time use.

CHAPTER 5

ASYMMETRIC AL-DDOS ATTACKS ON HTTP/2 SERVERS

The introduction of HTTP/2 has completely redefined the landscape of Internet communication. With the help of features such as binary framing, header compression, multiplexing and server push, HTTP/2 has considerably improved the performance of web servers and reduced the response time for clients. However, while doing so, it puts additional load on web servers, which has led to concerns in its own right. There is a lack of a quantitative analysis regarding the possibility of asymmetric AL-DDoS attacks on HTTP/2 servers. In particular, there are two important questions that need to be addressed, which forms the focus of this chapter.

- **Performance comparison of HTTP/1.1 and HTTP/2 Servers under AL-DDoS Attacks** : HTTP/2 servers have been criticized as being more vulnerable to AL-DDoS attacks due to the additional load incurred while processing the requests. However, HTTP/2 also possesses a number of performance enhancing features such as header compression and binary framing. There is a need for a quantitative analysis of how an HTTP/2 server performs under AL-DDoS attacks when compared to a server running HTTP/1.1.

- **Exploring the possible misuse of features to launch AL-DDoS attacks** : HTTP/2 introduces a number of new features such as multiplexing and server push. There has been no study related to these features from a security standpoint. More

specifically, there is a need to study whether these features can be misused to launch AL-DDoS attacks against HTTP/2 servers, and how to defend against these attacks.

The rest of this chapter is organized as follows: Section 5.1 discusses how the introduction of HTTP/2 changes the behavioural dynamics of users, and the associated implications on security. Section 5.2 discusses the process of generating asymmetric AL-DDoS attacks against HTTP/2 servers - both with and without multiplexing. We discuss a particularly potent attack against HTTP/2 servers, called a Multiplexed Asymmetric attack and demonstrate that it can exhaust server resources with just a handful of attacking clients. Section 5.3 discusses how our proposed model can be expanded to incorporate new features and how these features can aid in the detection of Multiplexed Asymmetric attacks.

5.1 THE CHANGING USER BEHAVIOURAL DYNAMICS UNDER HTTP/2

HTTP/2 still retains most of the semantics of HTTP/1.1, and the major changes are related to the underlying implementation. This means that the high level behavioural dynamics of users remains the same irrespective of the version of the HTTP protocol. However, the use of multiplexing and server push as part of the HTTP/2 specifications is capable of creating subtle changes in the behavioural dynamics of users.

A request to a web application is typically satisfied by sending one base request for retrieving the textual content and multiple inline requests for retrieving the content required to render the page correctly, such as image, CSS and JavaScript files. Base requests are typically computationally expensive as they initiate some processing at the server while inline requests are usually static and can be processed easily (Ranjan et al. 2009). Since web pages almost always need these inline resources to render properly, a request for a base page is usually followed by requests to its inline resources. HTTP/2 multiplexing and server push essentially seek to remove the overhead for requesting and receiving these resources by bundling all of them together and serving them over different streams. Server push goes one step further and allows web applications to send resources without an explicit request from a client. These two features in HTTP/2

introduce some subtle, yet poignant changes in the behavioural dynamics of users. For example, a client requesting a number of resources at the exact same time would be considered as an attempt at flooding under the HTTP/1.1 protocol, but is perfectly valid under the HTTP/2 protocol. Figure 5.1 illustrates how HTTP/2 changes the behavioural dynamics of users.

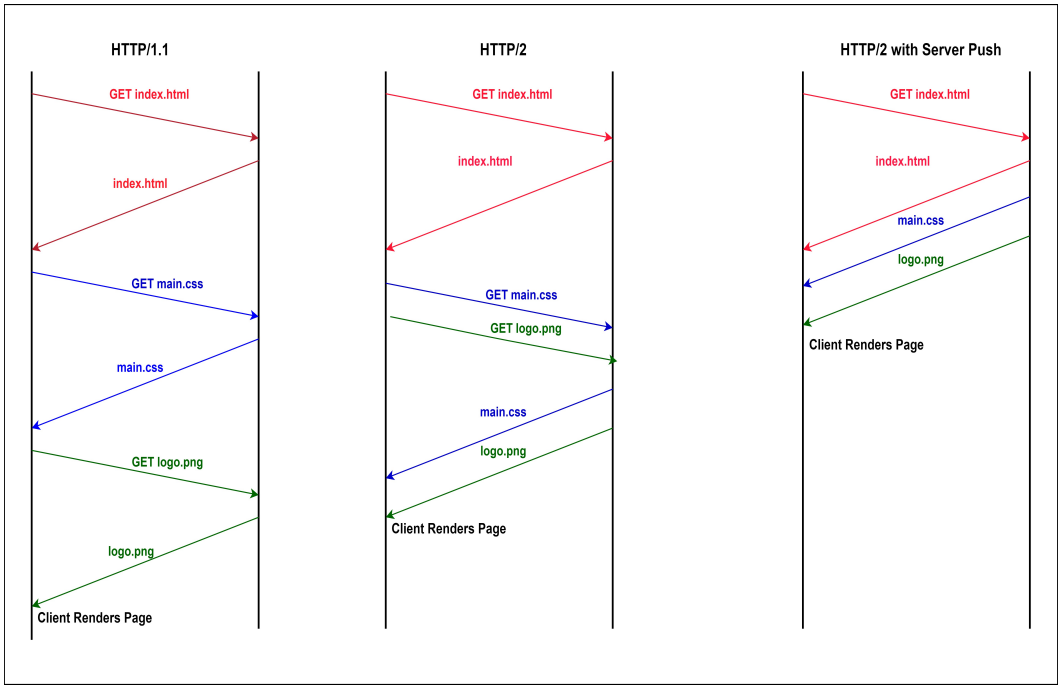


Figure 5.1: The Changing User Behavioural Dynamics under HTTP/2

However, multiplexing has raised some security considerations. Multiplexing essentially removes the need to procure large botnets to launch attacks by allowing multiple, simultaneous requests through a single TCP connection. Also, despite the intended use of multiplexing, there is no limitation on what requests can be multiplexed together. Attackers can take advantage of this and bundle multiple base requests into a single TCP connection and force the server to process them simultaneously. A DoS situation could occur if instead of random base requests, computationally expensive asymmetric requests are multiplexed to form an attack payload.

5.1.1 Multiplexed Asymmetric Attack

We propose an attack called a Multiplexed Asymmetric attack, wherein an attacker multiplexes as many asymmetric workload attack requests as allowed by the server into a single TCP connection, and launches them at the same time. This attack is extremely dangerous because each of the multiplexed requests is by itself an asymmetric workload request which requires intensive computation. The combination of all these requests could potentially bring down a server with just a few attacking systems. Apart from that, the ingress bandwidth generated by such an attack will be extremely low due to the impact of header compression. A stealthier version of the attack can be generated by selecting randomly from among the top k high workload requests. This provides a better camouflage for attackers than using a single URL as the target. In the absence of a proper behavioural analysis of incoming connections a server will fail in detecting such an attack because it utilizes legitimate requests and has been sent through features allowed by the HTTP/2 protocol.

5.1.2 Multiplexed Asymmetric Attack in the presence of Server Push

The proper use of Server Push improves the client's user experience by reducing the page load time, and reducing the client's workload, but simultaneously places the burden on the web server. The web server now needs to serve multiple requests simultaneously or in quick succession. Server Push, thus increases the server utilization when serving requests.

Another aspect of server push is that the server now sends multiple resources back to the client simultaneously leading to an increased egress bandwidth utilization. A particularly complex case may arise when a Multiplexed Asymmetric attack is launched on a server that supports Server Push. In such an attack, the server is forced to serve n requests simultaneously per attacker (where n is the degree of multiplexing) which leads to a huge increase in server utilization. On the other hand, the server now has to send the response for n requests, each of which has multiple associated inline resources (say c). Thus the egress bandwidth is amplified by a factor of $n * c$ when compared to a server running HTTP/1.1. This can lead to a flooding attack which affects the server

egress bandwidth and the nearby routers, causing a DDoS attack at the network layer as well.

5.2 ATTACK GENERATION ON HTTP/2 SERVERS

As in the case of generating asymmetric attacks on HTTP/1.1 servers (which was described in Section 4.4), asymmetric attack generation on HTTP/2 servers also requires a reconnaissance phase which gathers information about the web server. However, the major difference comes while selecting the plan of attack. The HTTP/2 protocol allows for multiple requests to be encapsulated into a single TCP connection. This presents a different avenue of attack against an HTTP/2 server. The entire process of generating an asymmetric workload attack can be summarized in four steps, which is illustrated in Figure 5.2 and Algorithm 4.

5.2.1 Web Application Scanning

The first step in generating an asymmetric workload attack is to identify the structure of the web application. More specifically, an attacker needs to know the different URLs in the web application to identify the most suitable one to target. To identify the structure of a web application, a web crawler can be employed to crawl through the target web application and identify the different URLs.

5.2.2 Identifying High Workload States

The next step is to identify the resource requirements for each request. Average Response time for a request can be used as an approximation of its workload (Ranjan et al. 2009). The mapping from all the request URLs in a web application to their response times (which represents their workload) constitutes the request workload Profile for that web application. Let us consider a web application designed to handle m different types of base requests $R = \{r_1, r_2, ..., r_m\}$ and n different inline requests, $S = \{s_1, s_2, ..., s_n\}$, such that $N = m + n$. These requests introduce different levels of computation which can be approximated to k different classes of workload, $W = \{w_1, w_2, ..., w_k\}, w_1 < w_2 < ... < w_k, k \leq m + n$ into which each on the $m + n$ requests can be mapped to.

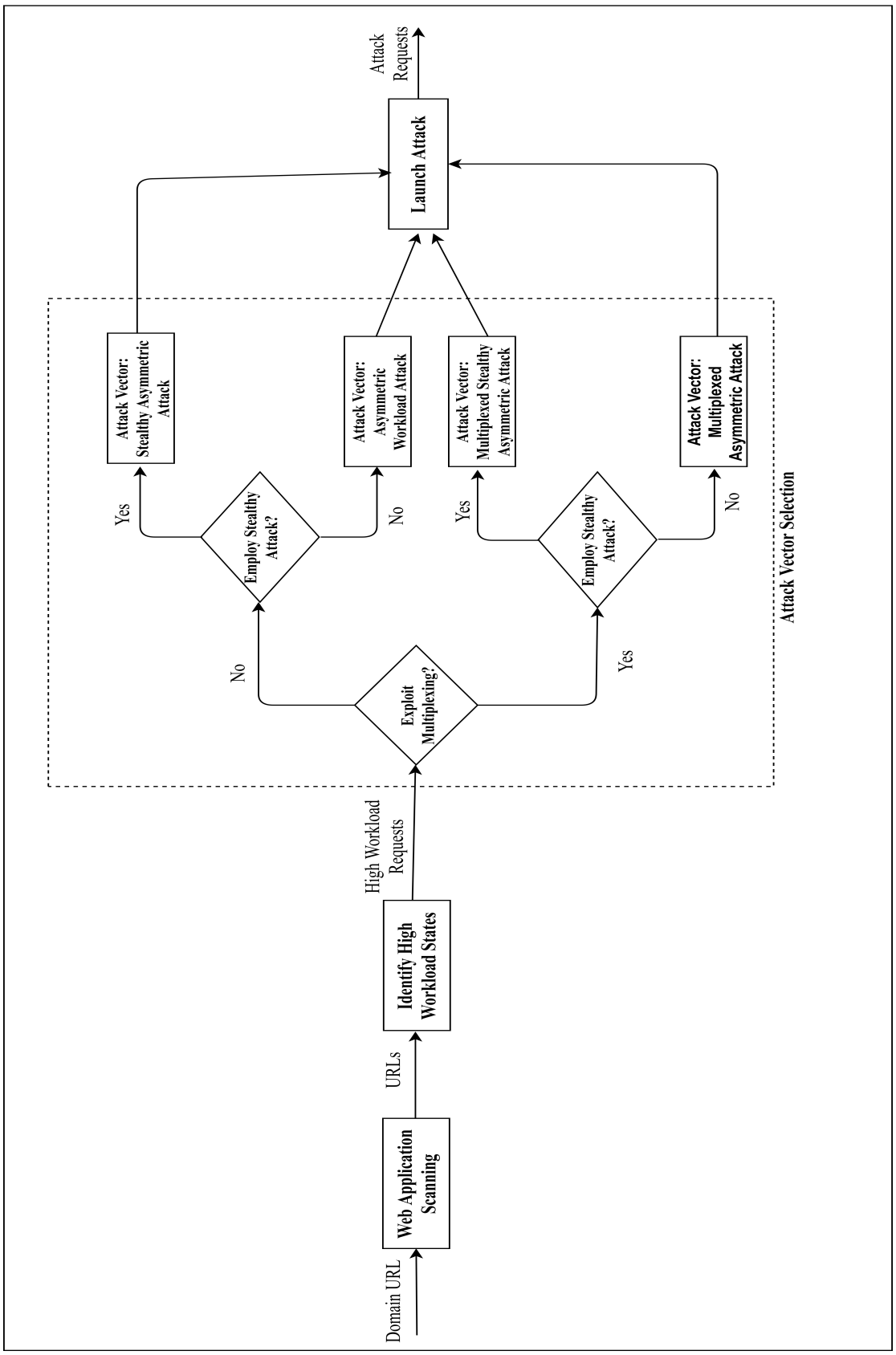


Figure 5.2: Workflow for Generating Asymmetric DDoS Attacks on HTTP/2 Servers

The request workload profile for a web application is essentially a function $f : R \cup S \rightarrow W$ so that $f(r) = w$ gives the workload class w that request r belongs to. Associated with every base request r_i there are x_i inline requests.

5.2.3 Attack Vector Selection

Once the request workload profile is generated for a web application, the attacker has information about the web application that can be used to launch an attack. An attacker first chooses the type of attack to be launched and chooses the appropriate attack request from the workload profile of the web application. Two types of attack are used in this work.

- Asymmetric Attack : An asymmetric workload attack utilizes the request workload profile of the web application and chooses the most computationally expensive requests in the web application. In other words, the request $r\prime \in \{r|r \in R \; and \; f(r) = w_k\}$.

- Stealthy Asymmetric Attack : A more stealthy variation of the asymmetric workload attack would be to choose requests from the top l workload classes instead of just the top one class. In this case, the request $r\prime \in \{r|r \in R \; and \; f(r) \in \{w_k, w_{k-1}, ..., w_{k-l+1}\}\}$. It can be noted that when $l = k$, this variation of the attack disintegrates to a Random HTTP Flood.

Once the request type is decided, the request has to be delivered to the target web application. Depending on how the request is delivered, we have two attack vectors.

5.2.3.1 Simple Asymmetric Attack

A simple asymmetric attack vector establishes ϵ TCP connections to the target web application, potentially using compromised systems, and sends the attack requests through these connections.

The workload incurred by an HTTP/1.1 web server (or an HTTP/2 server without multiplexing) due a single connection at any instant t is

$$W_t = f(r_t) \tag{5.1}$$

Algorithm 4 Algorithm for Generating Attacks on HTTP/2 Servers

Input: Domain name $Domain$ **of target web application**
Output: Asymmetric attack against the target web application is generated

```
1: WorkloadProfile ← Profile(Domain)
2: AttackURL ← ExtractAttackURLs(WorkloadProfile)
3: if MULTIPLEXING_ENABLED then
4:     if STEALTH_MODE_ENABLED then
5:         while TRUE do
6:             while LENGTH(RequestSet) < DegreeOfMultiplexing do
7:                 request ← SelectRandomRequest(AttackURL)
8:                 RequestSet.Add(request)
9:             end while
10:            Launch(RequestSet)
11:        end while
12:    else
13:        while TRUE do
14:            request ← SelectTopRequest(AttackURL)
15:            while LENGTH(RequestSet) < DegreeOfMultiplexing do
16:                RequestSet.Add(request)
17:            end while
18:            Launch(RequestSet)
19:        end while
20:    end if
21: else
22:    if STEALTH_MODE_ENABLED then
23:        while TRUE do
24:            request ← SelectRandomRequest(AttackURL)
25:            Launch(request)
26:        end while
27:    else
28:        request ← SelectTopRequest(AttackURL)
29:        while TRUE do
30:            Launch(request)
31:        end while
32:    end if
33: end if
```

where r_t is the request being processed at time t. If we assume that an attacker has ϵ attacking clients at its disposal, the total workload that is submitted to the server at every instant is:

$$W_t = \alpha * \epsilon * f(r_t) \tag{5.2}$$

where $0 < \alpha \leq 1$ is the Synchronization Efficiency. If all attacking clients can be perfectly synchronized to launch requests at exactly the same time, $\alpha = 1$ and

$$W_t = \epsilon * f(r_t) \tag{5.3}$$

However, due to issues in synchronization and network delays, all attacking clients will not be able to launch attacks at the exact same time, which leads to a slight drop in attacking efficiency. For any given value of α, the attacker always tries to reduce the number of attacking clients, ϵ, so that fewer resources are needed to launch the attack. This can be accomplished by maximizing $f(r_t)$ by sending the most computationally expensive requests in the web applications. This corresponds to using asymmetric workload requests to launch the attack, in which case the workload becomes

$$W_t = \alpha * \epsilon * w_k \tag{5.4}$$

5.2.3.2 Multiplexed Asymmetric Attack

Multiplexing in HTTP/2 allows attackers to send multiple requests in the same TCP connection, which means that these requests will arrive and be executed by the server at approximately the same time. We examine the impact of such an attack in two situations - with server push off and server push on.

Case I: Multiplexed Asymmetric Attack with Server Push Off: In such an attack, every attacking client establishes a single TCP connection and sends ω asymmetric workload requests simultaneously to the server. Thus, the workload that can be delivered to a server using one attacking client becomes

$$W_t = \omega * w_k; 2 \leq \omega \leq \Omega \tag{5.5}$$

where ω is the degree of multiplexing employed by the attacker and Ω is the maximum degree of multiplexing configured at the server. If the attacker can make use of ϵ attacking clients, the total workload delivered at the server will be

$$W_t = \alpha * \epsilon * \omega * w_k \tag{5.6}$$

Case II: Multiplexed Asymmetric Attack with Server Push On: When Server Push is also enabled, the server not only processes the request that it receives, but has to send the associated inline requests as well. Let us assume that every web page has x inline requests embedded in it on an average. In such a scenario, the workload delivered to the server by a single attacking client is

$$W_t = w_k + \sum_{j=1}^{x} f(s_j) \tag{5.7}$$

Assuming that inline requests do not exhibit the wide variation in computational complexity that base requests commonly do, the workload associated with a single HTTP/2 request can be reduced to:

$$W_t = w_k + W_{push} \tag{5.8a}$$

$$W_{push} = x * mu_s \tag{5.8b}$$

$$\mu_s = \frac{\sum_{j=1}^{n} f(s_j)}{n} \tag{5.8c}$$

When multiplexing is enabled along with server push, using ϵ attacking clients, the total workload delivered to the server becomes

$$W_t = \alpha * \epsilon * \omega * (w_k + W_{push}) \tag{5.9}$$

For exhausting the server resources, using a Multiplexed Asymmetric attack seems to be the best option, as it amplifies the attacking power by a factor of ω, which is the degree of multiplexing. It is evident that the attacker benefits the most by choosing asymmetric attack workload requests, and using the full strength of multiplexing offered by the server. In such a case the workload delivered to the web application becomes

$$W_{max} = \alpha * \epsilon * \Omega * w_k \tag{5.10}$$

In case Server Push is also enabled, the workload rises further to become

$$W_{max} = \alpha * \epsilon * \Omega * (w_k + \mu * x) \tag{5.11}$$

5.2.4 Launching the Attack

The fourth and final step associated with an asymmetric attack is the actual launching of the attack itself. This can be done by any of the HTTP request generation tools that support HTTP/2 such as nghttp2.

5.2.5 Experimental Study

5.2.5.1 Server Configuration

In our test setup, the victim is an Apache 2.4 web server on a system with an Intel Xeon 3.70 Ghz CPU, with 64 GB RAM running Ubuntu 18.04. We tested our proposed attack model on three e-commerce web applications - Opencart, Magento and Prestashop. Since our goal is to compare the performance of HTTP/1.1 and HTTP/2, our web server was configured to support both versions of the protocol. Also, since none of the existing implementations of HTTP/2 support unencrypted versions of the protocol, we compared the performance of HTTP/2 against HTTP/1.1 with SSL (HTTPS). Apache was configured to use the Worker MPM model and was configured to accept a maximum of 5000 simultaneous requests.

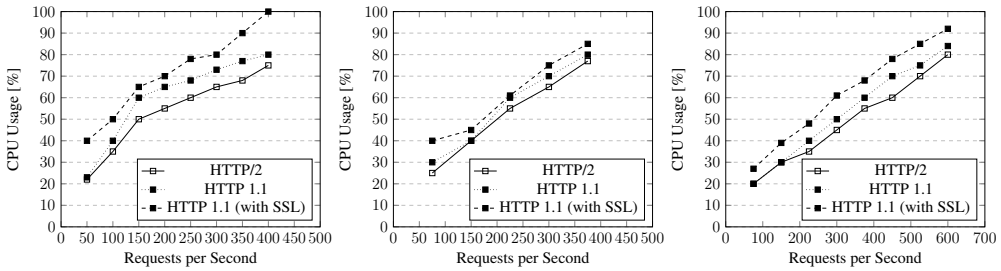
5.2.5.2 Attack Tools

A Linux shell script was written to crawl the web application and identify the different URLs or requests. The script then sends requests to each of these URLs and logs the average response time. Details regarding the response times for different URLs were sorted and the target URLs for asymmetric workload and stealthy asymmetric workload attacks were obtained. These URLs were passed on to the attack scripts for generating and sending requests to the target web server. Of the available request generation tools, nghttp2 provides the most comprehensive coverage of HTTP/2 features, so we have used it in our attack scripts.

5.2.5.3 Results and Discussion

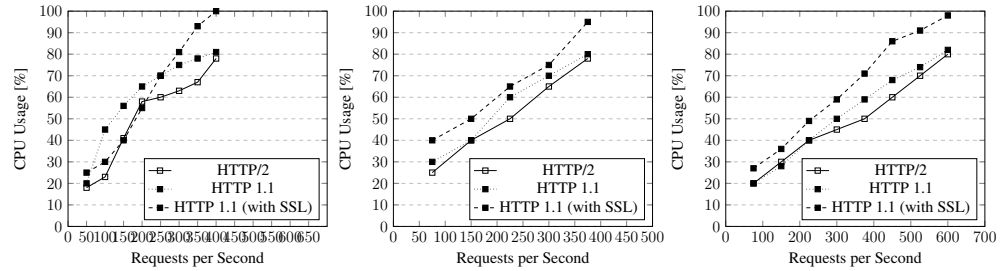
Performance Comparison of HTTP 1.1 and HTTP/2 under a Simple Asymmetric DDoS Attack: Figures 5.3 and 5.4 show how the performance of an HTTP/2 web server compares to that of a web server running HTTP 1.1. Figures 5.3a - 5.3c show the performance of these two protocols during a stealthy asymmetric workload and Figures 5.4a - 5.4c show the performance under an asymmetric workload as the number of attack request rate increases.

The results of this experiment show that HTTP/2 performs considerably better than HTTP/1.1 with SSL when under an asymmetric workload attack. HTTP/2 displays a



(a) Opencart (b) Magento (c) Prestashop

Figure 5.3: Relationship between CPU utilization and Number of Requests in an HTTP/2 Server Under Stealthy Asymmetric DDoS attack



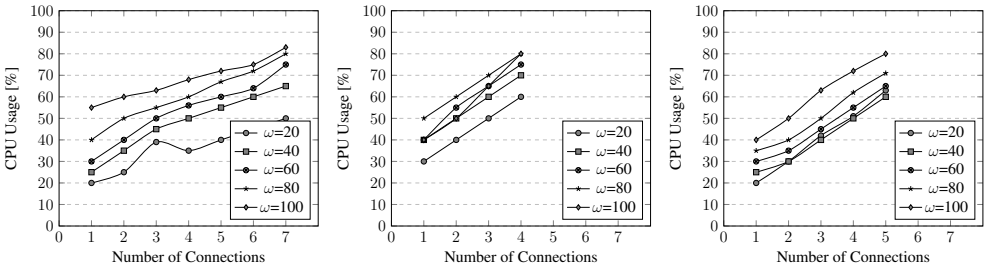
(a) Opencart (b) Magento (c) Prestashop

Figure 5.4: Relationship between CPU utilization and Number of Requests in an HTTP/2 Server Under Asymmetric DDoS attack

performance comparable to HTTP/1.1 without SSL, and even slightly outperforms it. This improved performance can be attributed to the numerous performance overheads in HTTP/2 like binary framing and header compression. This result clearly shows that despite the criticism against HTTP/2 it actually outperforms its predecessor under the same load.

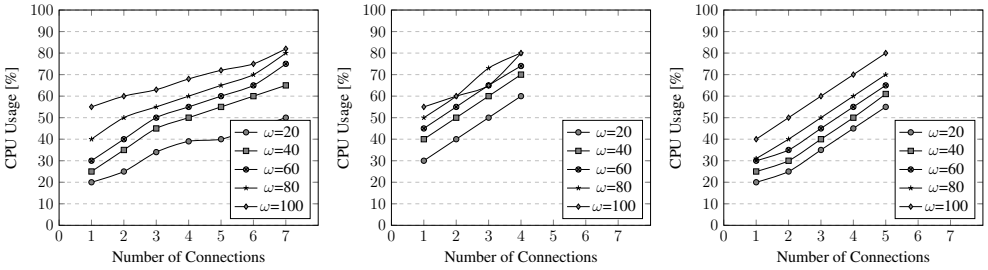
Analyzing the Performance of an HTTP/2 Server under a Multiplexed Asymmetric Attack: Even when a simple asymmetric attack proves effective in bringing down an HTTP/2 server, the ability to multiplex multiple high workload requests into a single TCP connection could prove to be an even bigger challenge for HTTP/2 servers. In this section, we analyze the impact of the said Multiplexed Asymmetric Attack on HTTP/2 servers. Similar to asymmetric workload attacks, a stealthier version of the attack can be employed by randomly selecting from the top k high workload requests.

Figures 5.5 and 5.6 shows how the CPU usage of the target server changes for different degrees of multiplexing and number of attacking clients. The results of this



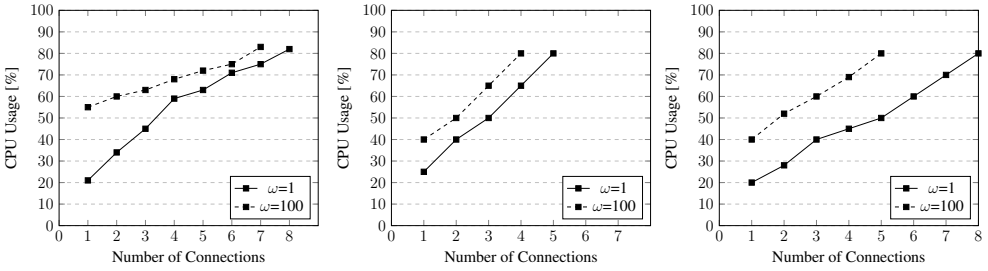
(a) Opencart (b) Magento (c) Prestashop

Figure 5.5: Relation between CPU Usage and Number of Connections during a Stealthy Multiplexed Asymmetric Attack



(a) Opencart (b) Magento (c) Prestashop

Figure 5.6: Relation between CPU Usage and Number of Connections during a Multiplexed Asymmetric Attack



(a) Opencart (b) Magento (c) Prestashop

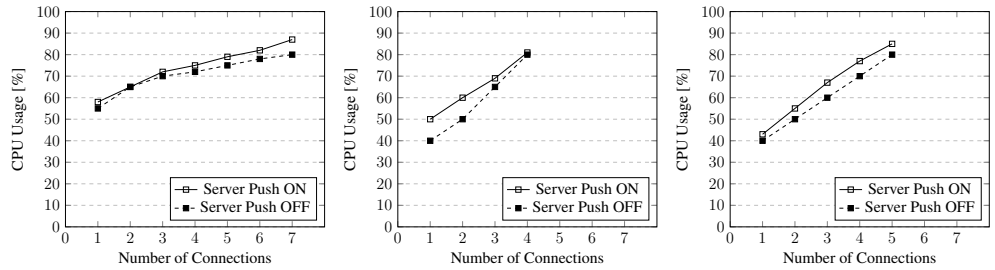
Figure 5.7: Comparison of HTTP/2 Asymmetric Attack with and without Multiplexing

experiment demonstrate that the ability to multiplex requests into a single TCP connection poses a serious security threat, allowing attackers to deliver more asymmetric attack payloads effortlessly. In this case, our target system Magento could be taken down by using just four attacking clients when the degree of multiplexing is 100, with the CPU usage reaching 80%. It can be observed that the multiplicative effect of the attack vectors is not linear. This is due to the effect of the Synchronization Efficiency factor(α), and the difficulty in synchronizing such a large number of attack requests. If proper synchronization can be ensured, the multiplicative effect will increase linearly and the attack will be even more devastating.

Figure 5.7 presents a comparison between HTTP/2 Asymmetric DDoS attack executed using nghttp2 tool with and without multiplexing. Here the degree of multiplexing is set as 100. The multiplicative effect of multiplexing is clearly visible when the number of attacking clients is less. As the number of clients increases, the multiplicative effect decreases due to the effect of the Syncronization Efficiency factor (α). With proper synchronization, the multiplicative effect will become close to ω. However, it can be observed that Multiplexed Asymmetric attacks can cause elevated CPU utilization with fewer resources when compared to simple asymmetric attacks.

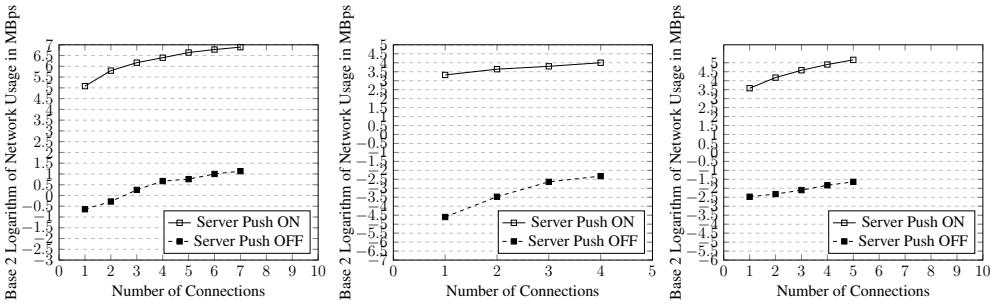
Analyzing the Impact of Server Push on an HTTP/2 Server Under a Multiplexed Asymmetric DDoS Attack: Figure 5.8 depicts the variation in CPU usage of the target server during a Multiplexed Asymmetric Attack (ω=100) when Server Push is turned on and off. The results show that Server Push is able to increase the server CPU utilization, but the increase is not much. According to equations 5.8a, 5.8b and 5.8c, the spike in CPU usage per request due to server push alone is $W_{push} = x_i * \mu_s$ and $\mu_s = \frac{\sum_{j=1}^{n} f(s_j)}{n}$. Here μ_s is the average workload for a inline request. Since the majority of inline requests are simple GET requests and do not perform much server computation, this result is understandable.

Figure 5.9 depicts how the egress network bandwidth varies with the number of attacking clients during a Multiplexed Asymmetric attack (ω=100). Due to the large difference in the values corresponding to network bandwidth in both cases, we have



(a) Opencart (b) Magento (c) Prestashop

Figure 5.8: Impact of Server Push during a Multiplexed Asymmetric Attack on an HTTP/2 Server



(a) Opencart (b) Magento (c) Prestashop

Figure 5.9: Impact of Server Push on Network Bandwidth during a Multiplexed Asymmetric Attack on an HTTP/2 Server

Table 5.1: Effect of Server Push on Egress Network Traffic during a Multiplexed Asymmetric attack for Opencart

Attacking Clients	**Egress Bandwidth (MBps)**	
	Push ON	**Push OFF**
1	0.64	34
2	0.82	56
3	1.2	72
4	1.6	85
5	1.7	100
6	2	110
7	2.2	119

chosen to represent the results using a semi logarithmic graph, with network bandwidth represented in a logarithmic scale with base 2. The actual values of network bandwidth for Opencart are given in Table 5.1.

Both Figure 5.9 and Table 5.1 clearly show that there is a huge increase in network bandwidth when Server Push is enabled on the server. This spike in the egress traffic

volume could choke routers and other network devices adjacent to the server. Most network layer DDoS defenses attempt to detect an increase in ingress traffic and usually do not perform any check on the traffic flowing out of the server. Even in the cases where egress filtering is enabled, it is unlikely to block the attack, because the traffic is flowing to a client with a legitimate IP address.

Despite the small percentage of servers that currently support Server Push, there is a growing effort to understanding the best way to implement and use server push. The number of servers using server push has also been growing steadily. Attacks such as these that exploit multiplexing and server push could become commonplace in the near future, and there is a need to develop pro-active defense mechanisms against them.

5.3 DETECTION OF ASYMMETRIC ATTACKS ON HTTP/2 SERVERS

HTTP/2 follows the same semantics as its predecessor HTTP/1.1 and the major difference exists only in the low level details and implementation. As a result, existing detection mechanisms for asymmetric AL-DDoS attacks for the HTTP/1.1 protocol can easily be adapted to support the HTTP/2 protocol. The only hurdle lies in the fact that HTTP/2 introduces two new features - multiplexing and server push - which essentially allow users to send multiple requests in a single TCP connection, which can be processed simultaneously at the server side. This is diametrically opposite to HTTP/1.1 specifications, and due to this, there will be a considerable difference between the access patterns of users using the two protocols. For example, a large number of requests arriving simultaneously at a server is usually indicative of a DDoS attack in HTTP/1.1 but such a generalization cannot be made in the case of HTTP/2 since it explicitly allows users to send multiple requests simultaneously.

In order to model the behavioural dynamics of users under the HTTP/2 protocol, there is a need to modify the approach proposed in Section 4.5 to incorporate features specific to the HTTP protocol.

5.3.1 Learning Phase

During the learning phase, the system builds a model of legitimate user behaviour, using which attack detection is carried out in the detection phase. The learning phase for the detection of asymmetric AL-DDoS attacks using the HTTP/2 protocol is essentially similar to that of the HTTP/1.1 protocol described in Section 4.5. The input to the learning phase is a set of access logs, which are parsed and examined to extract the features required to model user behaviour. Once the features are extracted and the model is constructed, then the suspicion score threshold is identified. The fundamental difference lies in the inclusion of HTTP/2 specific features to further expand the model. This, in turn, slightly modifies the process of assigning suspicion scores. The modifications made to the model and the suspicion scoring mechanism are described in this section.

5.3.1.1 Features used to Model User Behaviour

The major modification in the behavioral dynamics of users under the HTTP/2 protocol stems from the ability to send multiple requests simultaneously in a single TCP connection. In particular, the following observations can be made regarding the use of multiplexing.

- When users issue a base request, the browser automatically sends the associated inline requests as well. In the HTTP/2 protocol, all these requests can be multiplexed and issued simultaneously. Thus, in a legitimate scenario, a multiplexed set of requests invariably consists of a base requests and/or its associated inline requests.
- Most (but not necessarily all) of the inline requests are sent by users, but this behaviour is notably absent in the case of AL-DDoS attacks. Attack clients may specifically target base requests alone, or may send inline requests haphazardly.

5.3.1.2 Model Expansion

In order to incorporate these features, the annotated PTA described in Section 4.5 is modified and is now represented as $A = (B, I, s_{init}, C, W, E, F, \rho, \Phi, \alpha)$. It can be

Algorithm 5 Learning Phase for HTTP/2 Attack Detection

Input: I_+**, a training set of per-session request sequences**
Output: Annotated Probabilistic Timed Automata (PTA)

Initialize $pta = (S, s_{init}, C, W, E, F, \rho, \alpha) = \phi$
while LEARNING **do**
  $I = ReadLine(I_+)$
  $curr\_state \leftarrow ExtractState(I)$
  if $curr\_state$ *is not a base state* **then**
    if $curr\_state \notin \alpha[prev\_state]$ **then**
      $a \leftarrow CreateAssociation(prev\_state, curr\_state)$
      $\alpha \leftarrow \alpha \cup a$
    else
      $UpdateAssociation(prev\_state, curr\_state)$
    end if
  else
    if $curr\_state$ is initial state **then**
      $s_{init} \leftarrow curr\_state$
    end if
    if $curr\_state$ is final state **then**
      $F \leftarrow F \cup curr\_state$
    end if
    if $curr\_state \notin S$ **then**
      $S \leftarrow S \cup curr\_state$
      $w \leftarrow ExtractPriority(curr\_state)$
      $W \leftarrow W \cup (curr\_state, w)$
    end if
    if $(prev\_state, curr\_state) \notin E$ **then**
      $think\_time \leftarrow ExtractThinkTime(curr\_state, prev\_state)$
      $t \leftarrow CreateTransition(prev\_state, curr\_state, think\_time)$
      $E \leftarrow E \cup t$
    else
      $t \leftarrow ExtractTransition(prev\_state, curr\_state)$
      $UpdateTransition(\rho, t)$
    end if
    $prev\_state \leftarrow curr\_state$
  end if
end while
$\theta \leftarrow GetThreshold(pta)$

noted that the set of states S in the earlier model has now been segregated into two sets - B and I. B is the set of base states in our model, where each state represents a base URL in the web application. I is the set of inline states in our model, where each state represents an inline URL in the web application.

The other modification made to the model is the inclusion of a new association function α. The association function is defined as $\alpha : B \rightarrow (I \rightarrow [0, 1])$, and maps every base state B in the model to a fuzzy set described by the function $I \rightarrow [0, 1]$. Every base state is associated with multiple inline requests which are fired automatically when the base request is made. Due to caching and other network discrepancies, not all of these inline requests are made every time and some requests are more likely to be made than others. This is represented by the fuzzy set $I \rightarrow [0, 1]$. α essentially maps a base state to a fuzzy set describing its inline requests.

$$A_b = \{(y, \mu_y) | y \; is \; an \; inline \; request \; associated \; with \; base \; request \; b\} \tag{5.12}$$

where μ_y is the degree of inclusion of y in A_b. This degree of inclusion is essentially the conditional probability of observing the request y given that a request to b has been observed.

$$\mu_y = \frac{P(y \wedge b)}{P(b)} \tag{5.13}$$

where $P(y \wedge b)$ represents the probability of y and b occurring together. A diagrammatic representation of the annotated PTA after modification is given in Figure 5.10.

5.3.1.3 Suspicion Score Assignment for Detecting Malicious Clients

The suspicion score assignment mechanism described in Section 4.5.1.3 can more or less be used for the HTTP/2 protocol as well. However, in the case of an inline request, such an approach is inadequate. This is because inline requests do not follow a fixed sequence as they are requested by the browser and not the user. However, we can still identify certain features which can be used to identify malicious behaviour for inline requests.

- Association Probability: Association probability of an inline request with respect to a base request denotes the probability with which the inline request is sent following the base request. Attack scripts tend to be ignorant of which inline

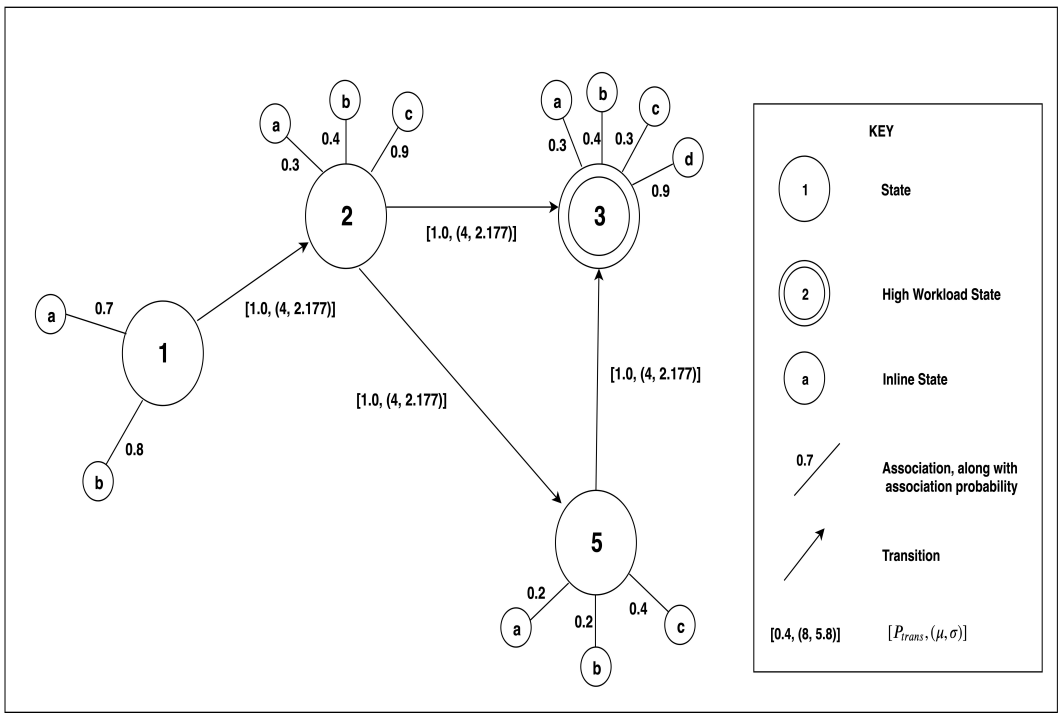


Figure 5.10: Diagrammatic Representation of the modified annotated PTA

resources are associated with a base page, and hence end up requesting random inline resources. This behaviour can be penalized by including a parameter related to association probability in the calculation of suspicion score. This value is essentially the conditional probability of observing an inline request i given a base request b ($P(i|b)$).

- Set Completion: Set completion denotes how many of the inline requests associated with the base request have actually been requested. This feature is important because legitimate users typically request a fair amount of inline resources associated with a base page, but attack scripts typically ignore these requests.

- State Workload: Similar to a base request, we incorporate a measure of state workload in computing the suspicion score for an inline request as well. However, it must be noted that most of the inline requests have a relatively low computational requirements, so this term will likely be set to 1 in all cases.

The association set for a base request is maintained as a fuzzy set, where the set consists of all the inline requests associated with the state and the probability that they are

requested. This constitutes the learned model of the fuzzy set, A. For a particular instance, let B represent the actual set of inline resources that were requested. Obviously, B will be a crisp set, defined as follows:

$$b_i = \begin{cases} 1, if\ the\ i^{th}\ inline\ resource\ is\ requested \\ 0, otherwise \end{cases}$$

Then, the probability of set completion can be defined as the conditional probability of observing the crisp set B given that the underlying distribution corresponds to state A.

$$P_{setcompletion} = P(B|A) \tag{5.14}$$

$$= \frac{\sum_{i=1}^{n} min(a_i, b_i)}{\sum_{i=1}^{n} a_i} \tag{5.15}$$

We define the change in suspicion score as:

$$SS_{increment} = (\sum (1 - P(i|b)).w)(1 - P_{setcompletion}) \tag{5.16}$$

Algorithm 5 describes the learning phase for HTTP/2 attack detection in pseudocode.

5.3.2 Detection Phase

The attack detection phase for the HTTP/2 protocol works essentially the same as that for HTTP/1.1. The system acts as a reverse proxy to intercept incoming requests. The incoming connections are assigned a Suspicion Score value as described in Section 5.3.1.3. If the suspicion score for a connection exceeds the threshold, then the connection is blocked. A pseudocode describing the detection phase is given in Algorithm 6.

5.3.3 Experimental Study

Datasets Used: There are no publicly available datasets for the HTTP/2 protocol. So, in order to evaluate our detection mechanism, we converted openly available SDSC-HTTP and CLARKNET-HTTP datasets to resemble HTTP/2 traces. A comparison of HTTP/1.1 and HTTP/2 traces reveal that there is not much difference between the two. The major difference arises in the case of the timing between a base request and its associated inline requests. In the case of HTTP/1.1, there is a small time delay between the base requests and subsequent inline requests, which is absent in HTTP/2.

Algorithm 6 HTTP/2 Attack Detection Phase

Input: Request R, PTA, Threshold θ
Output: Decision (blocked or allowed)

$updatetraces \leftarrow \phi$
while $True$ **do**
  $Input\ new\ request R$
  $curr\_state \leftarrow ExtractState(R)$
  $w \leftarrow ExtractPriority(curr\_state)$
  if $curr\_state\ is\ not\ a\ base\ state$ **then**
    $a \leftarrow ExtractAssociation(prev\_state, curr\_state)$
    $ss_{increment} \leftarrow ss_{increment} + ((1 - \alpha(a)) * w)$
    $setcompletion \leftarrow setcompletion + \alpha(a)$
  else
    $assoc\_sum \leftarrow 0.0$
    for all $assoc\ in\ \alpha(prev\_state)$ **do**
      $assoc\_sum \leftarrow assoc\_sum + \alpha(assoc)$
    end for
    $SS \leftarrow SS + ss_{increment} * (1 - \frac{setcompletion}{assoc\_sum})$
    $ss_{increment} \leftarrow 0.0$
    $setcompletion \leftarrow 0.0$
    $think\_time \leftarrow ExtractThinkTime(prev\_state, curr\_state)$
    if $(prev\_state, curr\_state) \in E$ **then**
      $t \leftarrow ExtractTransition(prev\_state, curr\_state)$
      $p_{trans} \leftarrow \rho(t)$
      $p_{think} \leftarrow GetThinkTimeProbability(t, think\_time)$
    else
      $p_{trans} \leftarrow 0.0$
      $p_{think} \leftarrow 0.0$
    end if
    $ss \leftarrow (1 - p_{trans} * p_{think}) * w$
    $SS \leftarrow SS + ss$
  end if
  if $SS \geq \theta$ **then**
    $The\ connection\ may\ be\ filtered$
    $updatetraces \leftarrow updatetraces - R$
  else
    $The\ connection\ can\ be\ forwarded\ to\ the\ server$
  end if
  if $SS \leq \theta_{update}$ **then**
    $updatetraces \leftarrow updatetraces \cup R$
  end if
  $prev\_state \leftarrow curr\_state$
  if $Update\ Condition\ is\ met$ **then**
    $Update(pta, updatetraces)$
  end if
end while

Training and Testing Data: After converting the available logs to HTTP/2, we divided the logs into training and testing data based on an 80-20 split. Logs corresponding to attacks were generated using Python scripts. While generating attacks, we consider two different classes of attacks - multiplexed and non-multiplexed. Non-multiplexed attacks are those where the degree of multiplexing is assumed to be one. These attacks resemble the attacks generated using the HTTP/1.1 protocol. Multiplexed attacks utilize the feature of multiplexing in HTTP/2 and send multiple requests simultaneously along single TCP connection. While generating the attack logs for multiplexed attacks, two important parameters were considered:

- Type of Request: Based on the type of requests used to launch the attack, we generated traces corresponding to two attacks. The first attack is a Random Multiplexed attack, where we randomly selected requests to use in the attack. The second attack uses only high workload requests, and is the Multiplexed Asymmetric attack.
- Degree of Multiplexing: As the degree of multiplexing increases, the potency of the attack also increases. We have generated attack logs with three different degrees of multiplexing - 10,50 and 100.

Table 5.2: Types of Non-Multiplexed Attacks Generated for Testing

Attack Code	Attack Type	Request Type	Request Workload	No. of URLs	Request Sequence
A1	Random Flood	Base	Any	Multiple	Random
A2	Single URL Flood	Base	Any	Single	Fixed
A3	Random Asymmetric	Base	High	Multiple	Random
A4	Single URL Asymmetric	Base	High	Single	Fixed
A5	Botnet based Attack	Base	Any	Multiple	Fixed
A6	Inline Flood	Inline	Low	Single	Fixed
A7	Random Inline Flood	Inline	Low	Multiple	Random
A8	Inline Bot	Inline	Low	Multiple	Fixed
A9	Combined Random Flood	Base + Inline	Any	Multiple	Random
A10	Combined Bot Attack	Base + Inline	Any	Multiple	Fixed

A summary of non-multiplexed attacks generated for testing are given in Table 5.2 and the details of multiplexed attacks generated are given in Table 5.3.

Table 5.3: Types of Multiplexed Attacks Generated for Testing

Attack Code	Attack Strategy	Request Type	Request Workload	No. of URLs	Degree of Multiplexing
M1	Baseline Attack	Base + Inline	Any	Multiple	10
M2	Baseline Attack	Base + Inline	Any	Multiple	20
M3	Baseline Attack	Base + Inline	Any	Multiple	30
M4	Baseline Attack	Base + Inline	Any	Multiple	50
M5	Random Multiplexed Attack	Base	Any	Single	10
M6	Random Multiplexed Attack	Base	Any	Single	50
M7	Random Multiplexed Attack	Base	Any	Single	100
M8	Multiplexed Asymmetric Attack	Base	High	Single	10
M9	Multiplexed Asymmetric Attack	Base	High	Single	50
M10	Multiplexed Asymmetric Attack	Base	High	Single	100

There are 10 non-multiplexed attacks generated. The first attack generated is a random flood, which randomly sends requests to a web server. The second attack generated is the Single URL flood, which is the most common flooding attack. Instead of randomly selecting from the available URLs, it sends repeated requests to a single URL. This is one of the most common forms of DDoS attack in practice. Attacks A3 and A4 replicate these same attacks (A1 and A2) but with asymmetric attacks. Attack A3 is a random asymmetric attack and A4 is a single URL asymmetric attack. Attack A5 is another popular attack variation wherein the attacker follows a predefined script. This attack, which we call a Botnet based attack due to its use in botnets, creates an attack script by randomly selecting a set of URLs. Once the script is generated, then the attacking connections repeatedly send requests to all the URLs in the script in the same order. Thus, the sequence of requests generated keeps repeating itself at regular intervals. Attack A6 is a single URL flood using inline requests, and A7 is a random flood using inline requests. Attack A8 recreates a botnet-based attack using only inline requests. Attack A9 is a random flood attack using both base and inline requests, and attack A10 is a botnet-based attack using both base and inline requests.

The multiplexed attacks generated can be broadly grouped into three classes - baseline attack, random multiplexed attack and multiplexed asymmetric attack. The predominant feature used in the detection of multiplexed attacks in HTTP/2 is the validity of the bundled set of requests. In other words, HTTP/2 allows certain set of requests to be bundled together (such as a base request and its associated inline requests). A baseline attack uses valid bundles of HTTP/2 requests to launch a flooding attack. Attacks M1-M4 are baseline attacks under different degrees of multiplexing. Attacks M5-M7 are Random Multiplexed attacks, wherein random base requests are multiplexed into a single connection and launched as an attack. Attacks M8-M10 are multiplexed asymmetric attacks, which simultaneously launch multiple high workload requests in a single TCP connection.

These attack logs are combined with legitimate HTTP/2 logs to constitute the testing dataset. A prototype of the system is implemented in the Go programming language. The training dataset is used as input in the Learning phase of the system. During the detection phase, the testing logs are utilized to test the efficiency of the detection mechanism.

5.3.3.1 Results and Discussion

A detailed description and analysis of our proposed approach for detecting asymmetric attacks using the HTTP/2 protocol is given in the following paragraphs. As HTTP/2 is a relatively new field of research, there are no other research works which address the issue of asymmetric AL-DDoS attacks on HTTP/2 servers, and hence, we are unable to present a comparison as in the case of HTTP/1.1.

Validity of the Proposed Approach: The proposed approach for detecting attacks on the HTTP/2 protocol works by assigning suspicion scores for incoming connections. The approach is fundamentally similar to that for the HTTP/1.1 protocol, with the assumption being that malicious connections will have a comparatively higher value of suspicion score than legitimate requests. Figure 5.11 confirms this assumption in the case of HTTP/2 connections. Compared to the HTTP/1.1 protocol, there is a much greater separation between legitimate and malicious connection, with the malicious

connections incurring a suspicion score more than 400.

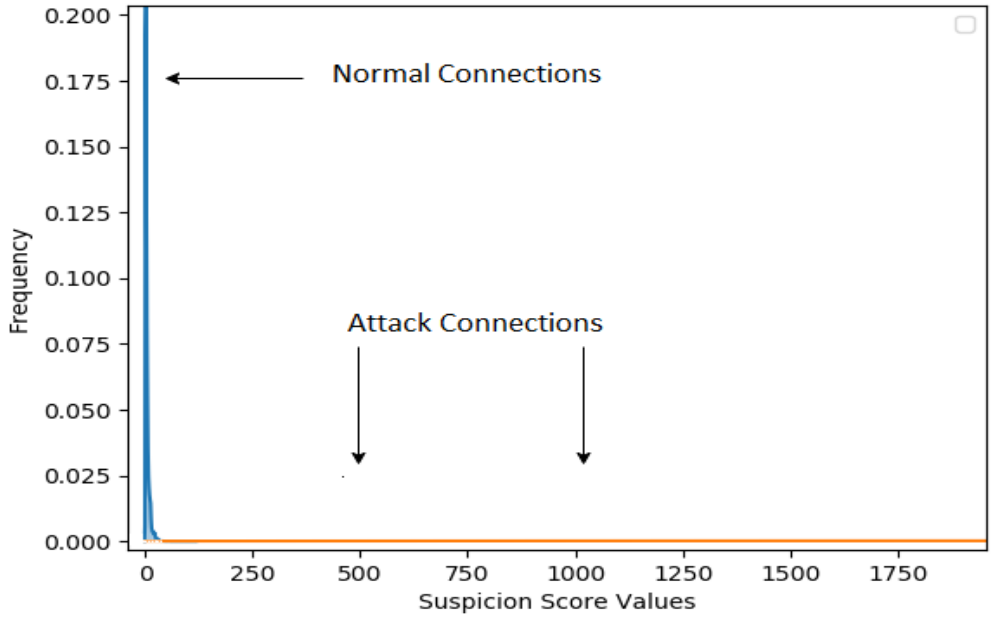


Figure 5.11: Suspicion Score Distribution

Performance of the Detection Mechanism: In the case of non-multiplexed attacks, the system essentially behaves similar to the detection mechanism for HTTP/1.1 with one significant advantage. The inclusion of inline requests into the model allows the system to detect a wider variety of AL-DDoS attacks rather than just asymmetric attacks. A number of flooding attacks, including those generated by using only inline requests, can now be effectively detected by the system. Table 5.4 gives the overall performance of the system and Table 5.5 gives a more detailed description of how the system performs for different classes of attacks.

Table 5.4: Performance Overview of Attack Detection Module

Parameter	**SDSC-HTTP**	**CLARKNET-HTTP**
Detection Rate	99.5%	99.7%
False Positive Rate (FPR)	0.001%	0.002%
False Negative Rate (FNR)	0.002%	0.003%
Precision	100%	99.93%
F1 Measure	0.9989	0.9981

Table 5.5: Attack-wise Performance of Attack Detection Module

Attack Type	**Detection Rate (%)**	
	SDSC	**CLARKNET**
Random URL Flood	100	100
Single URL Flood	98	91
Random Asymmetric Attack	100	100
Single URL Asymmetric	98.17	100
Bot Attack	100	100
Inline Flood	100	100
Random Inline Flood	100	100
Inline Bot	100	100
Combined Random Flood	100	100
Combined Bot Attack	100	100

Prioritized Detection: Figure 5.12 depicts the average detection latency for different classes of non-multiplexed attacks. This is similar to that with the HTTP/1.1 protocol, with the asymmetric attacks (A3 and A4) being identified considerably faster compared to the other attacks.

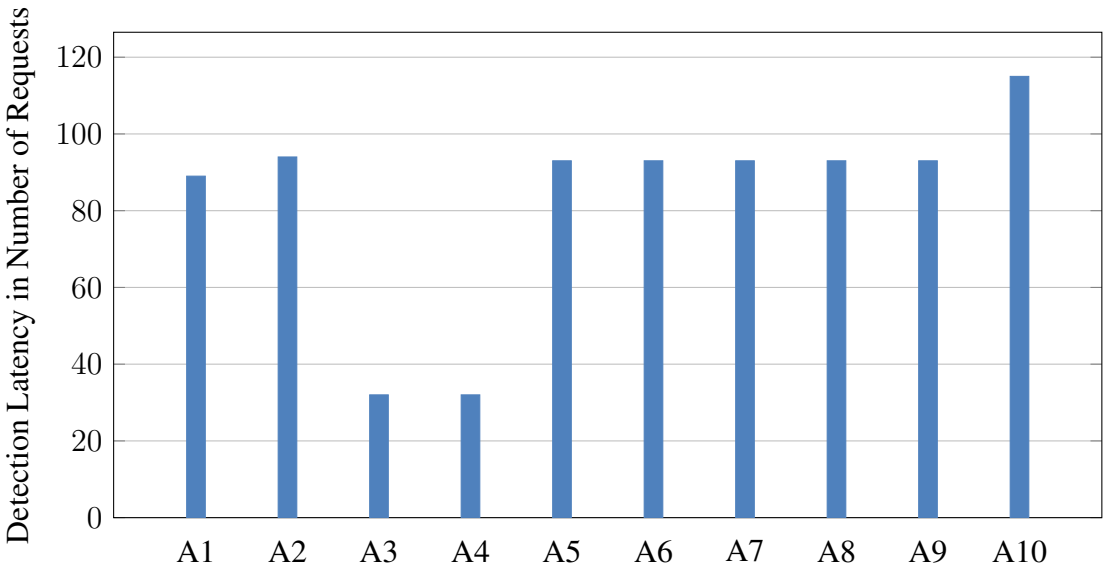


Figure 5.12: Detection Latency

This concept of prioritized detection is even more important in the case of multiplexed attacks. Since HTTP/2 allows multiple requests to be executed simultaneously at the server, it is essential that invalid bundles of requests be blocked from executing simultaneously. Figure 5.13 depicts the average detection latency for multiplexed

attacks. It can be observed that attacks M1-M4 are executed using valid bundles of requests (a base request and its associated inline requests) and hence take longer to detect because they are legal under the HTTP/2 protocol. Attacks M5-M7 are Random Multiplexed attacks, wherein random base requests are multiplexed into a single connection and launched as an attack. These attacks are detected extremely quickly because they represent a departure from the intended use of multiplexing. Attacks M8-M10 are multiplexed asymmetric attacks, which are detected even faster than random multiplexed attacks. It can also be noted that as the degree of multiplexing increases, the associated detection latency decreases, which is extremely prudent.

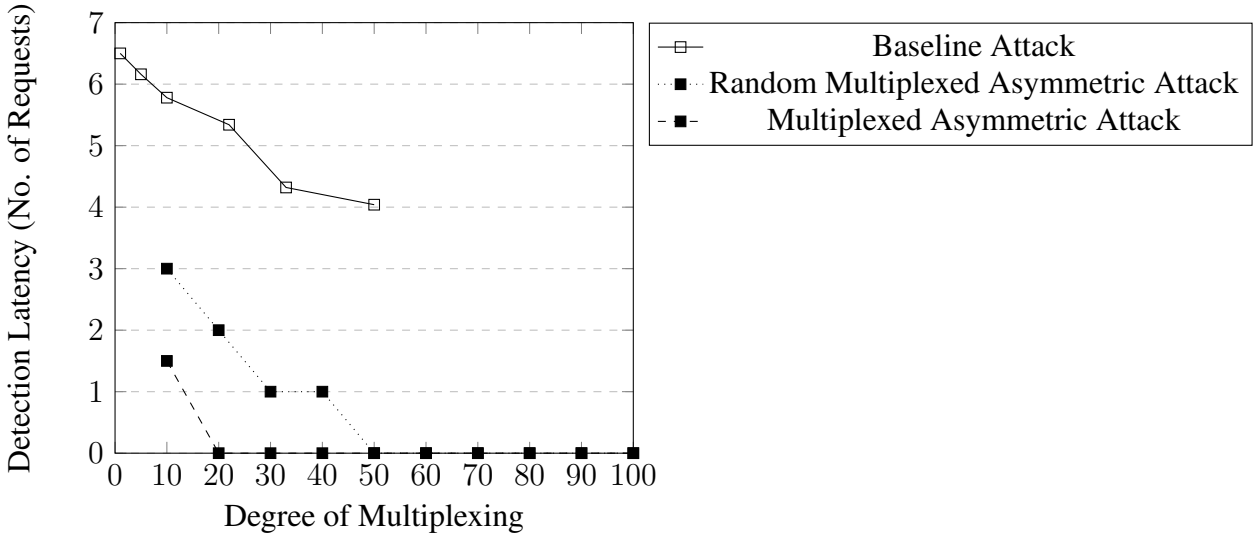


Figure 5.13: Detection Latency for Multiplexed AL-DDoS attacks using the HTTP/2 protocol

5.4 SUMMARY

The introduction of HTTP/2 has led to a dramatic decrease in page load times and has considerably improved user experience. However, it has also faced criticism due to the fact that it adds additional load at the server side, making them vulnerable to DDoS attacks. In this chapter, we compared the performance of HTTP/1.1 and HTTP/2 servers under an asymmetric AL-DDoS attack of the same load. Contrary to the criticism, an HTTP/2 server actually outperforms an HTTP/1.1 server due to the numerous performance enhancing features built into it. However, despite the increased resilience, the newly introduced features of multiplexing and server push can be misused

to launch sophisticated AL-DDoS attacks against HTTP/2 servers. We proposed an attack called Multiplexed Asymmetric attack which misuses the multiplexing feature, and also demonstrated that this attack can exhaust server resources with very few attacking clients. In order to detect these attacks, our proposed model of user behaviour was extended to include HTTP/2 specific features. Experiments on public datasets reveal that the modifications made to our model and detection approach can detect Multiplexed Asymmetric attacks efficiently. In addition, the modifications also allow for the detection of other HTTP/2 based AL-DDoS attacks.

CHAPTER 6

EARLY DETECTION OF AL-DDOS ATTACKS

A consistent theme in all existing research works on DDoS attacks is the effort to detect attacks as early as possible. Early detection of DDoS attacks prevents damage to the web server, prevents service degradation and maintains the availability of the web application. The use of more sophisticated features, global statistics and collaborative knowledge can aid in the early DDoS of network layer DDoS attacks to a considerable extent. However, these approaches cannot be replicated as such to detect AL-DDoS attacks due to the fundamental differences between the two classes of attacks. Nevertheless, even AL-DDoS attacks display a considerable amount of similarity, and the same can be exploited in order to enable early detection of these attacks. In this chapter, we endeavour to develop an Early Detection Module (EDM) for AL-DDoS attacks by exploiting the similarity in AL-DDoS traffic. In particular, our approach for early detection is not dependent on the anomaly detection technique used to detect AL-DDoS attacks, but is a stand-alone module that can be deployed alongside any AL-DDoS detection module.

6.1 SIMILARITY IN DDOS TRAFFIC

DDoS attacks are generated by bots using a common script supplied by the attacker through the C&C server. This means that all the attacking clients in a DDoS attack will follow the same script, and hence generate similar attack traffic (Yu et al. 2011). This behaviour lends a considerable amount of similarity between attack traffic generated by

malicious clients. In particular, we observe two kinds of similarity:

- Temporal Similarity, which indicates a common pattern repeating periodically within a single connection, and
- Spatial Similarity, which denotes the fact that the same pattern repeats across multiple (possibly all) malicious connections

The presence of temporal similarity has been used in DDoS detection considerably. Entropy of the traffic stream is a good indicator of temporal similarity, and can serve as a good indicator of attack for network layer DDoS attacks and some variations of HTTP floods. Spatial similarity on the other hand, can aid in detecting DDoS attacks early by using dynamic signature based detection mechanisms (Katkar and Bhirud 2012; Laskar and Mishra 2016; Networks 2019). Dynamic signature based approaches for DDoS attacks usually have both anomaly detection units and signature detection units. The signature database starts out empty, and any incoming attack has to be solely detected by the anomaly detection unit. Once the anomaly detection unit detects an attack, the system extracts a signature to represent that attack and populates the signature database. Subsequent instances of the same attack can be detected using a signature based approach, which significantly speeds up the detection process, and allows attacks to be detected much earlier. For example, a UDP flood attack targetting a web server consists of a large number of UDP packets coming to a web server. A DDoS detection mechanism could initially identify the attack by observing the deterioration in server performance, and start rate limiting on the incoming traffic. Then, the behavioural DDoS detection mechanism generates an attack signature to help throttle the attack. If the behavioural detection mechanism identifies that all incoming UDP packets have the same combination of flags and packet size, this information is encapsulated into the form of a dynamic signature. Further instances of the attack can be blocked efficiently by filtering for this signature.

The choice of the signature (which is the repeating pattern) is crucial and depends on the type of attack. The signature could consist of a particular IP header field value or the request inter-arrival duration in the case of network layer DDoS attacks. Since AL-

DDoS attacks are executed using legitimate HTTP requests, the attack signature must be crafted using features at the application layer. However, the use of HTTP header fields as attack signatures results in a significant false positive rate. Additionally, most of the existing AL-DDoS generation tools are capable of randomizing HTTP header fields, which makes it infeasible to use them as signatures. Despite randomizing header fields, a significant proportion of AL-DDoS attacks still use a fixed sequence of URLs to launch attacks, which raises the possibility of using request sequences as dynamic signatures int the case of AL-DDoS attacks.

AL-DDoS attacks are generated by scripts which specify a list of URLs to be attacked in the target web application. All the zombies in the botnet then proceed to request these URLs repeatedly for a pre-determined duration. This means that AL-DDoS attacks consist of a set of URLs that keep repeating periodically (Bukac and Matyas 2015), which can be used as an attack signature. Some botnets such as Blackenergy only send repeated requests to a single URL (Nazario 2007), while more sophisticated botnets such as DirtJumper and Yoddos support multiple target URLs (Liao et al. 2015; Welzel et al. 2014). In both cases, the use of the repeating request pattern as an attack signature allows the system to detect malicious clients early.

6.1.1 Request Patterns as Dynamic Signatures

Consider a request sequence $abcdabcdabcdabcd...$ that is received by a web application. The request sequence is intercepted by our AL-DDoS detection mechanism, and the sequence is cumulatively assigned a suspicion score. Suppose that the suspicion score corresponding to the request sequence exceeded the threshold value θ after 10 iterations of $abcd$, i.e. after 40 requests. Our AL-DDoS detection mechanism identifies this connection as malicious and blocks it. The temporal similarity of AL-DDoS attacks dictates that there is a high probability that other attacking connections will also send a request sequence in the same manner, i.e. a sequence consisting of $abcd$ repeated multiple times. Since our detection mechanism assigns suspicion score values based only on the sequence of requests encountered, any subsequent connection would also be blocked after 10 iterations of $abcd$ or after 40 requests.

However, if we combine the knowledge that a sequence $abcdabcdabcdabcd...$ has previously been marked as malicious, we could potentially detect subsequent attacks faster. For this, we use the repeating pattern in the malicious request sequence as a dynamic signature. In this case, $abcd$ would serve as the signature. The rationale behind early detection is that whenever an attack signature is detected in incoming connections, the suspicion score is increased more than normal. This is accomplished by generating a multiplier value in addition to a suspicion score value. The value of the multiplier is 1 in normal cases, but when an attack signature is encountered, it is set to a value $m, m > 1$. This allows for a faster increase in suspicion score values for connections containing attack signatures. This allows connections containing attack signatures to be blocked faster than usual, leading to the early detection of subsequent attacks. It is worth noting that the use of an EDM only affects the detection latency. All other performance parameters such as detection rate, precision, FPR and FNR of the ADM remain the same.

In our architecture, an Early Detection Module (EDM) is responsible for extracting attack signatures from malicious request sequences, maintaining a dictionary of attack signatures efficiently and generating multiplier values. Our AL-DDoS detection module operates as usual, assigning suspicion scores to incoming connections. The only difference is that the generated suspicion score is multiplied by the multiplier value generated by the EDM, and the fact that newly identified malicious request sequences are forwarded to the EDM. Another interesting fact is that the EDM is not dependent on our AL-DDoS detection algorithm, so it can be used with any AL-DDoS detection mechanism. Thus, we refer to the AL-DDoS detection mechanism in a generalized sense and represent it as an Anomaly Detection Module (ADM). A high level overview of how an our Anomaly Detection Module (ADM) integrates with the Early Detection Module (EDM) is shown in Figure 6.1.

6.2 EARLY DDOS DETECTION USING REQUEST PATTERNS AS SIGNATURES

In this section, we present the detailed working of our Early DDoS detection module (EDM). The EDM is an add-on module that can be used along with any Anomaly based

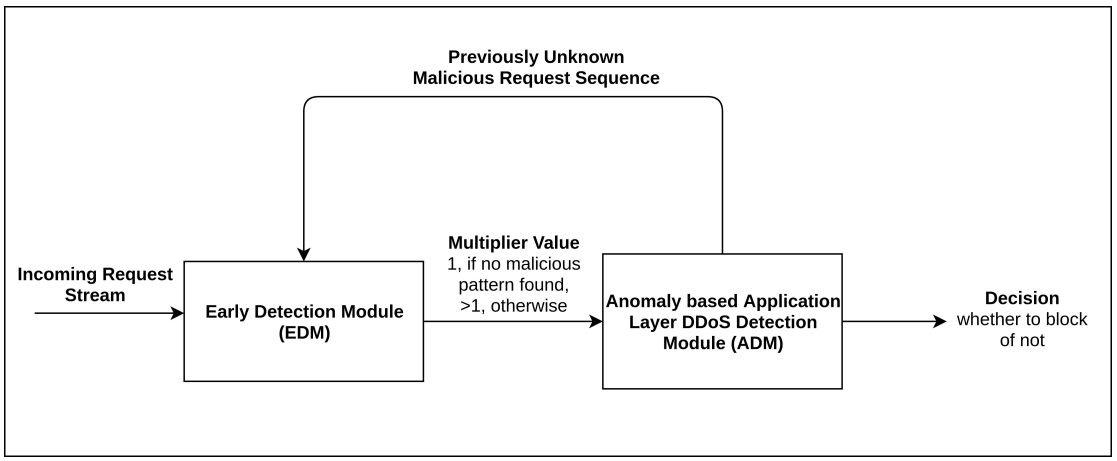


Figure 6.1: Integration of EDM with an existing ADM

Application Layer DDoS detection module (ADM) and enables early detection of attacks by using request patterns as dynamic signatures.

6.2.1 Architecture

The architecture of the EDM is depicted in Figure 6.2. A detailed description of the individual modules are presented below.

- String Minimization Module: As soon as a particular sequence of requests is identified as malicious, the EDM reduces it to its smallest repeating unit, which we call the request pattern. This is done using a modification of the Knuth-Morris-Pratt (KMP) algorithm. The request pattern is eventually stored in the Malicious Pattern Dictionary and the Bloom Filter Array.

- Malicious Pattern Dictionary (MPD): The Malicious Pattern Dictionary (MPD) stores the identified malicious request patterns. In order to ensure that the size of the MPD remains within bounds, only a hashed version of the request pattern is stored. The hash function used needs to have high collision resistance to avoid two patterns mapping to the same location, which could lead to false positives. In our implementation, we have chosen the FNV1-a hash function due to its collision resistance.

- Bloom Filter Array (BFA): Searching the MPD for every request pattern is computationally expensive with a complexity of $O(log\ n)$ per access where n is the

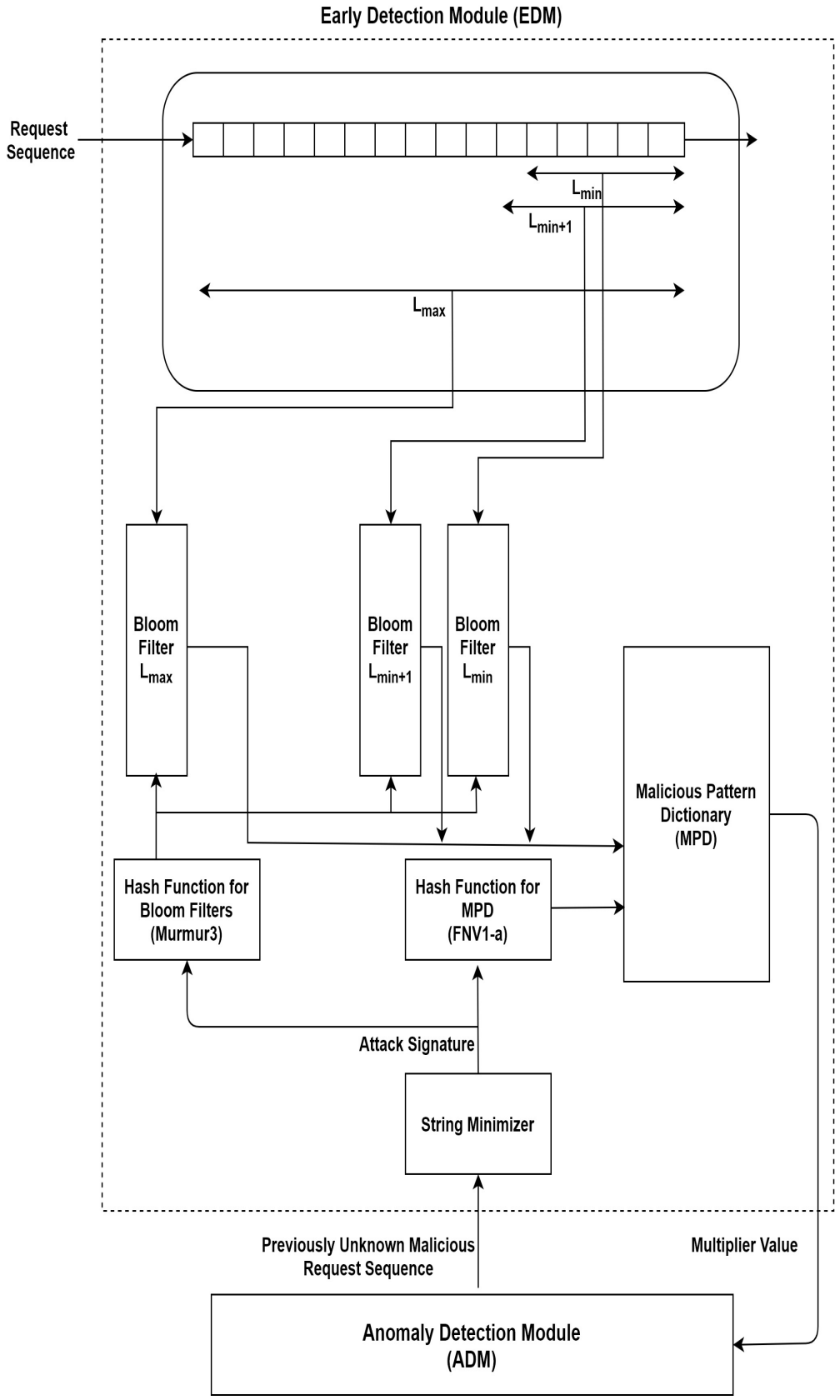


Figure 6.2: Architecture of the EDM

number of malicious request patterns in the MPD. This accumulates over time and creates a significant overhead. In order to avoid this overhead, the request patterns are also stored in a Bloom Filter Array (BFA). The use of a BFA allows to test for the non-inclusion of patterns in constant time. This is especially useful since majority of the request sequences are likely to be benign. The EDM maintains a separate bloom filter for different lengths of the malicious request patterns. Thus the maximum number of bloom filters used will be $L_{max} - L_{min}$ where L_{max} and L_{min} are the maximum and minimum lengths of the malicious request patterns encountered by the system. However, a BFA cannot confirm the existence of a pattern as malicious since it is prone to false positives. A second level of deterministic verification is required by checking in the MPD. Since every access is checked with the BFA, the hash function used in the Bloom Filters should be computationally inexpensive. In our implementation, we have used Murmur3 hash function for implementing the BFA.

6.2.2 Working

The EDM consists of two workflows that operate in parallel. The first workflow is concerned with adding malicious patterns to the MPD. This workflow is initiated by the ADM whenever it detects a previously unknown attack. The ADM forwards the malicious request sequence detected to the EDM. The string minimization module extracts the repeating pattern from the request sequence and stores it in the BFA and the malicious pattern dictionary. This ends the first workflow.

The second workflow operates continuously by inspecting the stream of requests for malicious patterns. A windowing mechanism is employed to extract patterns from the incoming request stream, and the extracted substring is checked for being malicious. The first stage of checking is performed using the BFA. If the BFA confirms that the substring is not malicious, then the examination is abandoned and the substring is deemed to be benign. However, if the BFA sends a positive response, the substring is checked for inclusion in the MPD using a search algorithm. If the substring is present in the MPD, then the substring is malicious. In this case, the EDM sends a value to

the ADM indicating that the request stream contains a malicious signature. The ADM can now choose to block the connection immediately, although this approach could lead to false positives. A better approach would be for the ADM to continue with anomaly detection while maintaining an anomaly score based on the value sent. The value of the anomaly score can be manipulated based on the input from the EDM to block malicious connections faster. This allows the EDM to function independently of the mechanism used in the ADM, making it independent of the firewall. Algorithm 7 describes the proposed detection mechanism with early detection incorporated. Algorithms 8 and 9 describe the functions $AddMaliciousPatter$ and $EarlyDetection$ used in Algorithm 7 respectively.

Table 6.1: Types of Attack Generated

Attack Code	Attack Type	Request Type	Request Workload	No. of URLs (Single/Multiple)
A1	Random Flood	Base	Any	Multiple
A2	Single URL flood	Base	Any	Single
A3	Random Asymmetric	Base	High	Multiple
A4	Single URL Asymmetric	Base	High	Single
A5	Botnet based Attack	Base	Any	Multiple
A6	Inline Flood	Inline	Low	Single
A7	Random Inline Flood	Inline	Low	Multiple
A8	Inline Botnet	Inline	Low	Multiple
A9	Combined Random Flood	Base + Inline	Any	Multiple
A10	Combined Bot Attack	Base + Inline	Any	Multiple

6.3 EXPERIMENTAL STUDY

A prototype of the proposed system was developed using the Go programming language and deployed on a Dell Optiplex machine with 16 GB RAM running Ubunut 16.04. In order to test the efficiency of the system we used the EDM in combination with our asymmetric AL-DDoS detection mechanism.

6.3.1 Experimental Setup

There are no attack generation tools which can generate asymmetric attacks on web applications. Due to this reason, we generated our own attack traces based on the traces available online. While generating attack vectors, we tried to include different varieties

Algorithm 7 Detection Algorithm with Early Detection

Input: Request R, PTA, Threshold θ
Output: Decision (blocked or allowed)

$updatetraces \leftarrow \phi$
$BFA \leftarrow \phi$
while $True$ **do**
  $Input\ new\ request\ R$
  $RequestHistory \leftarrow RequestHistory + R$
  $curr\_state \leftarrow ExtractState(R)$
  $w \leftarrow ExtractPriority(s)$
  $think\_time \leftarrow ExtractThinkTime(prev\_state, curr\_state)$
  if $(prev\_state, curr\_state) \in E$ **then**
    $t \leftarrow ExtractTransition(prev\_state, curr\_state)$
    $p_{trans} \leftarrow \rho(t)$
    $p_{think} = GetThinkTimeProbability(t, think\_time)$
  else
    $p_{trans} = 0.0$
    $p_{think} = 0.0$
  end if
  $multiplier \leftarrow EarlyDetection(RequestHistory)$
  $ss \leftarrow multiplier * ((1 - p_{trans} * p_{think}) * w)$
  $SS = SS + ss$
  if $SS \geq \theta$ **then**
    $The\ connection\ may\ be\ filtered$
    $AddMaliciousPattern(RequestHistory)$
    $updatetraces \leftarrow updatetraces - R$
  else
    $The\ connection\ can\ be\ forwarded\ to\ the\ server$
  end if
  if $SS \leq \theta_{update}$ **then**
    $updatetraces \leftarrow updatetraces \cup R$
  end if
  $prev\_state \leftarrow curr\_state$
  if $Update\ Condition\ is\ met$ **then**
    $Update(pta, updatetraces)$
  end if
end while

Algorithm 8 Function for Storing Detected Malicious Patterns

function $AddMaliciousPattern(RequestHistory)$
    $signature \leftarrow Reduce\_KMP(RequestHistory)$
    $size \leftarrow GetSize(signature)$
    if $BFA == \phi$ **then**
        $BFA.MaxSize \leftarrow size$
        $BFA.MinSize \leftarrow size$
    else if $BFA.MinSize > size$ **then**
        $BFA.MinSize \leftarrow size$
    else if $BFA.MaxSize < size$ **then**
        $BFA.MaxSize \leftarrow size$
    end if
    $BFA[size] \leftarrow BFA[size] \cup signature$
end function

Algorithm 9 Function for Checking the presence of Attack Signatures

Input: Request R, PTA, Threshold θ
Output: Decision (blocked or allowed)

function $EarlyDetection(RequestHistory)$
    for $BFA.MinSize \leq size \leq BFA.MaxSize$ **do**
        for $i \in [1, GetSize(RequestHistory) - size]$ **do**
            $pattern \leftarrow RequestHistory[i..i + size]$
            if $pattern \in BFA[size]$ **then return** $Multiplier[pattern]$
            end if
        end for
    end for
     return 1
end function

of attacks. A number of DDoS attacks target a single URL such as login pages. Such attacks are relatively easy to execute, but can be detected easily. The other strategy used by attackers is to use multiple URLs is a predefined sequence to evade detection. This attack requires knowledge about the structure of the web application, and hence requires more effort. However, the attacker is rewarded in the sense that such attacks are usually more difficult to detect. In addition to these two attack patterns, we introduce some more variations of attacks. Instead of treating all requests as the same, we made the following groupings:

- Base and Inline Requests: A web page is completely rendered only by the base HTML web page along with its constituent inline requests such as requests for image or CSS files. DDoS attacks can be carried out using either of the two types of requests, but attacks using base requests tend to be more effective (Ranjan et al. 2009).

- Request Workload: Every web request generates some amount of processing at the server side, or utilizes some server resources. Requests such as search queries require a larger amount of processing because they involve complicated joins and database searches. We call such queries high workload requests. High workload requests are used exclusively to launch asymmetric attacks.

In order to test the effectiveness of the proposed detection mechanism, a total of five different kinds of attacks were generated. The first attack (A1) generated is a random flood, which randomly sends base requests to a web server. The second attack (A2) generated is the Single URL flood, which is the most common flooding attack. Instead of randomly selecting from the available URLs, it sends repeated requests to a single base URL. This is one of the most common forms of DDoS attack in practice. Attacks A3 and A4 replicate these same attacks (A1 and A2) but with asymmetric attacks. Attack A3 is a random asymmetric attack and A4 is a single URL asymmetric attack. Attack A5 is another popular attack variation wherein the attacker follows a predefined script. This attack, which we call a Botnet based attack due to its use in botnets, creates an attack script by randomly selecting a set of URLs. Once the script is generated, then

the attacking connections repeatedly send requests to all the URLs in the script in the same order. Thus, the sequence of requests generated keeps repeating itself at regular intervals. The attacks discussed so far (A1-A5) specifically target base requests. Attacks A6-A9, on the other hand are specifically oriented at inline requests. Attack A6 is a single URL inline flood, while A7 generates a random inline flood. Attack A8 replicates a botnet based attack using only inline requests. Attack A9 uses both base and inline requests to launch a random flood attack, while attack A10 presents a sophisticated botnet based attack, which uses both base and inline requests. The details of the attacks generated are given in Table 6.1.

6.3.2 Effect of EDM on Detection Latency

Figures 6.3 and 6.4 illustrate how the average detection latency changes in the presence of the EDM. It can be clearly observed that in almost all cases, the use of an EDM causes a significant decrease in the average detection latency. The decrease in detection latency is directly related to the multiplier value generated by the EDM. In our experiments, we tested two schemes for generating the multiplier values.

- Static Multiplier EDM : Using the static multiplier scheme, the multiplier value generated by the EDM is a constant. In our experiments we have set the value to 5. In such a scheme, the value of the multiplier only represents whether a malicious pattern was identified in the incoming connection or not.

- Dynamic Multiplier EDM : A dynamic multiplier value tries to incorporate more global information, rather than just the presence or absence of a malicious pattern. In a dynamic multiplier scheme, every malicious pattern generates a distinct multiplier value based on the number of times it has been observed so far. This allows for the detection latency of attacks to be reduced considerably.

Figures 6.3 and 6.4 show that the decrease in detection latency is considerably more when a dynamic multiplier EDM is used. It is worth noting that the introduction of the EDM does not affect the detection rate and false positive rate of the detection mechanism in any way. It only enables the detection mechanism to detect AL-DDoS attacks

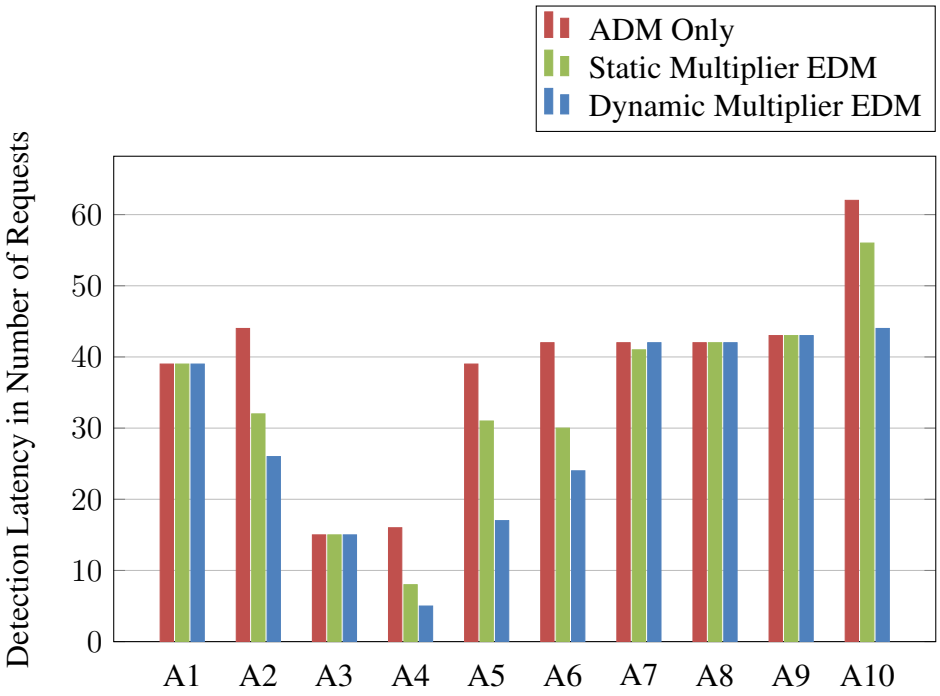


Figure 6.3: SDSC Detection Latency

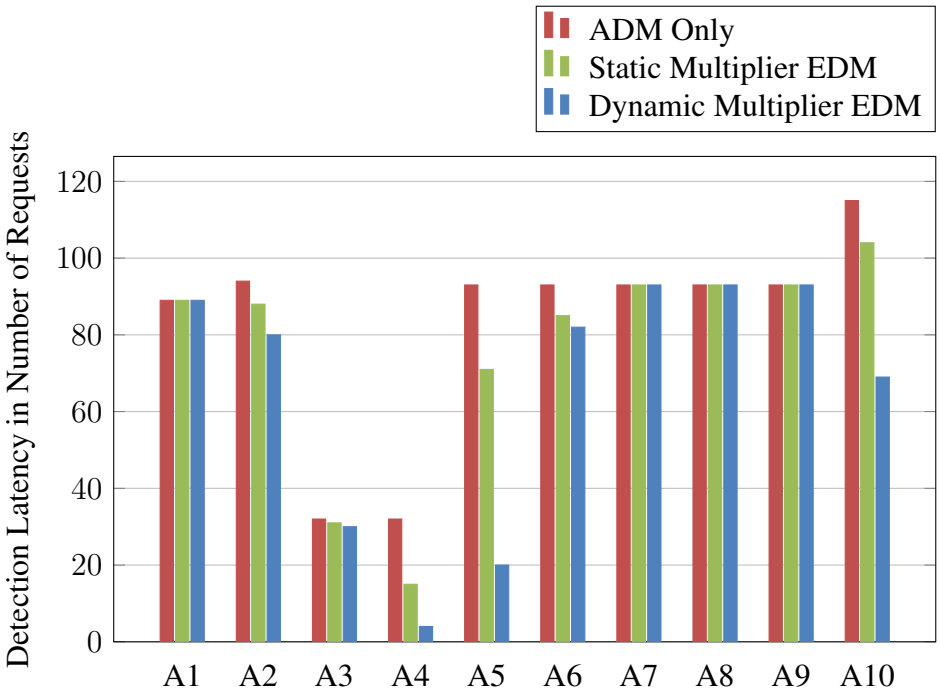


Figure 6.4: CLARKNET Detection Latency

early. It has to be noted that in the case of some attacks (A1,A3,A7,A8 and A9), the introduction of the EDM is unable to reduces the detection latency. This is due to the fact that these attacks are randomized attacks, and hence, there are no identifiable repeating patterns in these attack streams.

6.4 SUMMARY

The early detection of DDoS attacks has been a prominent research topic for years. Although there are a number of approaches proposed for the early detection of network layer DDoS attacks, these approaches cannot be extended easily to AL-DDoS attacks. In this chapter, we explain how the presence of repeating request patterns in AL-DDoS attacks can be used to detect these attacks faster. We provide a dynamic signature based approach using request patterns as signatures to enable the early detection of AL-DDoS attacks. Experiments on public datasets demonstrate that the use of an EDM can significantly reduce the detection latency for AL-DDoS attacks. In addition, the proposed approach is firewall agnostic, which means that it can be used in combination with any existing AL-DDoS detection approach or firewall to enable early detection.

CHAPTER 7

CONCLUSIONS AND FUTURE SCOPE

Detection of asymmetric AL-DDoS attacks against web applications is a daunting task. The use of legitimate requests, low attack volume, and similarity to legitimate user traffic make it extremely difficult to detect these attacks. A survey of existing literature related to asymmetric AL-DDoS attacks reveal that most of the existing detection mechanisms use indirect representations of user behaviour and complex modelling techniques, which leads to a larger false positive rate and longer detection times. They are also unable to adapt to changing user behaviour which further increases the false positive rate. These features indicate that existing detection mechanisms are not suited for real time use. In this work, a lightweight and incrementally updating model based on a Probabilistic Timed Automata (PTA) is used to model user behaviour, along with a fast detection mechanism for identifying anomalous clients. Experiments conducted on publicly available datasets confirm that the proposed approach is able to detect asymmetric AL-DDoS attacks faster compared to existing approaches, and is also comparatively lightweight, making it suitable for real time use.

The introduction of the HTTP/2 protocol adds additional complexity to the problem of asymmetric AL-DDoS detection. Since HTTP/2 retains most of the semantics of the HTTP/1.1 protocol, the proposed approach can easily be extended to cover attacks generated using the HTTP/2 protocol as well. However, our investigation into the newly introduced features of the HTTP/2 protocol reveal that attackers can launch potentially lethal AL-DDoS attacks against HTTP/2 servers by misusing the features. We pro-

pose an attack called Multiplexed Asymmetric Attack, which misuses the multiplexing feature in HTTP/2 to generate extremely dangerous AL-DDoS attacks, which can take down servers with relatively few attacking clients. In order to detect these attacks, our proposed approach was expanded to include HTTP/2 specific features. Experiments on public datasets reveal that our proposed approach can detect these attacks efficiently before they can cause any significant damage to the web server.

In addition to the detection of asymmetric AL-DDoS attacks, we develop an application-agnostic Early Detection Module (EDM) for AL-DDoS attacks using a dynamic signature based approach using HTTP request sequences. Experiments on public datasets reveal that the use of an EDM along with our proposed detection mechanism helps in reducing the detection latency of attacks considerably, thereby aiding in the early detection of these attacks.

7.1 FUTURE SCOPE

Research related to asymmetric AL-DDoS attacks is still in its infancy, and there are a lot of interesting research directions for further exploration:

- In our work, we have considered all search queries as high workload requests. Instead, a more sophisticated gradation of workload based on the search query could further increase the effectiveness of the detection mechanism.
- All anomaly detection techniques, including the proposed approach, rely on legitimate server logs in order to build a model of user behaviour. In some cases, such as when a web application has been newly launched, there is a lack of sufficient traces to build a model of user behaviour. An important research question is how a reliable model can be built in such a scenario.
- The security analysis of HTTP/2 protocol, especially the Server Push feature, is still lacking. In order to avoid exacerbating the egress flooding during an AL-DDoS attack, there is a need for more intelligent server push strategies.

To summarize, this dissertation explores the problem of efficient detection of asymmetric AL-DDoS attacks. We develop a lightweight and adaptable model for legit-

imate user behaviour based on a Probabilistic Timed Automata (PTA), along with a fast detection mechanism for identifying malicious clients. We also demonstrate that our approach can be extended to detect potentially lethal AL-DDoS attacks that can be launched by misusing the new features in HTTP/2. In addition, our proposed approach also uses a dynamic signature based approach using HTTP request sequences to enable the early detection of AL-DDoS attacks.

BIBLIOGRAPHY

Abbors, F., Ahmad, T., Truscan, D. and Porres, I. (2013). "Model-based performance testing of web services using probabilistic timed automata.." In *WEBIST*, 99–104.

Acunetix (2019). "Web application vulnerability report 2019." `https://cdn2.hubspot.net/hubfs/4595665/Acunetix_web_application_vulnerability_report_2019.pdf` (1 February, 2020).

Adi, E., Baig, Z., Lam, C. P. and Hingston, P. (2015). "Low-rate denial-of-service attacks against http/2 services." In *IT Convergence and Security (ICITCS), 2015 5th International Conference on*, IEEE, 1–5.

Aljuhani, A., Alharbi, T. and Taylor, B. (2019). "Mitigation of application layer ddos flood attack against web servers." *Journal of Information Security and Cybercrimes Research (JISCR)*, 2(1).

Andrews, A. A., Offutt, J. and Alexander, R. T. (2005). "Testing web applications by modeling with fsms." *Software & Systems Modeling*, 4(3), 326–345.

Apache (2011). "Apache httpd security advisory." `https://httpd.apache.org/security/CVE-2011-3192.txt` (1 February, 2020).

Beauquier, D. (2003). "On probabilistic timed automata." *Theoretical Computer Science*, 292(1), 65–84.

Beckett, D. and Sezer, S. (2017a). "Http/2 cannon: Experimental analysis on http/1 and http/2 request flood ddos attacks." In *Emerging Security Technologies (EST), 2017 Seventh International Conference on*, IEEE, 108–113.

Beckett, D. and Sezer, S. (2017b). "Http/2 tsunami: Investigating http/2 proxy amplification ddos attacks." In *Emerging Security Technologies (EST), 2017 Seventh International Conference on*, IEEE, 128–133.

Beckett, D., Sezer, S. and McCanny, J. (2017a). "New sensing technique for detecting application layer ddos attacks targeting back-end database resources." In *2017 IEEE International Conference on Communications (ICC)*, 1–7.

Beckett, D., Sezer, S. and McCanny, J. (2017b). "New sensing technique for detecting application layer ddos attacks targeting back-end database resources." In *2017 IEEE International Conference on Communications (ICC)*, 1–7.

Behal, S. and Kumar, K. (2017). "Characterization and comparison of ddos attack tools and traffic generators: A review.." *IJ Network Security*, 19(3), 383–393.

Behal, S., Kumar, K. and Sachdeva, M. (2018). "D-face: An anomaly based distributed approach for early detection of ddos attacks and flash events." *Journal of Network and Computer Applications*, 111, 49–63.

Beitollahi, H. and Deconinck, G. (2014). "Connectionscore: a statistical technique to resist application-layer ddos attacks." *Journal of Ambient Intelligence and Humanized Computing*, 5(3), 425–442.

Belshe, M., Peon, R. and Thomson, M. (2015). "Hypertext transfer protocol version 2 (http/2)." Technical report.

Boyd, S. and Keromytis, A. (2004). "Sqlrand: Preventing sql injection attacks." In *Applied Cryptography and Network Security*, Springer, 292–302.

Bravo, S. and Mauricio, D. (2018). "Ddos attack detection mechanism in the application layer using user features." In *2018 International Conference on Information and Computer Technologies (ICICT)*, 97–100.

Bukac, V. and Matyas, V. (2015). "Analyzing traffic features of common standalone dos attack tools." In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, Springer, 21–40.

Cambiaso, E., Papaleo, G. and Aiello, M. (2012). "Taxonomy of slow dos attacks to web applications." *Recent Trends in Computer Networks and Distributed Systems Security*, 195–204.

CCC (2011). "Effective denial of service attacks against web application platforms." `https://events.ccc.de/congress/2011/Fahrplan/events/4680.en.html` (1 February, 2020).

Chen, Y., Hwang, K. and Ku., W. (2007). "Collaborative detection of ddos attacks over multiple network domains." *IEEE Transactions on Parallel and Distributed Systems*, 18(12), 1649–1662.

Chwalinski, P., Belavkin, R. and Cheng, X. (2013a). "Detection of application layer ddos attack with clustering and likelihood analysis." In *Globecom Workshops (GC Wkshps), 2013 IEEE*, IEEE, 217–222.

Chwalinski, P., Belavkin, R. and Cheng, X. (2013b). "Detection of application layer ddos attacks with clustering and bayes factors." In *Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on*, IEEE, 156–161.

Cloudflare (2014). "Http flood attack." `https://www.cloudflare.com/learning/ddos/http-flood-ddos-attack/` (1 February, 2020).

Coindesk (2017). "Bitcoin gold website down following ddos attack." `https://www.coindesk.com/bitcoin-gold-website-following-massive-ddos-attack/` (1 February, 2020).

Corero (2016). "Ddos attacks plague olympic & brazilian government websites." `https://www.corero.com/blog/749-ddos-attacks-plague-olympic--brazilian-government-websites.html` (1 February, 2020).

Crosby, S. A. and Wallach, D. S. (2003). "Denial of service via algorithmic complexity attacks.." In *USENIX Security Symposium*, 29–44.

Dantas, Y. G., Nigam, V. and Fonseca, I. E. (2014). "A selective defense for application layer ddos attacks." In *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint*, IEEE, 75–82.

David, J. and Thomas, C. (2015). "Ddos attack detection using fast entropy approach on flow-based network traffic." *Procedia Computer Science*, 50, 30–36.

DDoS-Guard (2014). "Single request http flood (multiple verb single request)." `https://ddos-guard.net/en/terminology/single-request-http-flood-multiple-verb-single-request` (1 February, 2020).

de Saxc, H., Oprescu, I. and Chen, Y. (2015). "Is http/2 really faster than http/1.1?." In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 293–299.

Deepa, G., Thilagam, P. S., Khan, F. A., Praseed, A., Pais, A. R. and Palsetia, N. (2017). "Black-box detection of xquery injection and parameter tampering vulnerabilities in web applications." *International Journal of Information Security*, 1–16.

Deepa, G., Thilagam, P. S., Praseed, A. and Pais, A. R. (2018). "Detlogic: A black-box approach for detecting logic vulnerabilities in web applications." *Journal of Network and Computer Applications*, 109, 89–109.

Demoulin, H. M., Pedisich, I., Vasilakis, N., Liu, V., Loo, B. T. and Phan, L. T. X. (2019). "Detecting asymmetric application-layer denial-of-service attacks in-flight with finelame." In *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 693–708.

Devi, S. R. and Yogesh, P. (2012a). "Detection of application layer ddos attacks using information theory based metrics." *CS & IT-CSCP*, 10, 213–223.

Devi, S. R. and Yogesh, P. (2012b). "An effective approach to counter application layer ddos attacks." In *Computing Communication & Networking Technologies (ICCCNT), 2012 Third International Conference on*, IEEE, 1–4.

Dhanapal, A. and Nithyanandam, P. (2019). "The slow http distributed denial of service attack detection in cloud." *Scalable Computing: Practice and Experience*, 20(2), 285–298.

Dyn (2016). "Dyn analysis summary of friday october 21 attack." `https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack/` (1 February, 2020).

Emami-Taba, M., Amoui, M. and Tahvildari, L. (2015). "Strategy-aware mitigation using markov games for dynamic application-layer attacks." In *High Assurance Systems Engineering (HASE), 2015 IEEE 16th International Symposium on*, IEEE, 134–141.

Forbes (2014). "Bitcoin hit by massive ddos attack as tensions rise." `https://www.forbes.com/sites/leoking/2014/02/12/bitcoin-hit-by-massive-ddos-attack-as-tensions-rise/` (1 February, 2020).

Gandhi, R., Sharma, A., Mahoney, W., Sousan, W., Zhu, Q. and Laplante, P. (2011). "Dimensions of cyber-attacks: Cultural, social, economic, and political." *IEEE Technology and Society Magazine*, 30(1), 28–38.

Gao, H., Miao, H., Chen, S. and Mei, J. (2011). "Quantitative verification of navigation model for reliable web applications." In *2011 First ACIS/JNU International Conference on Computers, Networks, Systems and Industrial Engineering*, IEEE, 204–209.

Ghezzi, C., Pezzè, M., Sama, M. and Tamburrelli, G. (2014). "Mining behavior models from user-intensive web applications." In *Proceedings of the 36th International Conference on Software Engineering*, ACM, 277–287.

Giralte, L. C., Conde, C., De Diego, I. M. and Cabello, E. (2013). "Detecting denial of service by modelling web-server behaviour." *Computers & Electrical Engineering*, 39(7), 2252–2262.

Group, N. W. (1999). "Hypertext transfer protocol – http/1.1." `https://www.w3.org/Protocols/rfc2616/rfc2616.html` (27 August, 2019).

Guardian (2016a). "Hsbc suffers online banking cyber attack." https://www.theguardian.com/money/2016/jan/29/hsbc-online-banking-cyber-attack (1 February, 2020a).

Guardian (2016b). "Massive cyber-attack grinds liberia's internet to a halt." https://www.theguardian.com/technology/2016/nov/03/cyberattack-internet-liberia-ddos-hack-botnet (1 February, 2020b).

Guardian (2017). "Critical infrastructure not ready for ddos attacks: Foi data report." https://www.scmagazineuk.com/critical-infrastructure-not-ready-for-ddos-attacks-foi-data-report/article/684838/ (1 February, 2020).

Hashemian, R., Krishnamurthy, D. and Arlitt, M. (2012). "Web workload generation challenges–an empirical investigation." *Software: Practice and Experience*, 42(5), 629–647.

He, F., Baresi, L., Ghezzi, C. and Spoletini, P. (2007). "Formal analysis of publish-subscribe systems by probabilistic timed automata." In Derrick, J. and Vain, J., editors, *Formal Techniques for Networked and Distributed Systems – FORTE 2007*, Springer Berlin Heidelberg, Berlin, Heidelberg, 247–262.

Hu, X., Liu, C., Liu, S., You, W. and Zhao, Y. (2018). "Signalling security analysis: Is http/2 secure in 5g core network?." In *2018 10th International Conference on Wireless Communications and Signal Processing (WCSP)*, IEEE, 1–6.

Huang, C., Wang, J., Wu, G. and Chen, J. (2014). "Mining web user behaviors to detect application layer ddos attacks.." *JSW*, 9(4), 985–990.

Idhammad, M., Afdel, K. and Belouch, M. (2018). "Detection system of http ddos attacks in a cloud environment based on information theoretic entropy and random forest." *Security and Communication Networks*, 2018.

Imperva (2016). "Http/2: In-depth analysis of the top four flaws of the next generation web protocol." https://www.imperva.com/docs/Imperva_HII_HTTP2.pdf (10 January, 2019).

Incapsula (2015). "Analysis of vikingdom ddos attacks on u.s. government sites." https://www.incapsula.com/blog/vikingdom-ddos-attacks-us-government.html (1 February, 2020).

Incapsula (2016). "Ddos threat landscape report 2015-16." https://lp.incapsula.com/rs/804-TEY-921/images/2015-16%20DDoS%20Threat%20Landscape%20Report.pdf (1 February, 2020).

Incapsula (2017). "Global ddos threat landscape q1 2017." https://www.incapsula.com/ddos-report/ddos-report-q1-2017.html (1 February, 2020).

Johnson Singh, K., Thongam, K. and De, T. (2016). "Entropy-based application layer ddos attack detection using artificial neural networks." *Entropy*, 18(10), 350.

Kandula, S., Katabi, D., Jacob, M. and Berger, A. (2005). "Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds." In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, USENIX Association, 287–300.

Karami, M., Park, Y. and McCoy, D. (2016). "Stress testing the booters: Understanding and undermining the business of ddos services." In *Proceedings of the 25th International Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, 1033–1043.

Kaspersky (2016). "Kaspersky ddos intelligence report for q1 2016." https://securelist.com/kaspersky-ddos-intelligence-report-for-q1-2016/74550/ (1 February, 2020).

Katkar, V. and Bhirud, S. (2012). "Novel dos/ddos attack detection and signature generation." *International Journal of Computer Applications*, 47(10), 18–24.

Katkar, V., Zinjade, A., Dalvi, S., Bafna, T. and Mahajan, R. (2015). "Detection of dos/ddos attack against http servers using naive bayesian." In *2015 International Conference on Computing Communication Control and Automation*, 280–285.

Kaushal, K. and Sahni, V. (2016). "Early detection of ddos attack in wsn." *International Journal of Computer Applications*, 134(13), 0975–8887.

Laskar, S. and Mishra, D. (2016). "Qualified vector match and merge algorithm (qvmma) for ddos prevention and mitigation." *Procedia Computer Science*, 79, 41–52.

Lee, S., Kim, G. and Kim, S. (2011). "Sequence-order-independent network profiling for detecting application layer ddos attacks." *EURASIP Journal on Wireless Communications and Networking*, 2011(1), 50.

Li, B., Gao, M., Ma, L., Liang, Y. and Chen, G. (2019). "Web application-layer ddos attack detection based on generalized jaccard similarity and information entropy." In *International Conference on Artificial Intelligence and Security*, Springer, 576–585.

Liao, Q., Li, H., Kang, S. and Liu, C. (2015). "Application layer ddos attack detection using cluster with label based on sparse vector decomposition and rhythm matching." *Security and Communication Networks*, 8(17), 3111–3120.

Magazine, I. S. (2017). "Anonymous attacks spanish government sites." `https://www.infosecurity-magazine.com/news/anonymous-attacks-spanish/` (1 February, 2020).

McCombs, T. (2019). "Why turning on http/2 was a mistake." `https://www.lucidchart.com/techblog/2019/04/10/why-turning-on-http2-was-a-mistake/` (1 February, 2020).

Meng, B., Andi, W., Jian, X. and Fucai, Z. (2017). "Ddos attack detection system based on analysis of users' behaviors for application layer." In *Computational Science and Engineering (CSE) and Embedded and Ubiquitous Computing (EUC), 2017 IEEE International Conference on*, volume 1, IEEE, 596–599.

Meng, W., Qian, C., Hao, S., Borgolte, K., Vigna, G., Kruegel, C. and Lee, W. (2018). "Rampart: protecting web applications from cpu-exhaustion denial-of-service attacks." In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 393–410.

Mishra, A. K., Hellerstein, J. L., Cirne, W. and Das, C. R. (2010). "Towards characterizing cloud backend workloads: insights from google compute clusters." *ACM SIGMETRICS Performance Evaluation Review*, 37(4), 34–41.

Mongelli, M., Aiello, M., Cambiaso, E. and Papaleo, G. (2015). "Detection of dos attacks through fourier transform and mutual information." In *2015 IEEE International Conference on Communications (ICC)*, 7204–7209.

Mori, G. and Malik, J. (2003). "Recognizing objects in adversarial clutter: Breaking a visual captcha." In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, IEEE, I–I.

Moustis, D. and Kotzanikolaou, P. (2013). "Evaluating security controls against http-based ddos attacks." In *IISA 2013*, 1–6.

Nazario, J. (2007). "Blackenergy ddos bot analysis." *Arbor Networks*.

Nazario, J. (2009). "Politically motivated denial of service attacks." *The Virtual Battlefield: Perspectives on Cyber Warfare*, (s 165).

Ndibwile, J. D., Govardhan, A., Okada, K. and Kadobayashi, Y. (2015). "Web server protection against application layer ddos attacks using machine learning and traffic authentication." In *Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual*, volume 3, IEEE, 261–267.

Networks, A. (2012). "Leading us banks targeted in ddos attacks." `https://nakedsecurity.sophos.com/2012/09/27/banks-targeted-ddos-attacks/` (1 February, 2020).

Networks, A. (2016). "2016 ddos attack statistics." `https://www.arbornetworks.com/arbor-networks-releases-global-ddos-attack-data-for-1h-2016` (1 February, 2020).

Networks, F. (2019). "Detecting dos attacks dynamically." `https://techdocs.f5.com/kb/en-us/products/big-ip-afm/manuals/product/big-ip-system-dos-protection-and-protocol-firewall-implementations-14-0-0/07.html` (10 November, 2019).

Ni, T., Gu, X., Wang, H. and Li, Y. (2013). "Real-time detection of application-layer ddos attack using time series analysis." *Journal of Control Science and Engineering*, 2013, 4.

O'Gorman, G. and McDonald, G. (2012). *Ransomware: A growing menace*, Symantec Corporation.

Oikonomou, G. and Mirkovic, J. (2009). "Modeling human behavior for defense against flash-crowd attacks." In *Communications, 2009. ICC'09. IEEE International Conference on*, IEEE, 1–6.

Oo, K. K., Ye, K. Z., Tun, H., Lin, K. Z. and Portnov, E. (2016). "Enhancement of preventing application layer based on ddos attacks by using hidden semi-markov model." In *Genetic and Evolutionary Computing*, Springer, 125–135.

Oshima, S., Nakashima, T. and Sueyoshi, T. (2010). "Early dos/ddos detection method using short-term statistics." In *Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on*, IEEE, 168–173.

Park, J., Iwai, K., Tanaka, H. and Kurokawa, T. (2014). "Analysis of slow read dos attack." In *Information Theory and its Applications (ISITA), 2014 International Symposium on*, IEEE, 60–64.

Patni, P., Iyer, K., Sarode, R., Mali, A. and Nimkar, A. (2017). "Man-in-the-middle attack in http/2." In *Intelligent Computing and Control (I2C2), 2017 International Conference on*, IEEE, 1–6.

Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A. and Knightly, E. (2009). "Ddos-shield: Ddos-resilient scheduling to counter application layer attacks." *IEEE/ACM on Networking (TON)*, 17(1), 26–39.

Register (2012). "Anonymous turns its ddos cannons on india." https://www.theregister.co.uk/2012/05/18/anonymous_ddos_india_sites/ (1 February, 2020).

Register (2018). "Gits club github code tub with record-breaking 1.35tbps ddos drub." https://www.theregister.co.uk/2018/03/01/github_ddos_biggest_ever/ (1 February, 2020).

Santanna, J. J., Schmidt, R. d. O., Tuncer, D., de Vries, J., Granville, L. Z. and Pras, A. (2016). "Booter blacklist: Unveiling ddos-for-hire websites." In *2016 12th International Conference on Network and Service Management (CNSM)*, IEEE, 144–152.

Scaife, N., Carter, H., Traynor, P. and Butler, K. R. (2016). "Cryptolock (and drop it): stopping ransomware attacks on user data." In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, IEEE, 303–312.

Schneider, L. F., Krajina, A. and Krivobokova, T. (2019). "Threshold selection in univariate extreme value analysis." *arXiv preprint arXiv:1903.02517*.

SCMagazine (2017). "Ddos attacks delay trains, stymie transportation services in sweden." https://www.scmagazine.com/ddos-attacks-delay-trains-stymie-transportation-services-in-sweden/article/700227/ (1 February, 2020).

Shtern, M., Sandel, R., Litoiu, M., Bachalo, C. and Theodorou, V. (2014). "Towards mitigation of low and slow application ddos attacks." In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, IEEE, 604–609.

Silicon (2017). "Irish government websites taken down by ddos attacks." http://www.silicon.co.uk/e-regulation/irish-government-websites-ddos-18442 (1 February, 2020).

Singh, K., Singh, P. and Kumar, K. (2018). "User behavior analytics-based classification of application layer http-get flood attacks." *Journal of Network and Computer Applications*, 112, 97 – 114.

Singh, K. J. and De, T. (2017). "Mlp-ga based algorithm to detect application layer ddos attack." *Journal of Information Security and Applications*, 36, 145–153.

Sivabalan, S. and Radcliffe, P. (2013). "A novel framework to detect and block ddos attack at the application layer." In *TENCON Spring Conference, 2013 IEEE*, IEEE, 578–582.

Sivakorn, S., Polakis, J. and Keromytis, A. D. (2016). "Im not a human: Breaking the google recaptcha." *Black Hat,(i)*, 1–12.

Sonicwall (2019 (accessed August 13 2019)). "2019 sonicwall cyber threat report." `https://d3ik27cqx8s5ub.cloudfront.net/media/uploads/2019/03/2019-SonicWall-Cyber-Threat-Report.pdf` (1 February, 2020).

Statista (2019). "eservices worldwide." `https://www.statista.com/outlook/261/100/eservices/worldwide` (27 August, 2019).

Statistics, H. (2020). "Usage of http/2 for websites." `https://w3techs.com/technologies/details/ce-http2/all/all` (1 February, 2020).

Stevanovic, D. and Vlajic, N. (2014). "Next generation application-layer ddos defences: Applying the concepts of outlier detection in data streams with concept drift." In *2014 13th International Conference on Machine Learning and Applications*, 456–462.

Suresh, M., Amritha, P., Mohan, A. K. and Kumar, V. A. (2018). "An investigation on http/2 security." *Journal of Cyber Security and Mobility*, 7(1), 161–189.

Thirumaran, M., Dhavachelvan, P., Abarna, S. and Lakshmi, P. (2011). "Finite state machine based evaluation model for web service reliability analysis." *International Journal of Web & Semantic Technology*, 2(4), 125.

Times, I. (2016). "Hackers leave finnish residents cold after ddos attack knocks out heating systems." `http://www.ibtimes.co.uk/hackers-leave-finnish-residents-cold-after-ddos-attack-knocks-out-heating-systems-1590639` (1 February, 2020).

Tripathi, N. and Hubballi, N. (2018). "Slow rate denial of service attacks against http/2 and detection." *Computers & security*, 72, 255–272.

Tripathi, N., Hubballi, N. and Singh, Y. (2016). "How secure are web servers? an empirical study of slow http dos attacks and detection." In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, 454–463.

Union, I. T. (2018). "Statistics." `https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx` (1 February, 2020).

Van Hoorn, A., Rohr, M. and Hasselbring, W. (2008). "Generating probabilistic and intensity-varying workload for web-based software systems." In *SPEC International Performance Evaluation Workshop*, Springer, 124–143.

Wang, C., Miu, T. T. N., Luo, X. and Wang, J. (2018). "Skyshield: A sketch-based defense system against application layer ddos attacks." *IEEE Transactions on Information Forensics and Security*, 13(3), 559–573.

Wang, J., Yang, X. and Long, K. (2011). "Web ddos detection schemes based on measuring user's access behavior with large deviation." In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, IEEE, 1–5.

Welzel, A., Rossow, C. and Bos, H. (2014). "On measuring the impact of ddos botnets." In *Proceedings of the Seventh European Workshop on System Security*, ACM, 3.

Wen, S., Jia, W., Zhou, W., Zhou, W. and Xu, C. (2010). "Cald: Surviving various application-layer ddos attacks that mimic flash crowd." In *network and system security (nss), 2010 4th international conference on*, IEEE, 247–254.

Xiang, Y., Li, K. and Zhou, W. (2011). "Low-rate ddos attacks detection and traceback by using new information metrics." *IEEE Transactions on Information Forensics and Security*, 6(2), 426–437.

Xie, Y. and Yu, S.-Z. (2009a). "A large-scale hidden semi-markov model for anomaly detection on user browsing behaviors." *IEEE/ACM Transactions on Networking (TON)*, 17(1), 54–65.

Xie, Y. and Yu, S.-Z. (2009b). "Monitoring the application-layer ddos attacks for popular websites." *IEEE/ACM Transactions on Networking (TON)*, 17(1), 15–25.

Xu, C., Zhao, G., Xie, G. and Yu, S. (2014). "Detection on application layer ddos using random walk model." In *Communications (ICC), 2014 IEEE International Conference on*, IEEE, 707–712.

Yadav, S. and Selvakumar, S. (2015). "Detection of application layer ddos attack by modeling user behavior using logistic regression." In *Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions), 2015 4th International Conference on*, IEEE, 1–6.

Yadav, S. and Subramanian, S. (2016). "Detection of application layer ddos attack by feature learning using stacked autoencoder." In *Computational Techniques in Information and Communication Technologies (ICCTICT), 2016 International Conference on*, IEEE, 361–366.

Yan, J. and El Ahmad, A. S. (2007). "Breaking visual captchas with naive pattern recognition algorithms." In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, IEEE, 279–291.

Ye, C., Zheng, K. and She, C. (2012). "Application layer ddos detection using clustering analysis." In *Computer Science and Network Technology (ICCSNT), 2012 2nd International Conference on*, IEEE, 1038–1041.

Yu, S. and Zhou, W. (2008). "Entropy-based collaborative detection of ddos attacks on community networks." In *2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 566–571.

Yu, S., Zhou, W. and Doss, R. (2008). "Information theory based detection against network behavior mimicking ddos attacks." *IEEE Communications Letters*, 12(4), 318–321.

Yu, S., Zhou, W., Jia, W., Guo, S., Xiang, Y. and Tang, F. (2011). "Discriminating ddos attacks from flash crowds using flow correlation coefficient." *IEEE Transactions on Parallel and Distributed Systems*, 23(6), 1073–1080.

Yu Chen and Kai Hwang (2006). "Collaborative change detection of ddos attacks on community and isp networks." In *International Symposium on Collaborative Technologies and Systems (CTS'06)*, 401–410.

Zargar, S. T., Joshi, J. and Tipper, D. (2013). "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks." *IEEE communications surveys & tutorials*, 15(4), 2046–2069.

Zhang, H., Taha, A., Trapero, R., Luna, J. and Suri, N. (2016). "Sentry: A novel approach for mitigating application layer ddos threats." In *Trustcom/BigDataSE/I SPA, 2016 IEEE*, IEEE, 465–472.

Zhao, Y., Zhang, W., Feng, Y. and Yu, B. (2018). "A classification detection algorithm based on joint entropy vector against application-layer ddos attack." *Security and Communication Networks*, 2018.

Zhou, W., Jia, W., Wen, S., Xiang, Y. and Zhou, W. (2014). "Detection and defense of application-layer ddos attacks in backbone web traffic." *Future Generation Computer Systems*, 38, 36–46.

Zuzak, I., Budiselic, I. and Delac, G. (2011a). "A finite-state machine approach for modeling and analyzing restful systems." *Journal of Web Engineering*, 10(4), 353.

Zuzak, I., Budiselic, I. and Delac, G. (2011b). "Formal modeling of restful systems using finite-state machines." In *International Conference on Web Engineering*, Springer, 346–360.

RESEARCH OUTCOMES

PATENTS FILED

1. Title: System and Method for Detecting Asymmetric Application Layer DDoS Attacks using User Access Pattern Model
   Inventors: P. Santhi Thilagam, Amit Praseed
   Applicant: National Institute of Technology Karnataka, Surathkal
   Patent Application No.: 201941040132
   Filing Date: 03/10/2019

PUBLICATIONS

1. Praseed, A. & Thilagam, P. S. (2018). DDoS attacks at the application layer: Challenges and research perspectives for safeguarding Web applications. IEEE Communications Surveys & Tutorials, 21(1), 661-685. (DOI: `https://doi.org/10.1109/COMST.2018.2870658`, URL: `https://ieeexplore.ieee.org/document/8466561`) (Impact Factor: 23.7)

2. Praseed, A. & Thilagam, P. S. (2019). Multiplexed Asymmetric Attacks: Next-Generation DDoS on HTTP/2 Servers. IEEE Transactions on Information Forensics and Security. (DOI: `https://doi.org/10.1109/TIFS.2019.2950121`, URL: `https://ieeexplore.ieee.org/document/8886426`) (Impact Factor: 6.013)

3. Praseed, A. & Thilagam, P. S. (2020). Modelling Behavioural Dynamics for Asymmetric Application Layer DDoS Detection. IEEE Transactions on Information Forensics and Security (Accepted for publication on 7 August 2020) (Impact Factor: 6.013)

4. Praseed, A. & Thilagam, P. S. Fuzzy Request Set Modelling for Detecting Multiplexed Asymmetric DDoS Attacks on HTTP/2 Servers, *IEEE Transactions on Dependable and Secure Computing* (Paper submitted on May 8, 2020 and status is "Under Review").

5. Praseed, A. & Thilagam, P. S. HTTP Request Pattern based Signatures for Early Application Layer DDoS Detection: A Firewall Agnostic Approach, *Elsevier Computers and Security* (Paper submitted on 4 July 2020 and status is "Under Review").

BIODATA

Name:	Amit Praseed
Date of Birth:	5th January, 1993
Gender:	Male
Marital Status:	Single
Father's Name:	Praseed KM
Mother's Name:	Sangeetha K
Address:	"Sangeeth", Kadamberi Road, Bakkalam, PO Kanul, Kannur, Kerala PIN-670562.
E-mail:	amitpraseed@gmail.com
Mobile:	8904611847
Qualification:	B.Tech in Computer Science and Engineering (College of Engineering, Thiruvanathapuram)
	M.Tech in Computer Science and Engineering-Information Security (National Institute of Technology Karnataka, Surathkal)
Areas of Interest:	Web Security, Information Security