# Novel Application Layer Denial of Service Attacks and Detection

Submitted in partial fulfillment of the requirements

for the degree of

*DOCTOR OF PHILOSOPHY*

by

**Nikhil Tripathi**

**1401101002**

Supervisor:

Dr. Neminath Hubballi

DISCIPLINE OF COMPUTER SCIENCE AND

ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

INDORE - 453 552

March 2018

# Novel Application Layer Denial of Service Attacks and Detection

By

Nikhil Tripathi

A Thesis Submitted to

Indian Institute of Technology Indore

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Approved:

---

Dr. Neminath. Hubballi

Thesis Advisor

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

INDORE - 453 552

March 2018

# ACKNOWLEDGEMENTS

I would like to take this opportunity to express my heartfelt gratitude to a number of persons who in one or the other way contributed by making this time as learnable, enjoyable, and bearable. At first, I would like to thank my supervisor **Dr. Neminath Hubballi**, who was a constant source of inspiration during my work. Without his constant guidance and research directions, this research work could not be completed. His continuous support and encouragement has motivated me to remain streamlined in my research work. I am also grateful to **Dr. Surya Prakash**, HOD of Computer Science for all his extended help and support.

I am thankful to **Dr. Gourinath Banda** and **Dr. Trapti Jain**, my research progress committee members for taking out some valuable time to evaluate my progress all these years. Their valuable comments and suggestions helped me to improve my work at various stages.

My sincere acknowledgement and respect to **Prof. Pradeep Mathur**, Director, Indian Institute of Technology Indore for providing me the opportunity to explore my research capabilities at Indian Institute of Technology Indore.

I would like to appreciate the company of my dearest colleagues and friends especially, Mrs. Avani Sharma, Dr. Saumya Bhadauria, Mr. Piyush Joshi, Mr. Syed Sadaf Ali, Mr. Navneet Pratap Singh, Mr. Mayank Modak, Mr. Rakesh Sharma, Mr. Rajendra Choudhary, Mr. Chandan Gautam, Mr. Aditya Shastri, Mr. Ram Sharma and Mr. Aaditya Chouhan. Hearty thanks to my labmates Mr. Mayank Swarnkar, Ms. Anuja Tayal and Ms. Pratibha Khandait for their cooperation.

I would like to express my heartfelt respect to my parents for their love, care and support they have provided to me throughout my life. Special thanks to my sister and brother-in-law for their support and encouragements.

Finally, I am thankful to all who directly or indirectly contributed, helped and supported me. To sign off, I write a quote by James Joyce -

"Mistakes are the portals of discovery."

***Nikhil Tripathi***

*To my family and friends*

# ABSTRACT

Application layer Denial of Service (DoS) attacks exploit either flaws in protocol specification or implementation errors to render the application unusable for other legitimate users. These attacks can be generated with minimal number of specially crafted requests. Thus it is important to scrutinize application layer protocols for potential specification and implementation flaws. In this thesis, we study four application layer protocols namely Dynamic Host Configuration Protocol (DHCP), Hypertext Transfer Protocol (HTTP/1.1), its successor HTTP/2 and Network Time Protocol (NTP) for potential flaws. We describe new attacks against these protocols and also various detection methods to detect such attacks.

Dynamic Host Configuration Protocol (DHCP) is used for automatic configuration of IP address. This protocol is vulnerable to an attack known as starvation attack, which prevents clients from acquiring IP address. In our first contribution, we highlight some of the practical difficulties in generating this attack in wireless networks. Subsequently, we propose a new starvation attack (with three variants) which we name as "*Induced DHCP Starvation Attack*". This attack can be launched both in wired and wireless networks. We test various state-of-the-art security features available with modern network switches to counter starvation attack, and show that these security features can not mitigate this new attack. We also propose few anomaly detection methods to detect this attack.

Hypertext Transfer Protocol (HTTP) is a predominantly popular web communication protocol. We perform an empirical evaluation of four popular web servers against known Slow HTTP DoS attacks of HTTP/1.1 to conclude that majority of them are vulnerable. We extend this study with testing of 100 sample websites chosen from four different categories and furnish the vulnerability assessment report. Subsequently, we propose an anomaly detection system to detect these attacks.

We propose five new Slow Rate DoS attacks against HTTP/2 protocol. These attacks require sending specially crafted requests to a HTTP/2 server. We test the proposed attacks against different web servers and show that all of them are vulnerable

to at least one attack. We also propose an anomaly detection system which uses chi-square test using a set of feature statistics derived from HTTP/2 traffic to detect these attacks.

Clock synchronization among computers is important as inconsistencies in time can affect various core Internet services. Network Time Protocol (NTP) is used to synchronize clocks among computers. We propose an attack against NTPs broadcast mode that can prevent a NTP client configured in authenticated/unauthenticated broadcast or multicast mode from synchronizing its clock with the servers clock. This attack requires sending spoofed NTP packets from an adversary, hence we propose a method to detect spoofed packets in order to detect this new attack. Spoofed NTP packets are identified by noticing inconsistencies in the Time-To-Live values of received NTP packets.

# LIST OF PUBLICATIONS

## Published/Accepted

### International Journals

**J1.** **Nikhil Tripathi**, Neminath Hubballi. *Slow Rate Denial of Service Attacks against HTTP/2 and Detection*, Computers & Security, Volume 72, 2018, Pages 255-272 (Elsevier). (IF: 2.849)

**J2.** **Nikhil Tripathi**, Neminath Hubballi. *Detecting Stealth DHCP Starvation Attack using Machine Learning Approach*, Journal of Computer Virology and Hacking Techniques, 14(3), 2017, Pages 233-244 (Springer). (SNIP: 0.368)

**J3.** Neminath Hubballi, **Nikhil Tripathi**. *An Event Based Technique for Detecting Spoofed IP Packets*, Journal of Information Security and Applications, Volume 35, 2017, Pages 32-43 (Elsevier). (SNIP: 1.186)

**J4.** Neminath Hubballi, **Nikhil Tripathi**. *A Closer Look into DHCP Starvation Attack in Wireless Networks*, Computers & Security, Volume 65, 2017, Pages 387-404 (Elsevier). (IF: 2.849)

### International Conferences

**C1.** **Nikhil Tripathi**, Mayank Swarnkar, Neminath Hubballi. *DNS Spoofing in Local Networks Made Easy.* 2017 International Conference on Advanced Networks and Telecommunications Systems (ANTS), Bhubaneshwar, Pages 1-6 (IEEE).

**C2.** **Nikhil Tripathi**, Neminath Hubballi. *A Probabilistic Anomaly Detection Scheme to Detect DHCP Starvation Attacks.* 2016 International Conference on Advanced Networks and Telecommunications Systems (ANTS), Bangalore, 2016, Pages 1-6 (IEEE).

**C3.** **Nikhil Tripathi**, Neminath Hubballi, Yogendra Singh. *How Secure are Web Servers? An Empirical Study of Slow HTTP DoS Attacks and Detection.* 2016 International Conference on Availability, Reliability and Security (ARES), Salzburg, 2016, Pages 454-463 (IEEE).

**C4.** **Nikhil Tripathi**, Neminath Hubballi. *Exploiting DHCP Server-side IP Address Conflict Detection: A DHCP Starvation Attack.* 2015 International Conference on Advanced Networks and Telecommuncations Systems (ANTS), Kolkata, 2015, Pages 1-3 (IEEE).

## Under Preparation

**J5.** **Nikhil Tripathi**, Neminath Hubballi. *Preventing Time Synchronization in NTP's Broadcast Mode.* (Under Preparation).

**J6.** **Nikhil Tripathi**, Neminath Hubballi. *A Survey of Application Layer Denial of Service (DoS) Attacks and Defense Mechanisms.* (Under Preparation).

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **AD** | Active Directory |
| **AP** | Access Point |
| **ARP** | Address Resolution Protocol |
| **AS** | Autonomous System |
| **ATS** | Apache Traffic Server |
| **CUI** | Console User Interface |
| **CVE** | Common Vulnerabilities and Exposures |
| **DAI** | Dynamic ARP Inspection |
| **DDoS** | Distributed Denial of Service |
| **DES** | Discrete Event System |
| **DHCP** | Dynamic Host Configuration Protocol |
| **DHCPACK** | DHCP Acknowledgement Message |
| **DHCPNACK** | DHCP Negative Acknowledgement Message |
| **DNS** | Domain Name System |
| **DORA** | Discover Offer Request and Acknowledgment |
| **DoS** | Denial of Service |
| **FPR** | False Positive Rate |
| **FTP** | File Transfer Protocol |
| **GTK** | Group Temporal Key |
| **HD** | Hellinger Distance |
| **HTML** | Hyper Text Markup Language |

| | |
|---|---|
| **HTTP** | Hyper Text Transfer Protocol |
| **ICMP** | Internet Control Message Protocol |
| **IDS** | Intrusion Detection System |
| **IIS** | Internet Information Services |
| **IP** | Internet Protocol |
| **IPv4** | Internet Protocol version 4 |
| **IPv6** | Internet Protocol version 6 |
| **ISP** | Internet Service Provider |
| **KNN** | K-Nearest Neighbour |
| **KoD** | Kiss of Death |
| **LoRDAS** | Low Rate DoS Attacks |
| **LTS** | Long Term Support |
| **MAC** | Media Access Control |
| **MD** | Message Digest |
| **MIC** | Message Integrity Code |
| **MiTM** | Man-in-The-Middle |
| **NAT** | Network Address Translation |
| **NTP** | Network Time Protocol |
| **OF** | OpenFlow |
| **PCA** | Principal Component Analysis |
| **PMK** | Pairwise Master Key |
| **PTK** | Pairwise Transient Key |
| **RAM** | Random Access Memory |
| **RFC** | Request For Comment |
| **RoQ** | Reduction of Quality |
| **RPKI** | Resource Public Key Infrastructure |
| **RPS** | Request Per Second |
| **RTO** | Retransmission Time Out |
| **SIP** | Session Initiation Protocol |

| | |
|---|---|
| **SMTP** | Simple Mail Transfer Protocol |
| **SNMP** | Simple Network Management Protocol |
| **SOM** | Self Organizing Maps |
| **SPM** | Spoofing Prevention Method |
| **SSL** | Secure Socket Layer |
| **SVM** | Support Vector Machine |
| **TCP** | Transmission Control Protocol |
| **TLS** | Transport Layer Security |
| **TTL** | Time-To-Live |
| **UDP** | User Datagram Protocol |
| **URI** | Uniform Resource Indicator |
| **URL** | Uniform Resource Locator |
| **USNO** | U.S. Naval Observatory |
| **VoIP** | Voice over Internet Protocol |
| **WAF** | Web Application Firewall |
| **WAN** | Wide Area Network |
| **WPA** | Wireless Protection Access |
| **WUM** | Web Usage Mining |

# Chapter 1

# Introduction

*Confidentiality*, *Integrity* and *Availability* are the three major components of cyber security [4]. Confidentiality ensures that any sensitive information is prevented from reaching the unauthorized users while making sure that the authorized people can access it as and when required. Integrity maintains the consistency, accuracy and trustworthiness of data over its entire life cycle and also ensures that the data can not be altered by unauthorized users. Availability refers to ensuring that the authorized parties are able to access the information when needed. Hackers in various underground communities aim to compromise either of these security components to target their victims. Nevertheless, compromising confidentiality and integrity is becoming difficult due to fortunate reasons such as deployment of robust cryptographic solutions in networking infrastructure. However, hackers always find it easier to target availability component as they generally do not require a privileged access or position to compromise it. Moreover, a plethora of tools and techniques are available in the wild that can be used to compromise the availability of target's resources and services. An adversary compromises the availability component using what is commonly known as Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks.

The rest of the chapter is organized as follows. In Section 1.1, we describe different types of DoS and DDoS attacks and some recent incidents where these attacks were encountered. In Section 1.2, we discuss the motivation behind the research work presented in this thesis. We give a summary of thesis contributions in Section 1.3. In

Section 1.4, we give outline of rest of the thesis.

## 1.1 DoS/DDoS Attacks

Denial of Service (DoS) and its variant, Distributed Denial of Service (DDoS) attacks have become a serious concern for network administrators since last two decades. These attacks intend to exhaust the resources (such as memory, CPU and network bandwidth) to make it unavailable for the legitimate users. The popularity of DoS attacks has been continuously rising among various hacking groups due to several reasons ranging from simply getting recognition in the underground communities to the incentives given to individuals hired by an organization to launch DoS/DDoS attacks against its potential competitors in the market. Different types of DoS/DDoS attacks [5] have evolved over time. There have been several incidents of DoS and DDoS attacks encountered in the past. In some of the incidents, these attacks were launched against only a specific organization while in other cases, very large scale attacks were witnessed affecting major portions of the Internet. With the development of easy-to-use tools, a variety of DoS and DDoS attacks has been encountered in these incidents. Some of the major DoS/DDoS techniques are described in the next few subsections.

### 1.1.1 Flooding-based DDoS Attacks

In flooding-based DDoS attacks [6], an adversary first discovers a large number of vulnerable systems on the internet and then compromises them by exploiting vulnerabilities and security holes present in those systems. Once the adversary gets the control of those systems, it uses them as bots/zombies and commands them to send huge flood of TCP SYN, ICMP or UDP packets to the victim. Figure 1.1 shows an example of flooding-based DDoS attacks. These attacks disturb victim's connectivity by exhausting its network bandwidth. In late 90s, flooding-based DDoS attacks were prominent form of attacks.

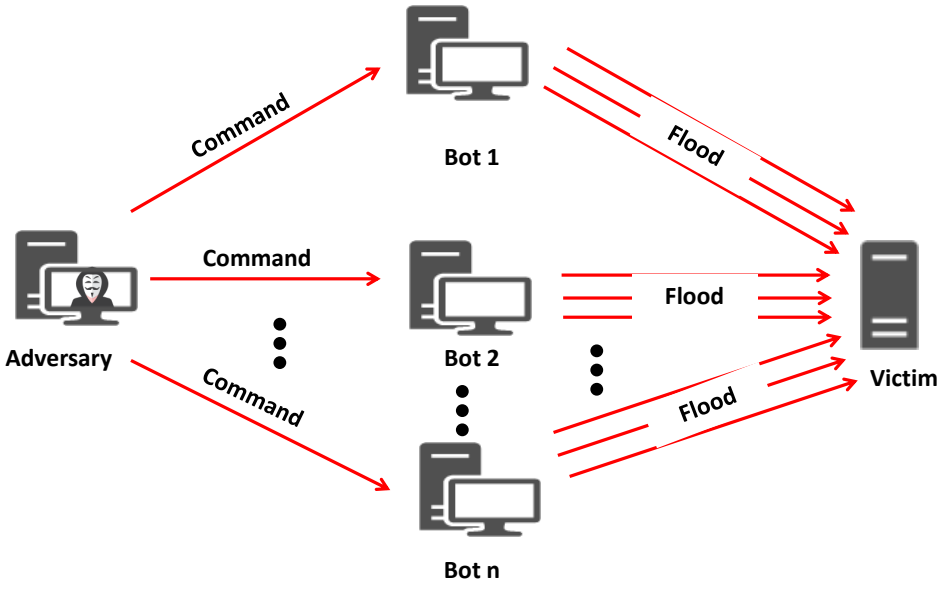



Figure 1.1: Flooding-based DDoS attack

### 1.1.2 Reflection- and Amplification-based DDoS Attacks

After flooding-based DDoS attacks, more advanced attacks such as *Smurf* attack were evolved that use reflection and amplification techniques [7, 8]. Reflection requires sending forged requests with victim's IP address as source to the reflectors (services using UDP as the transport layer protocol); hence, these reflectors send their replies to the victim and exhaust victim's resources. Figure 1.2 shows an example of reflection-based DDoS attacks. Amplification techniques, on the other hand, exploit services of protocols such as Domain Name System (DNS) [9] and Network Time Protocol (NTP) [10] to generate either large response or multiple responses for each request sent to these services. Usually, both reflection and amplification attacks are launched together to maximize the effect. Figure 1.3 shows an example of amplification-based DDoS attacks. Similar to flooding-based DDoS attacks, these types of DDoS attacks also target victim's network bandwidth.

With the advancement in networking technologies, today, it has become challenging for these attacks to disturb the victim's connectivity as these attacks require sending millions of packets continuously to maintain DoS at the victim side. Moreover, due to

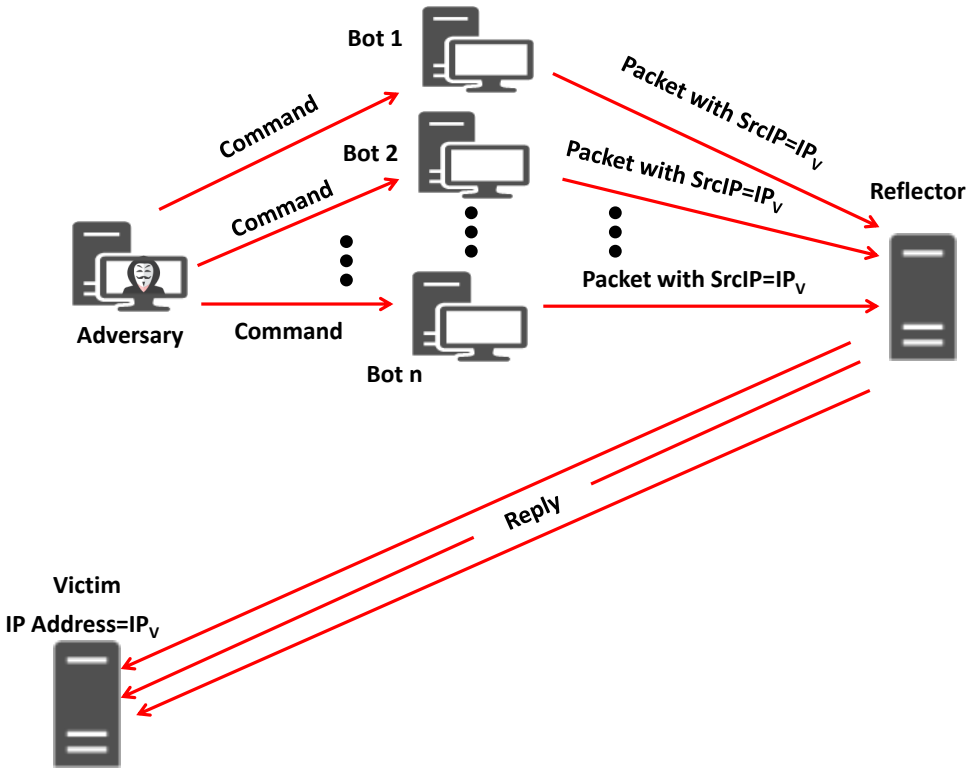



Figure 1.2: Reflection-based DDoS attack

such voluminous traffic, these attacks can easily be detected by Intrusion Detection Systems (IDSs). Another drawback of these attacks is the complexity involved to launch them as they require compromising several computers connected to Internet and use them as bots. Compromising systems and maintaining army of bots has become difficult for several fortunate reasons such as deployment of more secure firewalls on the edge routers of the networks. However, off-late, new type of DoS attacks known as application layer DoS attacks [11, 12] are gaining popularity. These attacks target top layer of the TCP/IP model which supports application and end-user processes. These attacks are capable of bringing down a server having huge computational power and network bandwidth using very limited resources.

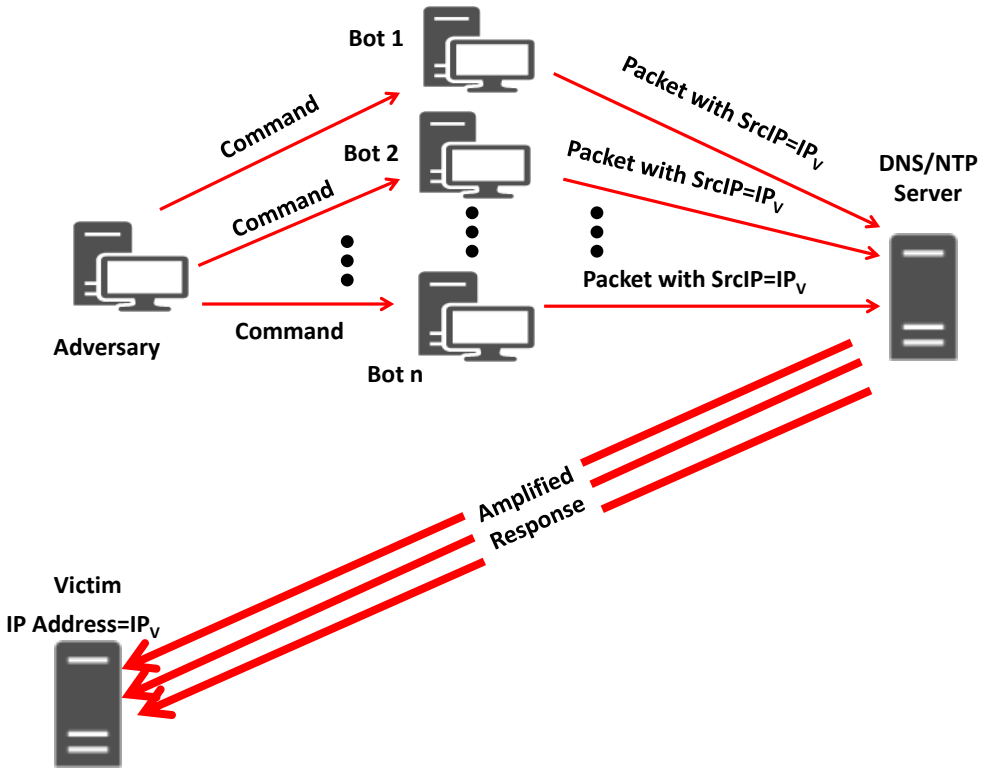



Figure 1.3: Amplification-based DDoS attack

### 1.1.3 Application Layer DoS Attacks

Application layer DoS attacks [11, 12] are launched by sending specially crafted requests in order to target the flaws or vulnerabilities present in either the protocol implementation or its design itself instead of victim's network bandwidth. Thus, launching these attacks do not require army of bots and can be launched even from a single computer having minimal bandwidth. Also, these attacks are much stealthier as compared to volumetric attacks (flooding attacks) as these attacks do not generate

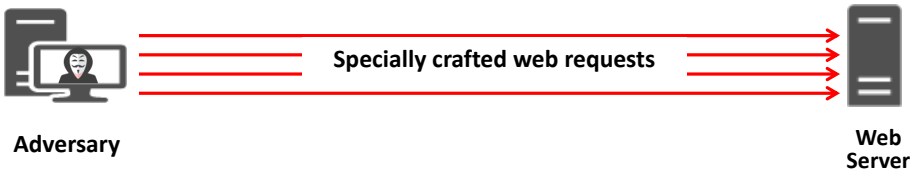



Figure 1.4: Application layer DoS attack

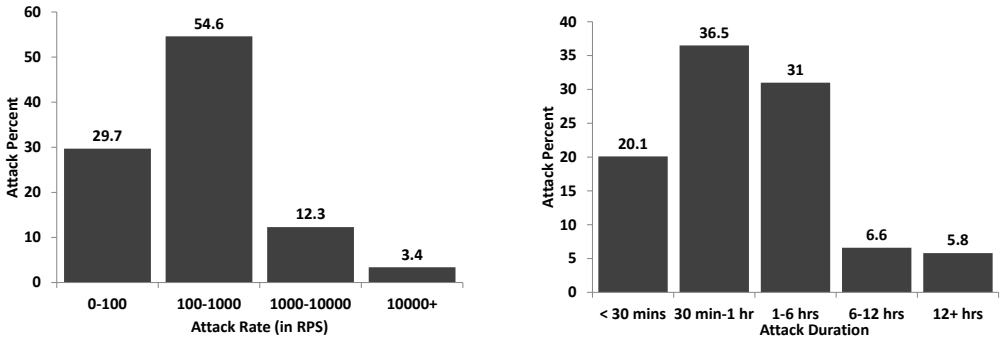
large amount of traffic. Unlike previous two types of attacks which disrupt all the services running in the victim, application layer DoS attacks target specific services in the victim to make it unavailable for genuine clients. For example, Slow Rate DoS attacks against HTTP protocol [5] intend to render web services unavailable for genuine HTTP clients. Similarly, attacks against NTP protocol [13, 14] prevent a genuine client from synchronizing its clock with a NTP server. Figure 1.4 shows an example where an adversary sends several specially crafted web requests to a web server to create DoS scenario.

### 1.1.4 Application Layer DoS Attack Incidents

According to *Global DDoS Threat Landscape Quarter 4, 2017*[1] report of Imperva, the number of network layer assaults in fourth quarter of 2017 were decreased by a huge margin of 50 % from the third quarter of 2017. At the same time, however, the number of application layer DoS attacks is doubling after every quarter of an year. The largest application layer DoS attack of fourth quarter of 2017 was launched at the rate of approximately 1.4M Requests Per Second (RPS) and approximately 16 % of attacks were launched at a rate higher than 1,000 RPS. Figure 1.5 [1] shows the distributions of attack rate in terms of Requests Per Second (RPS) and attack duration. Figure 1.5a shows the percentage of attacks and the rate with which they were launched in fourth quarter of 2017. We can notice that more than half of all attacks were launched at the rate between 100-1,000 RPS. The report also says that the duration of majority of application layer DoS attacks lasted between 30 minutes and 1 hour. Figure 1.5b shows the percentage of attacks and the time duration for which they lasted.

The report also presents a survey of the countries which were targeted and the countries from where attacks were originated. Figures 1.6a and 1.6b [1] show the statistics for the top attacking countries and top targeted countries respectively. We can see that United States tops the list of both most attacking and most targeted

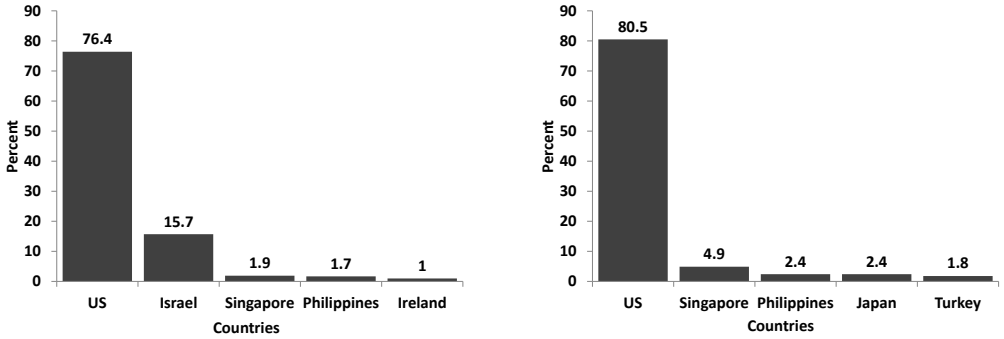
[1] https://www.incapsula.com/ddos-report/ddos-report-q4-2017.html




(a) (b)

Figure 1.5: Attacks and their (a) launching rate and (b) duration [1]

countries.




(a) (b)

Figure 1.6: Top (a) attacking and (b) targeting countries [1]

## 1.2 Motivation

Today, researchers and developers in the networking community expend considerable amount of time and effort to make an application layer protocol more efficient, however, designing secure protocols is not considered equally important. As a result, application layer protocols are left with large number of potential flaws or vulnerabilities which are then exploited to launch attacks such as application layer DoS attacks. There have been several popular incidences of application layer DoS attacks. For example, on October 21st, 2016, a very massive application layer DDoS

attack [2] was launched against the internet performance management company DYN's managed DNS infrastructure. Hackers used a botnet containing more than 100,000 infected devices to launch the attack against DNS infrastructure. For a couple of hours, high-profile websites such as Etsy, Github, Spotify and Twitter suffered service interruptions or went offline altogether. In another such incident, Scott Behrens, a security engineer at Netflix, launched a self-inflicted application layer DoS attack to showcase the possibility of an infiltration of Netflix's API or any other organization's APIs to cause a system failure from within. According to *The Netflix Tech Blog*[3], this experiment resulted into failure of 80 % API gateway errors. Beside such reported incidents of DoS/DDoS attacks, there have been several incidents of these attacks due to insider threats which are not reported. Several popular application layer DoS attacks such as DHCP starvation attack are launched within a local network and thus, are not reported. Also, according to a 2018 report[4], 53 % of the organizations confirmed insider attacks against them in the past one year. Given these types of DoS attacks can cause such chaos, application layer protocols should come under immediate scrutiny so that an appropriate vulnerability assessment of these protocols and their implementations can be performed.

In this work, we take into account four important application layer protocols - 1) Dynamic Host Configuration Protocol (DHCP) [15], 2) Hypertext Transfer Protocol (HTTP/1.1) [16], 3) its successor HTTP/2 [17] and 4) Network Time Protocol (NTP) [10] - to find potential vulnerabilities in them which can be exploited to launch application layer DoS attacks. While some of the vulnerabilities present in these protocols can not be patched as it may create surface for other types of attacks, some vulnerabilities require modifications in the protocol specification itself for appropriate patches [14]. However, changing an *Internet Standard Document* such as Request for Comment (RFC) needs deliberations and discussions between stakeholders and thus, takes a substantial amount of time. Moreover, reflection of these changes in the protocol

[2]https://www.tripwire.com/state-of-security/latest-security-news/dyn-restores-service-ddos-attack-brought-twitter-spotify-others/

[3]https://medium.com/netflix-techblog/starting-the-avalanche-640e69b14a06

[4]https://www.ca.com/content/dam/ca/us/files/ebook/insider-threat-report.pdf

implementations and releasing their newer versions by different vendors in the wild also take a long time. So as a first line of defense, there should be appropriate detection schemes that can be deployed to detect the application layer DoS attacks. Thus, we propose various detection techniques which can be implemented in the networks to detect the application layer DoS attacks proposed in this work.

## 1.3 Thesis Contributions

The thesis aims at proposing DoS attacks against various application layer protocols and detecting these attacks using appropriate detection mechanisms. The

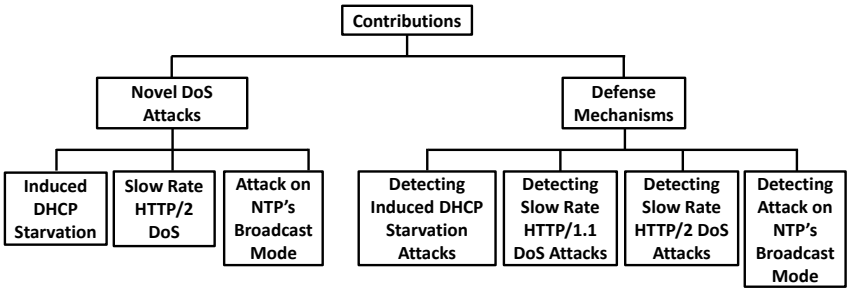



Figure 1.7: Thesis contributions

thesis contributions are shown in Figure 1.7 and are as follows:

**I. Induced DHCP Starvation Attack:** DHCP is used for automatic IP address allocation to the clients within a local network. As our first contribution, we propose a new attack against DHCP called as Induced DHCP starvation attack which can prevent a client from ever obtaining an IP address. The proposed attack exploits IP conflict detection schemes present at server and client sides. Using this scheme, a DHCP server, before offering an IP address to a client, checks if the address is not being used by some other client in the network. Similarly, the client, before configuring its interface with the offered IP address, also checks that no other client is using that IP address [15]. For conflict detection purpose, a broadcast probe request is sent in the network. If one or more probe replies are received such that the source

IP address of these replies is the IP address being verified, IP conflict is detected. In case of IP conflict, a DHCP client declines the offered IP address and these operations are repeated so that the client is left to starve forever. Depending on which conflict detection scheme is exploited and what type of probe is used (ARP/ICMP), there are three different variants of Induced DHCP starvation attack. The proposed Induced DHCP starvation attack can also be used to launch *targeted DNS spoofing* as a derivative attack within a local network wherein an adversary can send forged DNS replies in response to a particular DNS query sent by a particular client within the network.

**II. Detecting Induced DHCP Starvation Attack:** We propose three different approaches to detect Induced DHCP starvation attack. These are:

**1)** A statistical abnormality measurement technique is proposed that compares normal DHCP traffic profile generated during training phase with the profile generated in one of the time intervals $\Delta T$ of the testing phase. For this comparison, this scheme uses a metric known as Hellinger Distance [18] that represents dissimilarity between two probability distributions. If the calculated distance is greater than a predefined threshold, attack is detected in $\Delta T$.

**2)** The second approach simply compares the number of DHCP messages of a particular type received in a particular time interval $\Delta T$ of a day of training phase with the number of DHCP messages of the same type received in the same interval $\Delta T$ of the testing phase. If the difference in number of received messages exceeds a predefined threshold, Induced DHCP starvation attack is detected in $\Delta T$.

**3)** Our last detection approach uses six different one-class classification algorithms to detect different variants of the Induced DHCP starvation attack. The reason behind choosing different one-class classifiers is to help network administrators to select a suitable classification technique based on the requirement and suitability.

**III. Slow HTTP DoS Attacks against HTTP/1.1 and Detection:** We consider two popular Slow HTTP DoS attacks- *Slow Header* and *Slow Message Body*

attack and perform an empirical study of these attacks against different web servers and live websites. To launch these attacks, the malicious client first establishes enough number of connections with the web server and then sends incomplete request or message body from each of them to consume the connection queue space at the web server. This renders server unavailable for genuine clients, thereby, causing DoS scenario. We test these attacks against four popular web servers namely Apache, IIS, Nginx and Lighttpd and also perform an empirical study of these attacks against a large number of live websites hosted on servers present in different parts of the globe and furnish the results.

Since these attacks are a serious threat to web services [19] which our empirical study also confirms, we propose an anomaly detection scheme to detect these attacks. Similar to our first approach to detect Induced DHCP starvation attack, this detection scheme also uses Hellinger Distance to compare normal HTTP traffic profile generated during training phase with the profile generated during a particular time interval $\Delta T$ of testing phase. If calculated Hellinger Distance is greater than a predefined threshold, Slow HTTP DoS attacks are detected in $\Delta T$.

**IV. Slow Rate DoS Attacks against HTTP/2 and Detection:** HTTP/2 is the recent version of HTTP protocol standardized in May, 2015. HTTP/2 not only supports all the basic features of HTTP/1.1 but it is also very efficient in utilizing TCP transmission capability [17]. We propose five new Slow Rate DoS attacks against HTTP/2 protocol. To the best of our knowledge, proposed attacks are the first reported DoS attacks against HTTP/2 protocol. These attacks are launched by injecting specially crafted HTTP/2 requests. Similar to Slow HTTP DoS attacks, these attacks also target to deplete all the free connection slots available in web server's connection pool. The proposed attacks are tested against four popular HTTP/2 supporting web servers namely Apache, Nginx, H2O and Nghttp2 and it is observed that all the web servers are vulnerable to at least one of the attacks. We also compare both HTTP/1.1 and HTTP/2 protocols on the basis of similarity of threat vectors and show that HTTP/2 has relatively more number of threat vectors

as compared to its predecessor. To detect the proposed attacks, an anomaly detection scheme is also proposed. The proposed detection method uses chi-square test [20] to identify deviations in HTTP/2 traffic patterns in the presence of Slow Rate DoS attacks.

**V. Attack against NTP's Broadcast Mode and Detection:** NTP protocol is used to synchronize clocks of the systems connected to Internet. As our last contribution, we propose a new attack that exploits a vulnerability present in NTP broadcast mode to prevent a client from synchronizing its clock with a server. This attack works by repeatedly sending spoofed packets to the client and broadcast NTP server. We test the attack in a real network and show that the two most recent versions of NTP's reference implementation *ntpd* is vulnerable to the attack. As the attack involves sending spoofed NTP packets, we also propose a technique to detect spoofed NTP packets and hence the proposed attack. This technique verifies the authenticity of an NTP packet by proactively probing the source of the packet and then matching the Time-To-Live (TTL) value present in IP header of received probe reply and the NTP packet under consideration. If inconsistency in TTL value is found, the packet is considered as spoofed, otherwise it is considered as coming from genuine client.

## 1.4 Organization of the Thesis

The rest of the thesis is organized as follows.

**Chapter 2:** In this chapter, we describe the previously known application layer DoS attacks and the defense mechanisms to counter these attacks. Subsequently, we discuss the research gap in the field and present problem statement of the thesis.

**Chapter 3:** In this chapter, we propose an Induced DHCP starvation attack and its implication in the form of DNS spoofing in local networks.

**Chapter 4:** This chapter deals with detecting different variants of the proposed Induced DHCP starvation attack using three different detection approaches.

**Chapter 5:** In this chapter, we study the impact of two popular Slow HTTP DoS attacks against different web servers and live websites. We also propose a statistical abnormality measurement technique to detect Slow HTTP DoS attacks.

**Chapter 6:** In this chapter, we propose five new Slow Rate DoS attacks against HTTP/2 protocol and study the behaviour of different popular web servers against these attacks. We also propose an anomaly detection scheme to detect these attacks.

**Chapter 7:** In this chapter, we propose a new attack that can prevent a client from synchronizing its clock with a broadcast NTP server. We also propose an event based technique to detect spoofed packets which helps in preventing the proposed attack.

**Chapter 8:** This chapter summarizes the work presented in this thesis and provides directions to future work in this area.

# Chapter 2

# Literature Review

## 2.1 Introduction

Application layer DoS attacks target potential flaws and vulnerabilities present in the normal operation of the protocols. These attacks require minimal bandwidth and computational power from a malicious client's perspective. Also, these attacks are much stealthier as compared to volumetric attacks (flooding attacks) as these attacks do not generate large amount of traffic. Due to these reasons, the popularity of application layer DoS attacks among various underground hacking communities is continuously increasing. Since last decade, researchers in the security community scrutinized various application layer protocols and disclosed severe attacks. We categorize these attacks into different classes as shown in Figure 2.1. In this chapter, we present a systematic review of previously known application layer DoS attacks and the defense mechanisms proposed to counter these attacks. We discuss the execution method, effectiveness and shortcomings of different attacks and also the ability of previously known defense mechanisms to counter the attacks.

The structure of rest of this chapter closely follows Figure 2.1. In Section 2.2, we discuss application layer DoS attacks which are protocol specific and the known defense mechanisms to counter them. We describe various generic application layer DoS attacks and the defense mechanisms to counter them in Section 2.3. We present the research gap in this field and the problem statement in Section 2.4. Finally, the

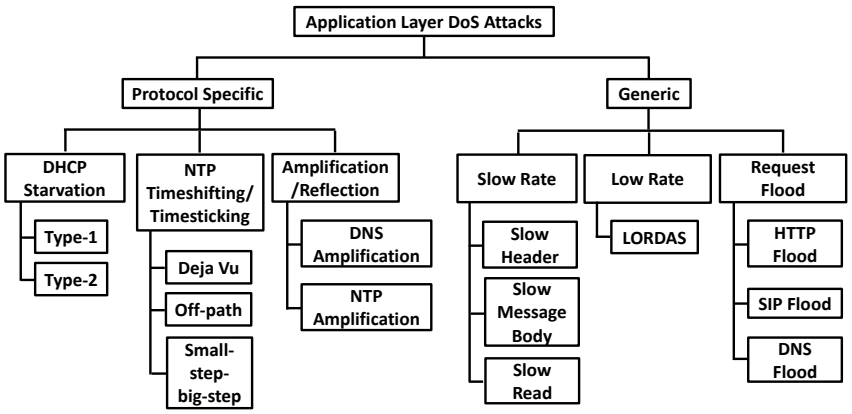



Figure 2.1: Categorization of application layer DoS attacks

chapter is concluded in Section 2.5.

## 2.2 Protocol-specific DoS Attacks

Protocol-specific DoS attacks are effective against particular application layer protocols only. Examples of protocol specific attacks are *DHCP Starvation*, *Timeshifting/Timesticking* and *Amplification/Reflection* attacks which are effective against DHCP, NTP and DNS protocols respectively. In this section, we describe these attacks and the defense mechanisms known to counter them.

### 2.2.1 DHCP Starvation Attack

A DHCP server maintains a pool of IP addresses from which it chooses one address and offers it to a client willing to join the network. A malicious client can launch classical DHCP starvation attack to consume all the IP addresses available in the pool so that the DHCP server could not offer an IP address to the clients joining the network in future. Since the clients are not able to obtain the IP address, they can not communicate with other devices which leads to DoS. The malicious client consumes the IP pool by generating several random MAC addresses and sending an IP request message by spoofing each of the MAC address. The classical DHCP starvation attack can be launched using two methods [21]. In first method (Type-1), the malicious client

puts same (randomly generated) MAC address in the DHCP header and Ethernet header of the frame carrying DHCP message while in the second method (Type-2), the malicious client puts randomly generated MAC address in the DHCP header and its own MAC address in the Ethernet header. Since classical DHCP starvation attack requires MAC address spoofing, it is not effective in wireless networks secured with Wireless Protection Access 2 (WPA2). We present a comprehensive discussion on limitation of this attack in Chapter 3 of the thesis.

#### 2.2.1.1 Countermeasures

Several defense mechanisms have been proposed in the literature to counter classical DHCP starvation attack. These mechanisms can be categorized into different

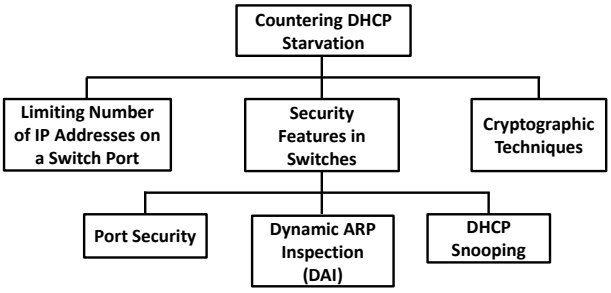



Figure 2.2: Defense mechanisms to counter DHCP starvation attack

classes as shown in Figure 2.2 and are as follows:

**1. Limiting Number of IP Addresses on a Switch Port:** There are few approaches [22, 21] which suggest to put a limit on number of IP addresses that can be assigned to a particular port of a switch. However, a malicious client can simply connect to a switch port and consume fair share of IP addresses allotted at that port. This leads to starvation for upcoming legitimate clients which are going to connect to the port, the malicious client is already connected with. Thus, limiting number of IP addresses on a switch port does not mitigate classical DHCP starvation attack.

**2. Security Features in Switches:** Modern network switches are equipped with built-in and easy-to-configure security features such as port security [23], DHCP

snooping [24] and Dynamic ARP Inspection (DAI) [25]. Port security limits the number of MAC addresses seen on a port of a network switch. Once this limit is reached, the port is automatically disabled and an SNMP trap is sent. Since a malicious client uses randomly generated MAC addresses to launch Type-1 classical DHCP starvation attack, port security can mitigate this attack. However, Type-2 classical DHCP starvation attack can not be mitigated using port security feature because malicious client uses its real MAC address in Ethernet frames carrying DHCP messages. Since port security does not analyze application layer protocols' headers, it is unable to mitigate Type-2 classical DHCP starvation attack. The second security feature DHCP Snooping, however, can notice the difference in MAC address present in the Ethernet and DHCP header. Thus, it can mitigate Type-2 classical DHCP starvation attack. However, since this feature does not limit the number of permissible MAC addresses per port in a network switch, it can not mitigate Type-1 classical DHCP starvation attack. The third security feature, Dynamic ARP Inspection (DAI), determines the validity of an Address Resolution Protocol (ARP) [26] message by checking valid IP-MAC bindings stored in DHCP snooping database. If a DAI enabled switch receives an ARP message that induces such an IP-MAC binding which contradicts with an IP-MAC binding present in the DHCP snooping database, the ARP message is dropped. Since Type-1 and Type-2 classical DHCP starvation attack do not require sending spoofed ARP replies, they can not be mitigated using DAI.

**3. Cryptography- and Certificate-based Techniques:** Various cryptographic techniques [27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37] are proposed in the literature to enforce authentication in DHCP to mitigate starvation attacks. However, these techniques have several issues and limitations. First, implementation of these techniques in a local network is very complex. Moreover, these techniques generate huge traffic load due to the involvement of various third party modules. Due to such complex infrastructural requirement, the cryptographic techniques are rarely implemented in local networks.

### 2.2.2 Timeshifting/Timesticking Attacks

Timeshifting/timesticking attacks target the normal operation of NTP protocol and aim to prevent computers from synchronizing their clocks with more accurate NTP servers. Authors of [13, 14, 38] recently disclosed these attacks. These attacks exploit the vulnerability present in either the protocol specification or its implementation. These attacks are as follows:

**1. Deja Vu (On-path Timesticking Attack):** To launch this attack (CVE-2015-7973), a malicious client present between a NTP broadcast server and a victim client first collects and records a contiguous sequence of NTP server's broadcast packets and then replays this sequence of packets (while dropping the legitimate packets), over and over, to the victim client. As a result, the victim gets stuck at a particular time. After attack disclosure, *ntpd* v4.2.8p6 and later versions were patched to mitigate this attack [13].

**2. Off-path DoS Attacks:** In [13], authors proposed an attack (CVE-2015-7979) wherein an off-path malicious client causes an error in the NTP operation of victim client by sending a badly authenticated NTP broadcast packet. As a result, the victim client is not able to collect enough good NTP packets from the broadcast server to update its local clock. Following attack disclosure, appropriate patches have been added to *ntpd* v4.2.8p6 and later versions to mitigate this attack [13]. In [38], authors described few other variants of this attack which are captured in CVE-2015-7704 and CVE-2015-7705 following which the vulnerable *ntpd* versions were patched.

**3. Small-step-big-step Attack:** In this attack, a malicious client sends bogus small time shifts to a victim client and then, when the malicious client is ready, it sends a big time shift to the victim client, performs harmful activities, and finally sends another big time shift that sets the victim client's clock back to normal. As a result, this attack is quite stealthy. This issue is captured in CVE-2015-5300 following which patches have been added to the vulnerable *ntpd* versions [14].

We describe the working principle of these attacks in detail in Chapter 7 of the

thesis.

#### 2.2.2.1 Countermeasures

After the disclosure of timeshifting/timesticking attacks, the vulnerable *ntpd* versions were appropriately patched. However, as a first line of defense, authors in [38, 13, 14] also recommended some countermeasures to prevent these attacks. In [13], authors first suggested that NTP should drop badly authenticated packets instead of directly tear down established associations on receiving such malformed packets. The second suggestion was to prevent replaying of broadcast packets by using an incrementing counter in broadcast packet which increases monotonically. However, both these recommendations require change in RFC 5905. In another recommendation, authors suggested that only the broadcast server should be able to sign broadcast packets and it should not distribute its symmetric key to other clients into the network. However, signing every NTP packet with a standard public-key digital signature scheme must be very fast as it requires a lot of cryptographic computation that itself can harm NTP's clock synchronization process. Another recommendation was to put a check on the set of IP address that can communicate with an NTP client, especially for those clients which are not designed to serve time to Internet on a large scale.

Along with these countermeasures, a new backward-compatible client/server protocol is proposed in [38] that sets the timestamp of the last NTP query sent by the client to a random 64-bit value instead of setting it to zero. As a result, the malicious client can not guess this random timestamp due to which it can not spoof NTP server's response packets. Thus, the attack [38] wherein the malicious client can spoof NTP server's response packets with their origin timestamp set to zero gets mitigated.

### 2.2.3 Amplification/Reflection Attack

This attack uses reflection and amplification techniques [39, 8] in tandem to launch DoS attack. They take advantage of publicly accessible DNS and NTP servers to overload victims with response traffic. Reflection technique involves spoofing the source

address of request packets by an adversary pretending to be the victim and sending the requests to a DNS or NTP server. In response, the server sends the reply packets directly to the victim. This technique hides the real IP address of the adversary from both the victim's system and the abused server.

The other technique is traffic amplification. The adversary's goal is to make the abused service produce as much response data as possible. The ratio between the sizes of the response and the request is called amplification factor. The adversary wants to achieve the largest possible ratio. For example, if an open DNS service is used to flood a victim, an amplification factor of up to 70 times can be observed [39]. However, if an open NTP service is used, the adversary can achieve an amplification factor of up to 557 times [8]. When these techniques are repeatedly used together, an attack is generated. Servers in multiple locations can be involved to produce more devastating results. It is important to realize that abused services are victims as well as those targeted by reply floods. These servers suddenly have to deal with abnormally large amounts of spoofed requests that may prevent them from serving legitimate traffic.

#### 2.2.3.1 Countermeasures

Launching amplification/reflection DoS attacks requires spoofing IP address of a victim and thus, several works in the literature recommend mitigating IP address spoofing in order to counter amplification/reflection DoS attacks. The approaches proposed in the literature to counter IP spoofing based attacks can be divided into three different categories as shown in Figure 2.3 and are described as follows:

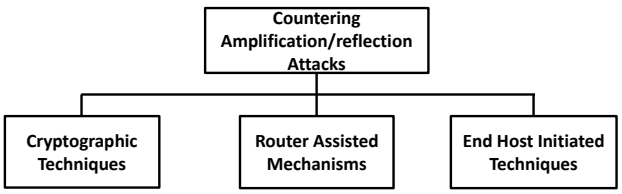



Figure 2.3: Defense mechanisms to counter amplification/reflection attack

**1. Cryptography-based Techniques:** These techniques rely on cryptography methods to verify the authenticity of received packets. Spoofing Prevention Method (SPM) [40] is a such a scheme in which participants (involved Autonomous Systems (ASes)) authenticate their packets by a source AS key. Source AS key is assigned to every autonomous system and shared with all other autonomous systems. Recipient router or gateway verifies the AS key of received IP packets and decides whether it is spoofed or genuine. This method brings a challenge of exchanging keys between all ASes in the Internet. In addition, as all clients within an AS use the same AS key, a client may spoof an IP address of other client within the same AS. To alleviate this problem, Shen et al. [41] extended SPM to have two levels of authentication- first within the Autonomous System and other at Inter Autonomous System. The signature added by end client is verified by gateway of that network and signature added by gateway of a network is verified by its peer at destination network. TCP SYN cookies [42] which is a technique introduced to prevent TCP SYN flood attacks also falls under this category.

**2. Router Assisted Mechanisms:** A class of work known as packet marking [43] helps in tracing back the spoofed IP packets. In this category, each router on way probabilistically adds its identity to a packet so that the path taken by the packet can be traced back. Several improvements [44, 45] are made to this scheme including the one which uses TTL as a parameter [46] to mark the packet. Strayer et al. [47] proposed an alternative technique where intermediate routers store details of packets in their Bloom filters. The IP traceback is performed by querying relevant routers if their local Bloom filter contains details about the packet. This approach puts lot of overhead on routers and given the speed of today's networks, it is not feasible to deploy such approach in practice. Kim et al. [48] described a technique where intermediate routers generate a score based on the attributes of a particular packet and decide to drop a packet based on that score. A US patent [49] describes to use mapping between IP addresses and autonomous systems at the gateway level to detect spoofed packets. However, these techniques severely suffer from scalability issues.

**3. End Host Initiated Techniques:** Another major technique of detecting spoofed packets found in the literature is through TTL values of received packets [50, 51]. These techniques work on the premise of predicting the initial TTL value based on received TTL value of packets. Predicting the initial TTL values is possible as many operating systems use only a few selected initial TTL values such as 32, 64 and 128. Detection of spoofed packets is done through another known fact that any two clients connected to Internet are not more than 30 hops away [52]. Thus by guessing approximate number of hops taken by a packet, initial TTL value set by a client is found. However, the scheme proposed in [50, 51] causes a collateral damage for some legitimate clients during the attack time [53]. According to this scheme, the new packets with source IP address not present in the IP2HC table (a database of previously seen IP addresses and corresponding hop count values) are considered as spoofed packets during the attack time even though these packets are legitimate. As a result, such packets are dropped.

## 2.3 Generic DoS Attacks

Unlike protocol specific attacks, this class of application layer DoS attacks is effective against large number of application layer protocols. Examples of generic attacks are *Slow Rate*, *Low Rate* and *Request Flood* attacks which are effective against large number of popular protocols such as HTTP, SMTP, FTP, SIP, DNS, etc. In this section, we describe these attacks and the defense mechanisms known to counter them.

### 2.3.1 Slow Rate DoS Attacks

Various Slow HTTP DoS attacks have been disclosed since last decade. The most popular among them are *Slow Header*, *Slow Message Body* and *Slow Read* attacks and are as follows:

**1. Slow Header Attack:** HTTP protocol requires GET headers to be completely received before they are processed [16] by a web server. If an incomplete

HTTP GET header is received, the server assumes that the client is operating from a slow Internet connection and thus keeps its resources busy waiting for the rest of the header. In order to generate this attack, a malicious client establishes enough connections to web server and sends incomplete HTTP GET requests from each of them to fill the connection queue of the web server so that legitimate clients can not connect to the server.

**2. Slow Message Body Attack:** In this attack, the malicious client sends complete HTTP POST header however it sends the message body in very small chunks, thereby, forcing server to wait to receive complete message body. When web server receives such requests, it assumes that only a part of the message body is received and remaining part is still pending. Thus, server keeps its resources busy waiting for the rest of the message. The malicious client creates enough number of connections to web server and sends such HTTP messages from each of them to fill the connection queue of the web server, thereby, causing a DoS scenario.

**3. Slow Read Attack:** Slow Read attack against a HTTP/1.1 server requires a malicious client to send a legitimate GET request after TCP 3-way handshake and then immediately advertises a receiver window size of zero byte. As a result, server stops sending data although it holds the connection in the hope of receiving a non-zero window size advertisement from the client. Client on the other hand never advertises non-zero window size, thus, forcing server to wait for indefinite time. Nevertheless, web server implementations have been patched to immediately close the connection on which a receiver window of size zero is advertised and thus, the servers are no longer vulnerable to Slow Read attack.

### 2.3.2 Low Rate DoS Attack

Kuzmanovic et. al [54] investigated a Low Rate TCP-targeted DoS attack wherein a malicious client creates DoS scenario by the use of a low-rate traffic against a server. In particular, the malicious client exploits the knowledge of Retransmission Time Out (RTO) timer for congestion control implemented in TCP. By exploiting this mechanism, the malicious client can launch an ON/OFF attack that results in low-rate

traffic but with high efficiency to create DoS scenario. Maciá-Fernández et al. [55, 56] adapted this attack and proposed an attack called LoRDAS that targets application layer protocols which use TCP as the transport layer protocol. In this attack, a malicious client can forecast the instants at which the server retrieves one request from the connection queue and, therefore, generates a free slot in the queue. After this, the malicious client sends a legitimate request to the server to fill the available space in the connection queue. In such a way, a time comes when the whole connection queue is filled with the requests sent by the malicious client. Thereafter, whenever the server retrieves a request from the queue, processes it and returns the response, the malicious client immediately establishes another connection to capture the free slot. This prevents other legitimate requests from being served by server which results into DoS scenario. Since this attack requires sending small traffic at regular intervals, the detection mechanisms deployed to monitor high-bandwidth traffic fails to detect this attack.

### 2.3.3 Request Flood Attacks

In this type of attack, an adversary sends a large number of requests to a server in order to overwhelm it. These requests are sent from either a single computer (DoS attack) or a large number of computers (DDoS attack). In the latter case, the adversary controls several computers and commands them to send flood of requests to the server to create DoS scenario. Since the adversary has option of sending flood of packets belonging to different protocols, there are different types of request flood attacks such as *HTTP Flood*, *SIP Flood*, *DNS Flood*, etc. These attacks are as follows:

**1. HTTP Flood:** HTTP flooding attack is the most popular type of request flooding attack. An adversary can launch this attack by sending session connection request rates to a web server at a very high rate which exhausts the server resources and leads to DoS scenario on the server. One of the famous attacks in this category is the HTTP get/post flooding attack [5] in which an adversary generates a large number of valid HTTP requests (get/post) to a victim web server. In [2] and [3], authors

described different flooding attacks which involve sending flood of web requests to a HTTP/2 enabled server and then monitoring different resource parameters such as CPU usage, memory consumption, network throughput and packet loss. These attacks require sending a large number of HTTP/2 requests to create DoS scenario.

**2. SIP Flood:** SIP flooding attacks are characterized by sending huge number of messages to SIP server or any SIP entity so as to make it difficult to process these messages and eventually to crash the system. This attack mainly targets the resources of the SIP system like memory, CPU and bandwidth to crash it. SIP flooding attack can be carried out on any component of the SIP system such as SIP Registrar, SIP proxy, etc.

**3. DNS Flood:** In this attack, an adversary sends a large number of DNS requests to a DNS server so that it is not able to process DNS requests sent by benign clients. This results into DoS scenario.

### 2.3.4 Countermeasures

The research community has invested considerable effort to detect generic application layer DoS attacks and thus, a comprehensive literature is available on this class of attack [5]. In this thesis, we discuss defense mechanisms that focus to combat Slow Rate, Low Rate and Request Flood DoS attacks against HTTP protocol. The defense mechanisms to counter these three attacks can be classified into two categories as follows:

**1. Anomaly-based Techniques:** There are many anomaly detection techniques that have been proposed to detect anomalies in HTTP traffic by simply looking at deviations from normal flows statistics [57]. To detect Slow Rate DoS attacks, authors of [58] proposed a scheme that extracts traffic features like amount of time required to generate a HTTP request. Based on this feature, a comparison is made between sampling distribution generated from unknown traffic, potentially anomalous, and distribution generated from legitimate HTTP traffic to detect Slow Header attack. However, authors did not discuss how the proposed scheme can

be adapted to detect Slow Message Body attack. In another work, Giralte et al. [59] used three types of analyzers to detect attacks against HTTP wherein the first analyzer performs statistical analysis of HTTP flow such as request and response size. The second analyzer checks access behaviour of users using a graph, to model the several paths of the web server as well as the different costs of navigating through the server. The last analyzer counts the frequency of HTTP operations carried on the web server by a particular user. However, the proposed technique can not detect Slow HTTP *Distributed* DoS (DDoS) attack if number of involved bots is large and each bot contributes very less traffic to evade detection. Moreover, to avoid detection, a determined adversary can simulate the behaviour of normal users by using techniques such as Web Usage Mining (WUM) which involves the application of machine learning techniques on web data to extract behavioural patterns of web users [60]. The authors of [61, 19] presented a technique that tracks number of packets a web server receives in a given period of time. This feature is monitored in two subsequent temporal periods to detect normal or anomalous behavior. The weakness of this technique lies in the feature selection itself. Monitoring number of packets received at a web server can cause high false positive rate because if a high burst of traffic is received due to normal scenarios like Flash Event [62], the detection technique notifies this as an attack traffic. Moreover, these techniques can not detect Slow Rate DDoS attacks [63].

**2. Implementation Modules:** To detect HTTP DoS attacks, few modules such as 'Core' [64], 'Antiloris' [65], 'Limitipconn' [66], 'mod_reqtimeout' [67], etc. are particularly available for Apache server. The first module 'Core' buffers entire HTTP requests at the kernel level. Once an entire request is received, the kernel then sends it to the server. This ensures that incomplete HTTP requests do not go to the server's connection queue. On the other hand, second and third modules limit the number of incoming complete/incomplete requests on a per IP basis while the fourth module requires a sender to send a complete request and/or complete message body within a predefined amount of time. However, these modules either have tendency to block legitimate requests from being served or they are unable to mitigate Slow

Rate DDoS attacks. Moreover, authors of [68] showed that a web server configured with these modules, though, repulsed the attack; there was a noticeable delay in the server's response when attack was launched by 4 bots. Thus, it is possible that a more massive attack scenario can affect the server's performance.

Although we discussed several previously known defense methods to counter attacks against HTTP protocol, these methods are either not adapted to detect all types of Slow Rate DoS attack [58] or unable to detect Slow Rate Distributed DoS (DDoS) attacks launched using several compromised bots [59] or known to generate large number of false positives [61, 19] in case of scenarios such as Flash Event [62].

## 2.4 Research Gap

With this literature review, we presented state of the art on application layer DoS attacks and defense mechanisms. As evident from the existing literature, several application layer DoS attacks have been proposed against protocols such as DHCP, HTTP, DNS, SIP, VoIP and NTP especially in the last decade and the tools and techniques used to launch these attacks are continuously evolving with time. The security aspects of various popular application protocols have always been overlooked and instead, performance and efficiency of these protocols have been given more importance. As a result, these protocols are left with potential vulnerabilities that provide surface to launch application layer DoS attacks. Thus, we argue that the most commonly targeted application layer protocols such as DHCP, HTTP, DNS and NTP [69, 70] should come under immediate scrutiny so that a detailed vulnerability assessment of these protocols can be performed.

It is also evident from the literature that several defense mechanisms have been proposed to counter application layer DoS attacks. Some defense mechanisms detect attacks by identifying anomalies in the traffic profile of different protocols. However, these defense mechanisms can detect only a subset of attacks against an application layer protocol as discussed in Section 2.3.4 of this chapter. Some defense mechanisms counter the attacks using either the security features present in networking devices

or software security modules enabled in a server, e.g. a web server. The drawbacks of these defense mechanisms is that either they can detect only some of the DoS attacks or they generate large number of false positives as discussed in Section 2.2.1.1 and Section 2.3.4 of this chapter respectively. Some of the defense mechanisms use cryptography to effectively mitigate different types of identity spoofing based attacks. However, these mechanisms suffer from various drawbacks such as high implementation complexity, traffic overhead, etc. as discussed in Section 2.2.1.1 of this chapter. Due to such complex infrastructural requirement, the cryptographic techniques are rarely implemented. Few defense mechanisms also propose to patch known vulnerabilities in the protocol by modifying the protocol specification itself. However, as discussed in Chapter 1 of the thesis, modifying a protocol specification needs deliberations and discussions between stakeholders and thus, takes a substantial amount of time. Moreover, reflection of these changes in the protocol implementations and releasing their newer versions by different vendors in the wild also take a long time.

### 2.4.1 Problem Statement

Given the above discussed research gap, we contribute towards identifying vulnerabilities in three application layer protocols (DHCP, HTTP and NTP) which can be exploited to launch DoS attacks and proposing defense mechanisms to counter the attacks. The problem statement is as follows:

> **Identifying potential vulnerabilities in three most commonly targeted application layer protocols (DHCP, HTTP and NTP) which can be exploited to launch DoS attacks and proposing appropriate detection methods which can be deployed as a first line of defense to counter the attacks.**

## 2.5 Conclusion

In this chapter, we presented a systematic literature review of application layer DoS attacks and the defense mechanisms proposed to counter these attacks. We classified application layer DoS attacks into different categories and explained the working principle of these attacks. Further, we described the detection ability, effectiveness and shortcomings of various defense mechanisms proposed to detect and mitigate each of these attacks. We also presented the research gap in this field by discussing the lack of study performed on security aspects of different application layer protocols and the limitations of defense mechanisms known to counter application layer DoS attacks. Subsequently, we presented problem statement for the work presented in this thesis.

# Chapter 3

# Induced DHCP Starvation Attack against DHCP Protocol

## 3.1 Introduction

Dynamic Host Configuration Protocol (DHCP) [15] is used to automatically configure DHCP enabled clients in a network with IP address and network parameters. DHCP enables network administrators to define TCP/IP configurations from a central location, thus, helps to efficiently handle any network configuration changes in the future. It also minimizes configuration errors such as address conflicts. However, absence of any built-in authentication scheme makes this protocol vulnerable to a popularly known Denial of Service (DoS) attack called DHCP starvation attack (henceforth we name this attack as *classical* DHCP starvation attack). In this attack, a malicious client prevents other clients in the network from obtaining an IP address from a DHCP server. Since the clients are not able to obtain the IP address, they can not communicate with other clients in the network. There are various open source tools such as Gobbler [71], DHCPIG [72] to launch this attack. Though DHCP starvation attack is effective in wired network topologies, it is very difficult to launch this attack in wireless networks.

In this chapter, we describe the practical difficulty in launching the DHCP starvation attack in wireless networks and subsequently, we propose new DHCP starvation

attack termed as *Induced DHCP starvation attack* that is effective in both wired and wireless networks. Our contributions in this chapter are:

**1.** We propose Induced DHCP starvation attack that is effective in both wired and wireless network. This attack abuses IP address conflict detection scheme used in DHCP clients and server as a precautionary measure to ensure that an IP address is not in use by some other client in the network.

**2.** We compare Induced DHCP starvation attack with classical DHCP starvation attack and show that Induced DHCP starvation attack is easier to launch and requires less number of messages as compared to the classical DHCP starvation attack.

**3.** We present an easy way of launching DNS spoofing attack in a local network using proposed Induced DHCP starvation attack.

Rest of the chapter is organized as follows. We describe the basics of DHCP protocol and its working in Section 3.2. The well known classical DHCP starvation attacks are discussed in Section 3.3. We present three variants of proposed Induced DHCP starvation attack in Section 3.4. The experiments performed to test the proposed attack are described in Section 3.5. We present an implication of the proposed attack in the form of selective DNS spoofing in Section 3.6. Finally, we conclude the chapter in Section 3.7.

## 3.2 DHCP Protocol

The working of DHCP is described in RFC 2131 [15]. DHCP is used to configure DHCP enabled clients in a network with IP address and other network configuration parameters such as subnet mask, default gateway's IP address, DNS server's IP address, etc. DHCP functionality comes pre-installed as a default feature in most of the modern operating systems. DHCP is a good alternative to the time-consuming and error prone manual configuration of the network settings on a client. In this section, we describe the process of automatic IP address allocation using DHCP, different types of DHCP messages exchanged during this process and the common message format for all message types.

### 3.2.1 Automatic IP Address Allocation using DHCP

In a network, a DHCP server is configured with a pool of IP addresses and other network configuration parameters. DHCP enabled clients join the network and rely on the DHCP server to obtain IP address. This automatic IP address allocation and configuration is done through what is commonly known as *DORA* process where 'D','O','R' and 'A' stand for 'Discover', 'Offer', 'Request' and 'Acknowledgment' respectively. The steps involved in *DORA* process are shown in Figure 3.1:

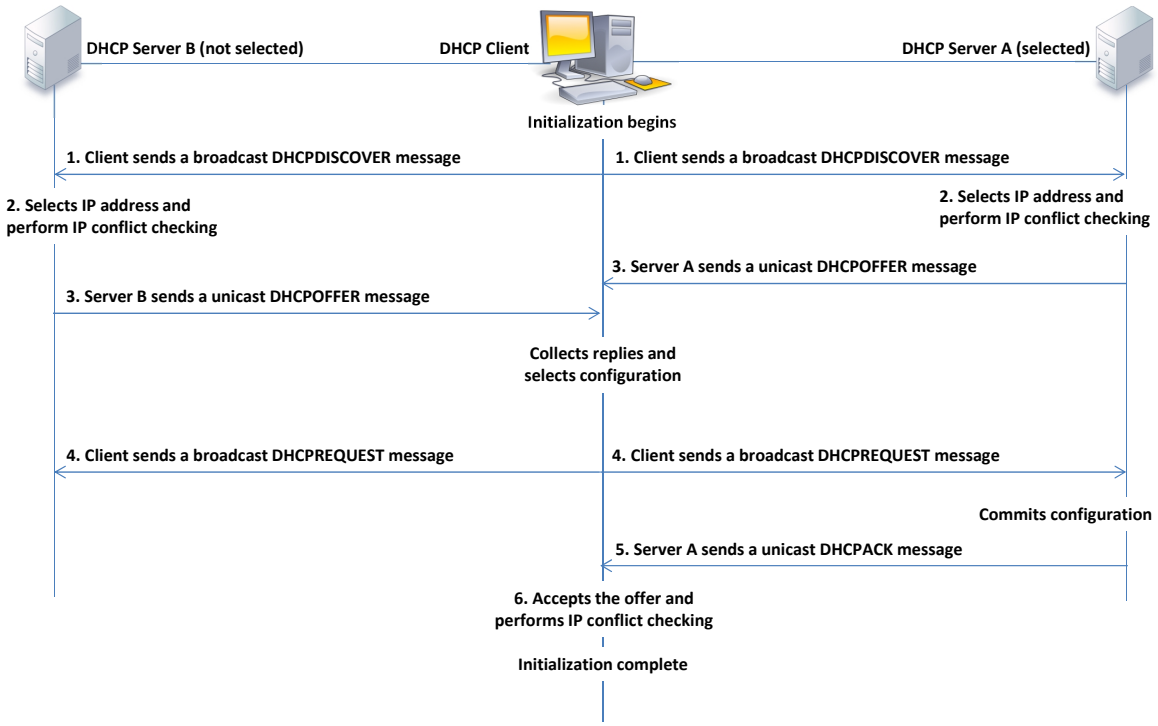



Figure 3.1: Exchange of messages during normal DHCP operation

**1. DHCPDISCOVER by Client:** As soon as a DHCP enabled client joins the network, it sends a broadcast DHCPDISCOVER message in the network to find available DHCP server(s) in the network which can issue IP address and network configuration parameters to it.

**2. IP Conflict Checking by Server:** RFC 2131 [15] suggests that a DHCP server and a DHCP client, before offering and configuring its interface with an IP address respectively, should ensure that the IP address is not in use by any other client in the

network. This precautionary checking is called *IP conflict detection*. Thus, on receiving DHCPDISCOVER message sent by the client in previous step, the DHCP server selects one IP address from the address pool and sends a broadcast probe request to check if the IP address is being used by any other client in the network. If server does not receive a probe reply, server offers the IP address to the client. However, if server receives the one or more probe replies in response to the probe request sent, server marks this IP address as unavailable for one lease period (the time for which IP address is allocated to a client) and starts performing IP conflict detection for the next available IP address in the pool. The server usually performs this check for next three IP addresses available in the pool. If it receives the probe reply for all these address, it finally gives up without offering IP address to the client. However, client again sends a DHCPDISCOVER message due to which the server again starts performing this conflict detection check for the next available IP addresses.

**3. DHCPOFFER by Server:** To offer the IP address selected in the last step, DHCP server sends a unicast DHCPOFFER message to the client that contains the to-be-offered IP address and other network configuration parameters. If more than one DHCP servers are present in the network, each one of them sends a DHCPOFFER message to the DHCP client.

**4. DHCPREQUEST by Client:** After receiving the DHCPOFFER message, the client broadcasts a DHCPREQUEST message to request the IP address offered by the DHCP server. If more than one servers respond to client with offers, the DHCP client selects an offer from one of the DHCP servers and implicitly declines offers made from remaining DHCP servers using this DHCPREQUEST message. As a result, the other DHCP servers relinquish the offer made and return the offered IP address back to the pool.

**5. DHCPACK by Server:** On receiving DHCPREQUEST message sent by the client in previous step, the selected DHCP server sends a unicast DHCPACK message to the client that contains the information like lease time and other network configuration parameters.

**6. IP Conflict Checking by Client:** As soon as client receives the DHCPACK

message, it performs IP conflict detection by sending a broadcast probe request. If client does not receive the corresponding probe reply, client configures its interface with the offered IP address. Otherwise client sends a DHCPDECLINE message to the server to inform that some other client in the network is already using the offered IP address. On receiving the DHCPDECLINE message, server marks the IP address as unavailable for one lease period. In this case, the client reinitiates *DORA* process from Step 1 to obtain an IP address from the server.

### 3.2.2 Other DHCP Message Types

Along with DHCPDISCOVER, DHCPOFFER, DHCPREQUEST and DHCPACK, RFC 2131 describes few other DHCP messages also. These messages are as follows.

**1. DHCPINFORM:** This message is sent by a client to the server if the client is configured with an IP address manually and it requires only network configuration parameters.

**2. DHCPRELEASE:** If a DHCP client wants to voluntarily relinquish the IP address offered to it by the DHCP server and cancel the remaining lease, it sends a DHCPRELEASE message to a server.

**3. DHCPNACK:** A DHCP server sends a DHCPNACK message to the client if it is unable to acknowledge a DHCPREQUEST message sent by a client. This case arises, for example, when the client has moved to a new subnet or the client's lease time has expired while it is renewing its IP address.

### 3.2.3 DHCP Message Format

Different types of DHCP messages described earlier use a common message format. This message format is shown in Figure 3.2. Specific values set in various fields of this message identify the type of message. The fields present in this message format and their description are shown in Table 3.1.

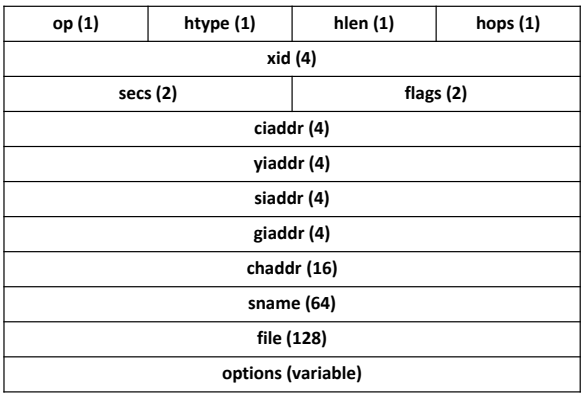



Figure 3.2: DHCP message format

Table 3.1: DHCP header fields

| Notation | Description |
|---|---|
| op | Type of DHCP message |
| htype | Type of hardware address |
| hlen | Hardware address length |
| hops | Used by routers acting as relay agents to forward DHCP messages in case a DHCP server and a client are present in different subnets |
| xid | Random number to associate DHCP requests and responses between the client and the server |
| secs | Seconds elapsed since client started the address acquisition or renewal process |
| flags | Two flags - *BROADCAST* and *Must Be Zero (MBZ)*. Setting *Broadcast* flag=1 represents client can not receive unicast packets |
| ciaddr | Client's IP address if it is in RENEW state, else set to '0.0.0.0' |
| yiaddr | The IP address DHCP server has offered to client |
| siaddr | Server's IP address in DHCPOFFER and DHCPACK messages |
| giaddr | Relay agent's IP address if present in the network, else set to '0.0.0.0' |
| chaddr | Client's hardware address |
| sname | Optional field represents server's host name |
| file | Optional field represents the name of a boot file |
| options | Contains configuration parameters such as subnet mask and default gateway's IP address |

## 3.3 Prior Work

As described previously, DHCP does not have any built-in authentication scheme due to which it is vulnerable to classical DHCP starvation attack wherein a malicious client prevents legitimate client(s) in the network from obtaining an IP address from a DHCP server. Since the clients are not able to obtain the IP address, they can not communicate with other devices which leads to DoS. There are two methods of launching classical starvation attack - 1) *Type-1: Using same MAC address in Ethernet header and DHCP header* and 2) *Type-2: Using different MAC address in*

*Ethernet header and DHCP header* as follows.

**I. Using Same MAC Address in Ethernet Header and DHCP Header (Type-1):** To launch classical DHCP starvation attack using this method, a malicious client sends large number of DHCPDISCOVER messages having randomly generated MAC address in the *chaddr* field of DHCP header. The same (randomly generated) MAC address is also used as the source MAC address of the Ethernet frames carrying these DHCPDISCOVER messages as shown in Figure 3.3. The

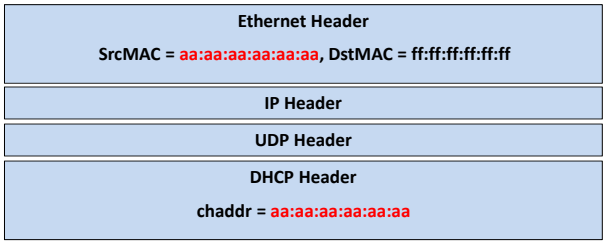



Figure 3.3: DHCPDISCOVER message having randomly generated MAC address in DHCP and Ethernet header

server allocates IP address to each of the MAC addresses using *DORA* process. This leads to the exhaustion of IP address pool present at the DHCP server. As a result, server is not able to offer IP address to new legitimate clients. However, in wireless networks secured with Wired Protection Access (WPA2) [73], this attack can not be easily generated. IEEE 802.11 specification mandates a wireless client to associate itself with an access point. As part of this association, a session key is generated. One of the input parameter for this session key generation is client's MAC address. This association and session key generation is done using a 4-way handshake scheme as shown in Figure 3.4 and is explained below:

**4-way Handshake in WPA2:** 4-way handshake is used to convey that both endpoints (access point and wireless client) know the Pairwise Master Key (PMK). Instead of disclosing the key, the access point and wireless client send encrypted messages to each other such that the messages can only be decrypted using the already shared PMK. If decryption of the messages is successful, this shows that both

wireless client and access point have knowledge of the PMK. The PMK remains valid unless the established session is expired. Moreover, the PMK should be exposed as little as possible, thus, there should be a new key that is to be derived in order to encrypt the traffic.

This 4-way handshake scheme is also used to establish another key called the Pairwise Transient Key (PTK). The PTK is generated by concatenating the following attributes: PMK, ANonce, SNonce, access point's MAC address and wireless client's MAC address. The product is then put through a pseudo-random function. The 4-way handshake also yields the Group Temporal Key (GTK) which is used to decrypt multicast and broadcast traffic. The set of messages exchanged during the 4-way handshake are shown in Figure 3.4.

**1. ANonce by Access Point:** The access point sends a nonce value ANonce to the

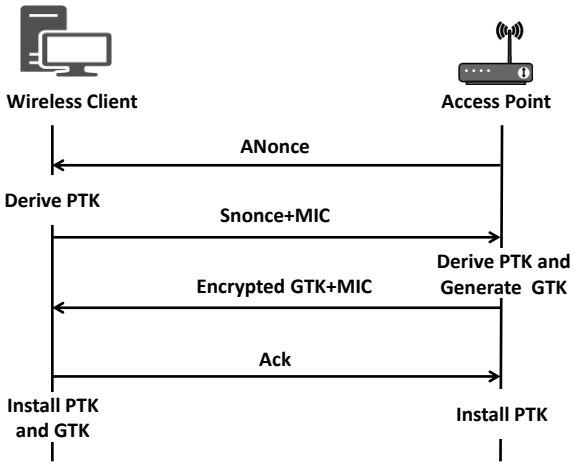



Figure 3.4: Association of client with AP

wireless client along with a Key Replay Counter, which is a number that is used to match each pair of messages sent and discard replayed messages.

**2. SNonce and MIC by Wireless Client:** The wireless client sends its own nonce value SNonce to the access point along with a Message Integrity Code (MIC) and the Key Replay Counter same as the one sent by access point in Step1 to allow access point to match the right message.

**3. GTK and MIC by Access Point:** The access point verifies the previous message

sent by the wireless client by checking MIC, ANonce and Key Replay Counter Field and if the message is valid, the access point constructs and sends the GTK to the wireless client with another MIC.

**4. Confirmation by Wireless Client:** The wireless client verifies the previous message sent by the access point by checking MIC and Key Replay Counter Field and if the message is valid, the wireless client sends a confirmation to the access point. After this, both wireless client and access point can start using the derived keys for secure communication.

As the PTK shared between the wireless client and the access point is derived using the MAC address of wireless client and access point, the access point drops Ethernet frames having random source MAC addresses as these MAC addresses are not associated with the access point and it does not have any PTK corresponding to the randomly generated MAC addresses. As a result, this method of launching the classical DHCP starvation attack does not work in wireless networks unless the malicious client precedes an association of each spoofed MAC address with the access point. However, considering the association phase and the session key generation phase using 4-way handshake, it is tedious for the malicious client to perform these operations for every randomly generated MAC address. Moreover, an access point has restriction of allowing only limited number of MAC address associations with it at a time. This also prevents address pool exhaustion by the malicious client if the number of IP addresses in the DHCP server pool is greater than the maximum number of associations permitted by the access point.

**II. Using Different MAC Address in Ethernet Header and DHCP Header (Type-2):** To launch classical DHCP starvation attack using this method, a malicious client sends a large number of DHCPDISCOVER messages having randomly generated MAC address in its *chaddr* field but its own MAC address as the source MAC address in the Ethernet frames carrying these DHCPDISCOVER messages as shown in Figure 3.5. In a wireless network, access point accepts these DHCPDISCOVER messages and forwards it to a DHCP server because the source MAC address

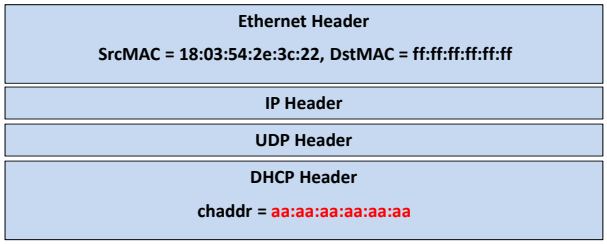



Figure 3.5: DHCPDISCOVER message having randomly generated MAC address in DHCP header only

(i.e. malicious client's MAC) present in the Ethernet frames is associated with the access point. When the DHCP server receives these DHCPDISCOVER messages, it uses the MAC address present in the *chaddr* field of the received DHCPDISCOVER messages to send unicast DHCPOFFER messages. As these MAC addresses are randomly generated by the malicious client and there was no association of these addresses with the access point, it drops the corresponding DHCPOFFER packets. Hence *DORA* process is not completed and malicious client fails to exhaust the IP address pool to create DHCP starvation.

## 3.4 Induced DHCP Starvation Attack

In this section, we describe the working of our proposed Induced DHCP starvation attack in both wired and wireless networks. This attack prevents legitimate DHCP clients from obtaining IP address by exploiting the IP conflict detection scheme present in both client and server sides. This attack requires fewer messages and easier to launch as compared to both types of classical DHCP starvation attack. Depending on which IP conflict detection scheme is exploited and using what type of probe, there are three variants of Induced DHCP starvation attack. In the next three subsections, we describe these three variants. We use the notations shown in Table 3.2 while describing the Induced DHCP starvation attack.

Table 3.2: Notations

| Notation | Description |
|---|---|
| MAC_VICTIM | MAC address of victim client |
| MAC_MALICIOUS | MAC address of malicious client |
| ARP_REQ | ARP Request |
| ICMP_REQ | ICMP Request |
| ARP_REPLY | ARP Reply |
| ICMP_REPLY | ICMP Reply |
| SrcIP | Source IP address |
| TarIP | Target IP address |
| SrcMAC | Source MAC address |
| DestMAC | Destination MAC address |
| IP_SERVER | IP Address of DHCP Server |
| MAC_SERVER | MAC Address of DHCP Server |
| IP_SELECTED | IP Address selected for offering by DHCP Server |
| IP_SELECTED_MALICIOUS | IP Address selected for offering by malicious client |

### 3.4.1 Exploiting Client's IP Conflict Detection Scheme

The first variant of proposed Induced DHCP starvation attack exploits the IP conflict checking performed by a DHCP client after it is offered an IP address from a DHCP server. To describe the procedure of launching this attack, we consider a network setup as shown in Figure 3.6. This network setup consists of 4 devices namely Access Point (AP), a DHCP server, a malicious client and a victim client which fails to acquire IP address from the DHCP server. Both malicious client and victim client are connected to the AP using their wireless interface and the DHCP server is connected to the AP through wired connection. The sequence of messages exchanged while launching the attack is as follows.

**1. DHCPDISCOVER by Victim Client:** When the victim client comes up, it associates itself with AP using the 4-way handshake mechanism and sends a broadcast DHCPDISCOVER message to find the DHCP server(s) in the network which can lease

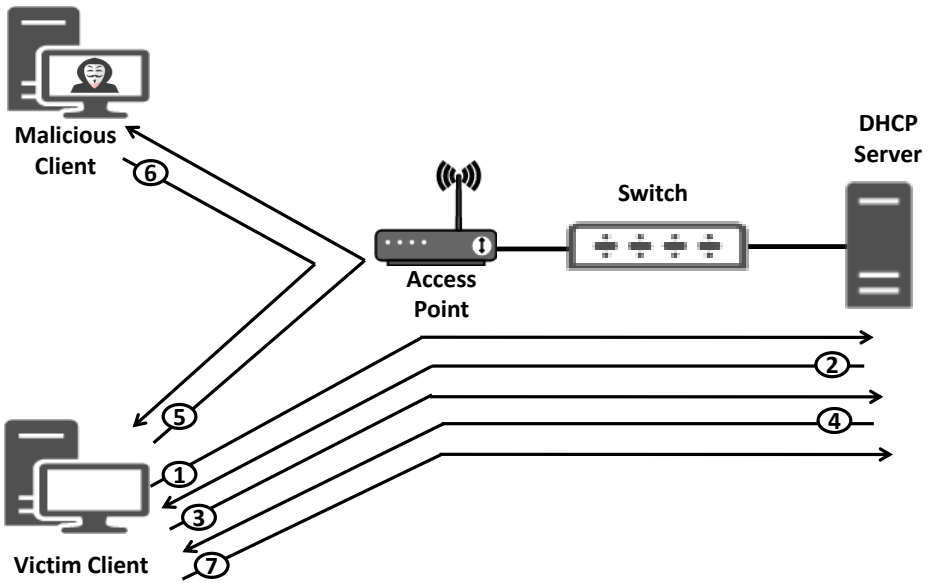



Figure 3.6: Exploiting client-side IP conflict detection scheme

an IP address.

**2. DHCPOFFER by Server:** The DHCP server replies with a unicast DHCPOFFER message in response to victim client's DHCPDISCOVER.

**3. DHCPREQUEST by Victim Client:** Victim client sends a broadcast DHCPREQUEST message to the server to request the IP address offered to it in previous stage. The *chaddr* field of this DHCPREQUEST message has the MAC address of victim i.e. MAC_VICTIM. As this message is a broadcast message, the malicious client also receives this message and it takes note of MAC_VICTIM.

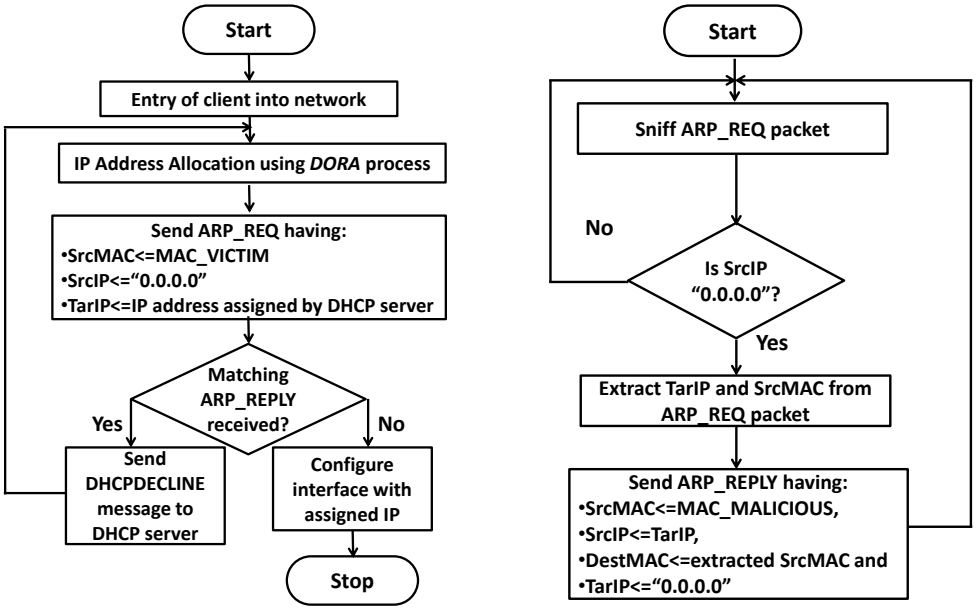
**4. DHCPACK by Server:** The server sends a unicast DHCPACK message to the victim client to confirm the allocation of IP address to it.

**5. ARP_REQ by Victim Client:** Victim client sends an ARP_REQ broadcast packet to check if any other client in the network is using the IP address offered by server. The source IP address of ARP_REQ is set to "0.0.0.0". This address is used as the victim client does not have a configured IP address yet.

**6. ARP_REPLY by Malicious Client:** Malicious client sniffs ARP_REQ sent by victim with a filter set to source IP address "0.0.0.0". Malicious client replies with a fake ARP_REPLY with the IP address allocated to victim as source IP address and its

own MAC address MAC_MALICIOUS as source MAC address. The target IP address of this message is set to "0.0.0.0" and target MAC address to MAC_VICTIM.

**7. DHCPDECLINE by Victim Client:** As soon as the victim client receives ARP_REPLY sent by the malicious client in previous step, it interprets this message as some other client is already using the IP address offered to it [74]. In order to avoid IP conflict, victim client sends a DHCPDECLINE message to server denying the use of offered IP address.




(a) (b)

Figure 3.7: Variant-1 from (a) victim client's and (b) malicious client's perspective

Once DHCP server receives this DHCPDECLINE message, it marks the IP address as unavailable for one lease period [15]. Victim client re-initiates the process of acquiring a new IP address again and set of operations are repeated; effectively denying the victim client from acquiring an address. This ensures that the victim client is starved forever. Figures 3.7a and 3.7b summarize the sequence of operations described earlier from victim and malicious client's perspective respectively while launching the Induced DHCP starvation attack by exploiting client-side IP conflict detection scheme.

### 3.4.2 Exploiting Server's IP Address Conflict Detection Scheme:

The second variant of proposed Induced DHCP starvation attack exploits the IP conflict checking performed by a DHCP server before it offers an IP address to a client. To describe the procedure of launching this attack, we consider a network setup as shown in Figure 3.8. The sequence of messages exchanged while launching the Induced

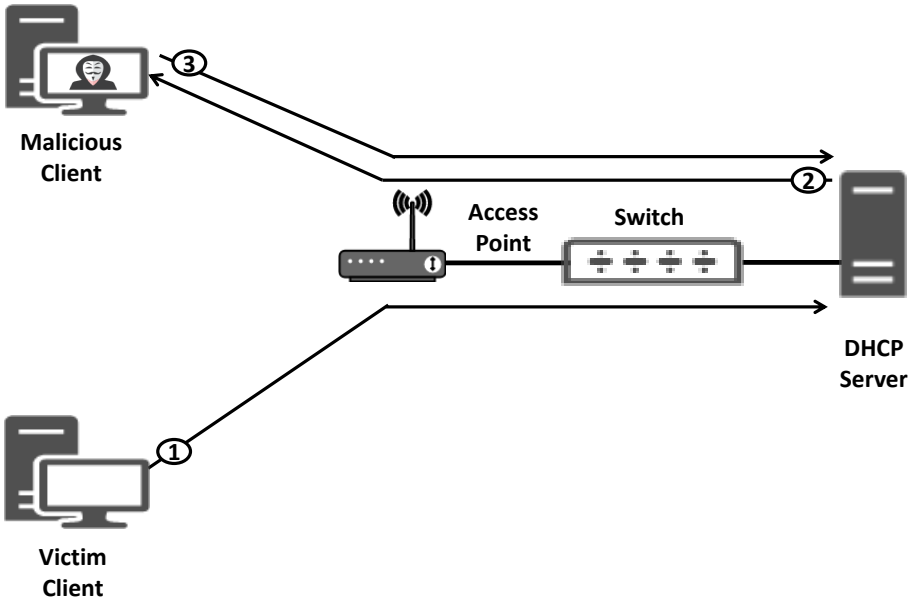



Figure 3.8: Exploiting server-side IP conflict detection scheme

DHCP starvation attack by exploiting server-side IP conflict detection scheme is as follows.

**1. DHCPDISCOVER message by Victim Client:** As soon as victim client connects to the network, it sends a DHCPDISCOVER message to find the available DHCP server(s) in the network.

**2. Probe Request by DHCP Server:** After receiving the DHCPDISCOVER message sent by the victim client in previous step, a DHCP server selects one IP address IP_SELECTED from its pool and sends a broadcast probe request to check if this IP address is in use by some other client. Depending on the DHCP's software implementation, server uses either ARP_REQ or ICMP_REQ for the probing purpose.

Table 3.3 shows few sample DHCP servers and the probe types they use to detect IP address conflicts. However, it should be noted that even if ICMP_REQ probe is used, ARP_REQ is automatically generated by the operating system of the DHCP server to get the MAC address of the client which is using IP_SELECTED.

Table 3.3: Few DHCP servers and the probe types

| **Vendor** | **Probe Type** |
|---|---|
| ISC DHCP server | ICMP Ping Request |
| Microsoft Windows Server 2008 | ICMP Ping Request |
| Netgear N150 Wireless Router (inbuilt DHCP) | ARP Request |
| D-Link DIR-600M N150 Wireless Router (built-in DHCP) | ARP Request |

- **ARP_REQ as a Probe:** If server sends ARP_REQ as probe; the source IP, source MAC, target IP and destination MAC addresses of this probe are set to IP_SERVER, MAC_SERVER, IP_SELECTED and "ff:ff:ff:ff:ff:ff" respectively.

- **ICMP_REQ as a probe:** If server selects ICMP_REQ as probe; it crafts an ICMP_REQ with source IP and destination IP as IP_SERVER and IP_SELECTED respectively.

**3. Fake Probe Reply by Malicious Client:** Malicious client sniffs probe sent by server with a filter set on source IP address, IP_SERVER, and then injects corresponding fake probe reply.

- **ARP_REPLY as a Probe Reply:** Corresponding to ARP_REQ sent by DHCP server as probe, the malicious client sends a fake ARP_REPLY by setting the source IP, source MAC, target IP and destination MAC addresses to IP_SELECTED, MAC_MALICIOUS, IP_SERVER and MAC_SERVER respectively.

- **ICMP_REPLY as a Probe Reply:** If the malicious client captures the ICMP_REQ probe, it sends back a fake ICMP_REPLY to the server having

its source IP and destination IP set to IP_SELECTED and IP_SERVER respectively.

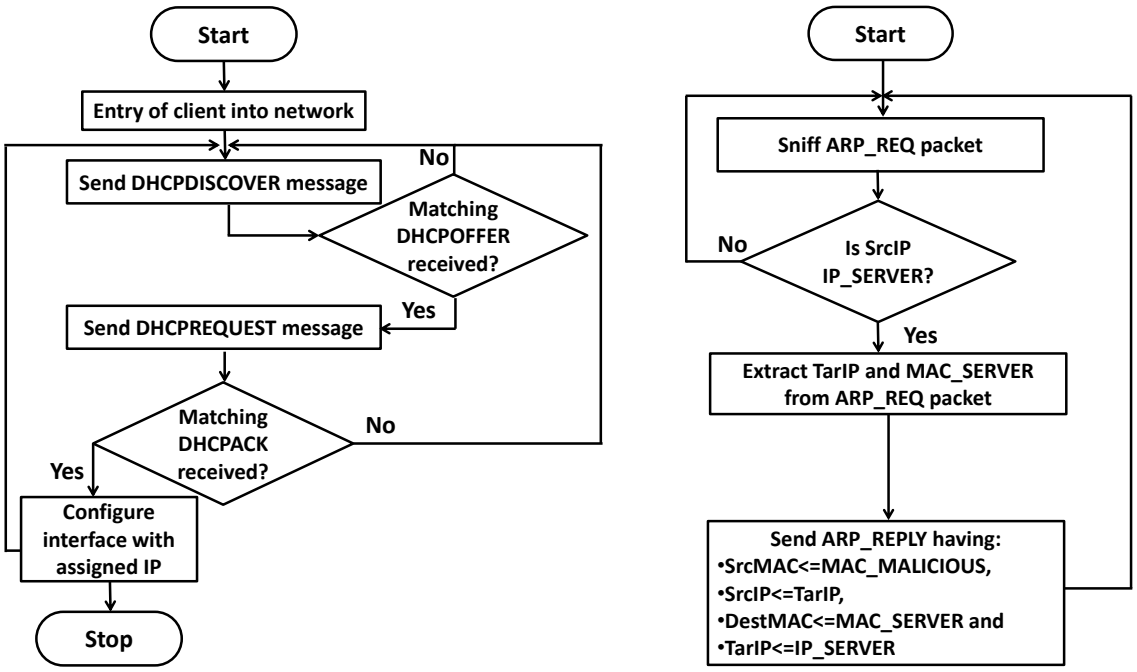



Figure 3.9: Variant-2 from (a) victim client's and (b) malicious client's perspective

Once DHCP server receives ARP_REPLY/ICMP_REPLY, it interprets that some other client is already using IP_SELECTED. As a result, server marks this IP address unavailable for one lease period and does not offer it to the victim client. Victim client re-initiates the process of acquiring a new IP address again and these set of operations are repeated effectively denying the victim client from ever acquiring an IP address. Figures 3.9a and 3.9b[1] summarize the sequence of operations involved while launching the Induced DHCP starvation attack by exploiting server-side IP conflict detection scheme from victim and malicious client's perspective respectively.

[1]For simpler representation, we show the case in which DHCP server uses ARP request as a probe.

### 3.4.3 Exploiting Client's IP Conflict Detection Scheme using Spoofed Probe Requests:

Similar to the first variant, the third variant of proposed Induced DHCP starvation attack also exploits the client's IP conflict detection scheme. To describe the procedure of launching this attack, we consider a network setup as shown in Figure 3.10. The steps involved in launching this variant of the proposed attack are as follows:

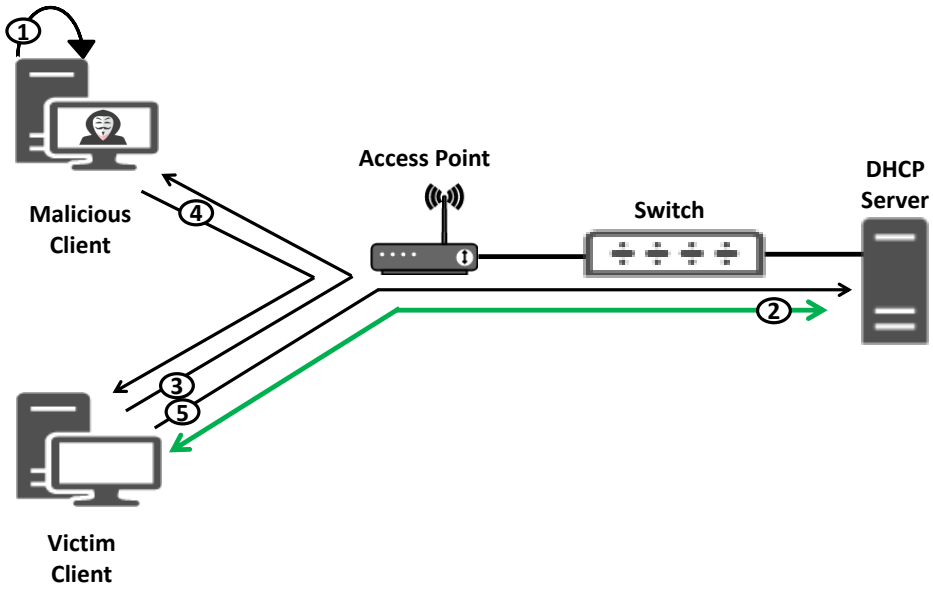
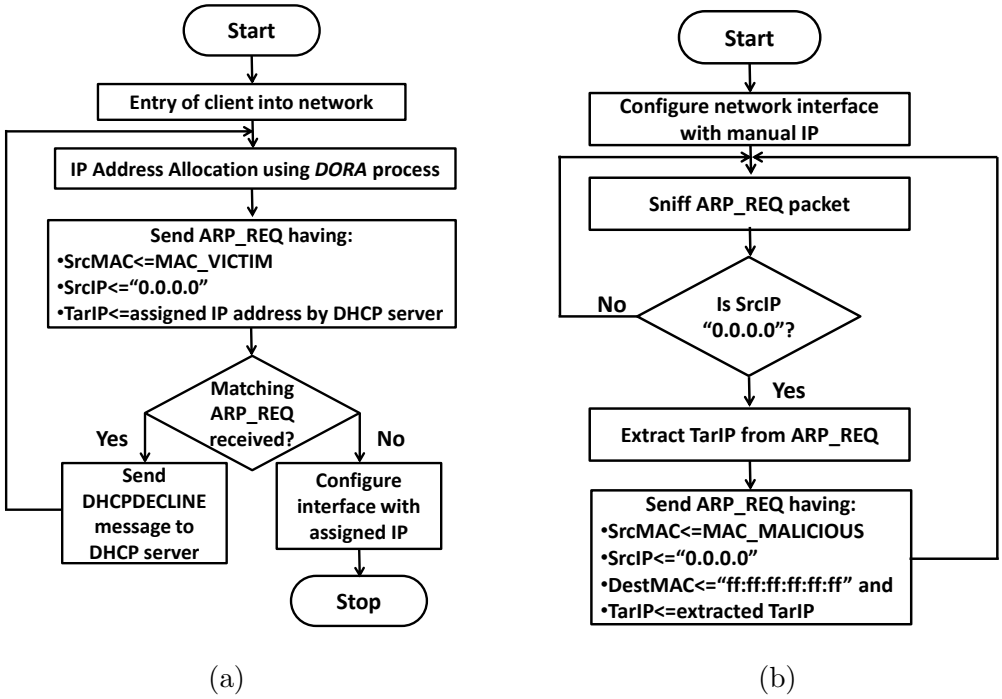



Figure 3.10: Exploiting client-side IP conflict detection scheme using spoofed probe requests

**1. Manual IP Address Configuration by Malicious Client:** As soon as malicious client joins the network, it disables its DHCP daemon running in the background and manually configures its interface with an IP address and other configuration parameters. The malicious client does so to bypass security features present in modern network switches such as Dynamic ARP Inspection (DAI) [25] discussed in Chapter 4.

**2. Address Allocation to Victim Client using *DORA* Process:** Victim client joins the network and acquires IP address from DHCP server using *DORA* process. After successful allocation, DHCP server adds an entry for a binding between offered IP and victim client's MAC address in its database.

**3. ARP_REQ Broadcast by Victim Client:** The victim client broadcasts an




(a) (b)

Figure 3.11: Variant-3 from (a) victim client's and (b) malicious client's perspective

ARP_REQ to check if the offered IP address is already in use. The source IP address of this ARP_REQ is "0.0.0.0" as victim client does not configure its network interface with the offered IP prior to performing conflict check.

**4. ARP_REQ Broadcast by Malicious Client:** As soon as malicious client receives ARP_REQ sent in step 3, it also broadcasts an ARP_REQ. The source IP address of this probe is set to "0.0.0.0", target IP address is set to IP address in question, target MAC address is set to "00:00:00:00:00:00", destination MAC in Ethernet header is set to "ff:ff:ff:ff:ff:ff" while the source MAC address in Ethernet header is set to MAC_MALICIOUS.

**5. DHCPDECLINE Message Broadcast by Victim Client:** Once the victim client receives the ARP probe request sent by malicious client in step 4, it declines the assigned IP address by broadcasting a DHCPDECLINE message. DHCP server, on receiving this message, marks the IP address in question as unavailable for one lease period. Victim client reinitiates the process of acquiring a new IP address and set of

operations are repeated; effectively preventing the victim client from ever acquiring an address. This leads to DoS scenario. Figures 3.11a and 3.11b summarize the sequence of operations involved while launching the Induced DHCP starvation attack by exploiting client-side IP conflict detection scheme using spoofed probe requests from victim and malicious client's perspective respectively.

### 3.4.4 Comparison of Induced DHCP Starvation Attack with Classical DHCP Starvation Attack

In this subsection, we compare Induced DHCP starvation attack with previously known classical DHCP starvation attack.

**1. Attack Launching Technique:** On the one hand where classical DHCP starvation attack requires sending multiple DHCPDISCOVER and DHCPREQUEST messages to consume IP address pool, Induced DHCP starvation attack requires sending spoofed probe requests or replies to consume the IP address pool.

**2. Attack's Effectiveness in Wired and Wireless Networks:** As described earlier in Section 3.3, though classical DHCP starvation attack is effective in wired networks, it is ineffective in wireless networks unless the malicious client precedes an association with AP for every spoofed MAC address which is quite tedious. Even if fake associations are preceded, classical starvation attack again fails to exhaust IP address pool in wireless networks due to restrictions on number of MAC address associations enforced by AP. On the other hand, since Induced DHCP starvation attack neither requires spoofed MAC addresses nor does it require fake associations of these MAC addresses with AP, they are quite easy and effective in both wired and wireless networks.

**3. Traffic Overhead due to Attack:** In case of classical DHCP starvation attack, two DHCP messages (one DHCPDISCOVER and one DHCPREQUEST) are required to consume one IP address from the pool. However, the number of messages required to launch Induced DHCP starvation attack by exploiting client's IP conflict detection scheme is only one, i.e. a fake probe request or reply. In order to launch Induced

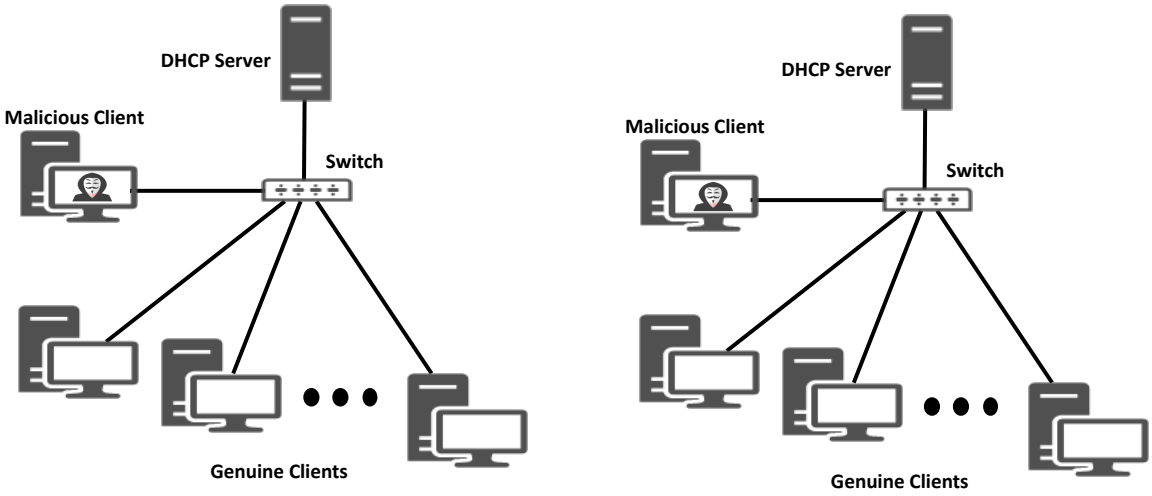
DHCP starvation attack by exploiting server's IP conflict detection scheme, a malicious client requires sending either one or two fake probe replies depending on whether the server is using ARP or ICMP probes respectively for conflict detection. For example, ISC DHCP server [75] uses ICMP requests for IP address conflict detection. In that case, the malicious client requires sending two fake probe replies (one ARP reply followed by one ICMP reply) to consume one IP address. If we consider most of the built-in DHCP softwares available with different wireless access points, they use ARP request for the conflict detection purpose. So in this case, the malicious client needs to send just one fake probe reply to consume one IP address.

## 3.5 Experiments and Discussions

In this section, we describe the experiments performed to test the proposed Induced DHCP starvation attack. We also compare the classical DHCP starvation attack with Induced DHCP starvation attack in terms of traffic overhead and discuss the rate with which IP address pool is exhausted when Induced DHCP starvation attack is launched.

**Testbed Setup:** To test the proposed Induced DHCP starvation attack, we created two testbeds having network topologies as shown in Figures 3.12a and 3.12b. The first testbed was having a switch (Cisco WS-C2950T) to which an ISC DHCP server and a wireless access point (Netgear N150 Wireless Router) were connected. This setup was also having one malicious client and 33 clients such that at a time, either only one or all of the 33 clients were designated as victim clients. All these clients were connected to the access point using their wireless network interface. In the second testbed, DHCP server, malicious client and 42 clients were connected to a switch using their wired network interface. In this testbed also, either one or all of the 42 clients were designated as victim clients at a time. The DHCP server in both the testbeds was configured to offer 256 IP addresses[2] in the range of 10.200.1.0

[2]Among 256 , one each was allotted to malicious client and server itself. Other two IP addresses, 10.200.1.0 and 10.200.1.255, were Network and Broadcast address respectively and were not used.




(a) (b)

Figure 3.12: Testbed setup: (a) Wireless topology (b) Wired topology

to 10.200.1.255. In both the testbeds, the malicious client was running Kali Linux (version 1.1.0a) operating system while victim client(s) were running Windows 7 service pack 1 operating system.

### 3.5.1 Experiments with Variant-1

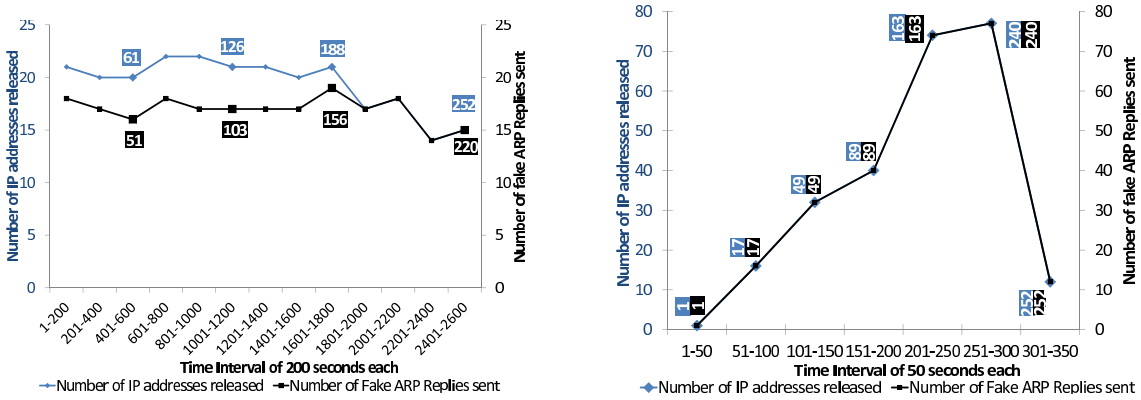
Using the testbeds, we tested the first variant of Induced DHCP starvation attack in which a malicious client exploits the client's IP conflict detection scheme in both wireless and wired networks. To sniff ARP requests having source IP address "0.0.0.0" and to send corresponding fake ARP replies from the malicious client, a C program was written. Sniffing the ARP requests and sending corresponding replies were implemented using two threads. First thread did the sniffing part and other thread generated fake ARP replies. The procedure of launching the first variant of Induced DHCP starvation attack in wireless and wired networks is as follows.

**I. In Wireless Networks:** In this experiment, we first checked the maximum number of wireless clients that could connect to our access point at a time. For this purpose, we kept on connecting one client at a time to the access point. From this

experiment, we found that the access point allowed 34 clients to associate with it. The 35th association request was rejected and client displayed a *"Can't connect to this network"* message. Subsequently, we used 34 machines (including malicious and victim clients) in our first testbed as shown in Figure 3.12a to launch the first variant of Induced DHCP starvation attack. We created two scenarios such that in the first scenario, there was only one victim client that was prevented from getting an IP address while in the second scenario, there were multiple victim clients. These two scenarios are as follows.

**A. First Scenario:** In this scenario, there was only one victim client which was prevented from acquiring an IP address. This was done by sending fake ARP replies only to this client's IP conflict detection probes. Remaining computers acted as genuine hosts requesting for IP address in between and they did so successfully. Figure 3.13a shows the number of IP addresses (blue line) released and the number of fake ARP replies (black line) that malicious client sent over time. The left y-axis represents the number of addresses released and the right y-axis represents the number of fake ARP replies sent in a time interval of 200 seconds. The numbers in blue and black boxes show the total number of IP addresses released[3] and total number of ARP replies sent by malicious client up to that interval (cumulative) respectively. We can notice that, in approximately 2600 seconds, all the IP addresses in the DHCP server pool were released. We can also notice that after around 2000 seconds, these two lines (blue and black) merge. This was because upto this point, all the 32 genuine clients had already acquired IP addresses successfully as malicious client did not reply to these clients' IP conflict detection probes. Thus there is a difference of 32 between the number in blue and black box at this point. However, victim client had not got the IP address yet and it kept on attempting to acquire an IP address but in each attempt, malicious client prevented the victim client from acquiring an IP address by sending fake probe replies. Hence beyond this point, IP addresses were only marked as unusable (but released) and was equal to number of

[3]Number of IP addresses released is the sum of victim client's attempt to acquire an IP address and other clients' successful attempt




(a) (b)

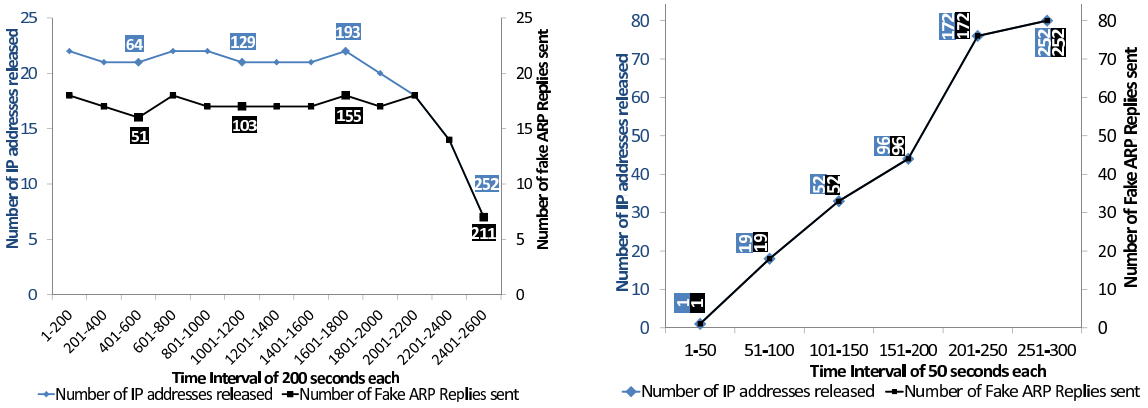
Figure 3.13: IP address allocation and fake ARP replies sent over time with (a) single victim (b) multiple victim clients in wireless network

ARP replies sent by the malicious client.

**B. Second Scenario:** In the second scenario, malicious client prevented every other client from acquiring an IP address (every client was a victim). Figure 3.13b shows the number of IP addresses released (blue line) and the number of fake ARP replies (black line) that malicious client sent over time. We can notice that in approximately 350 seconds, all the IP addresses were released. Compared to the first scenario, IP address pool was exhausted much quickly. This was because all the hosts were simultaneously trying to acquire IP address and every attempt failed and DHCP server marked one IP address unusable after receivng one DHCPDECLINE message. We can also notice that the number of IP addresses released and number of ARP replies sent by malicious client were equal throughout. This was because the malicious client was sending fake ARP replies for each client's IP conflict detection probes and there were no successful attempts of IP address acquisition.

**II. In Wired Networks:** For this experiment, we used 43 clients (including malicious and victim clients) and a DHCP server in our second testbed as shown in Figure 3.12b. In this case also, we experimented with one victim and multiple (all the 43 clients) victims scenarios. Figure 3.14a and Figure 3.14b shows the number of

IP addresses released/number of ARP replies sent over time in both the scenarios. We can notice that the rate of allocation is almost similar to previous case. We can



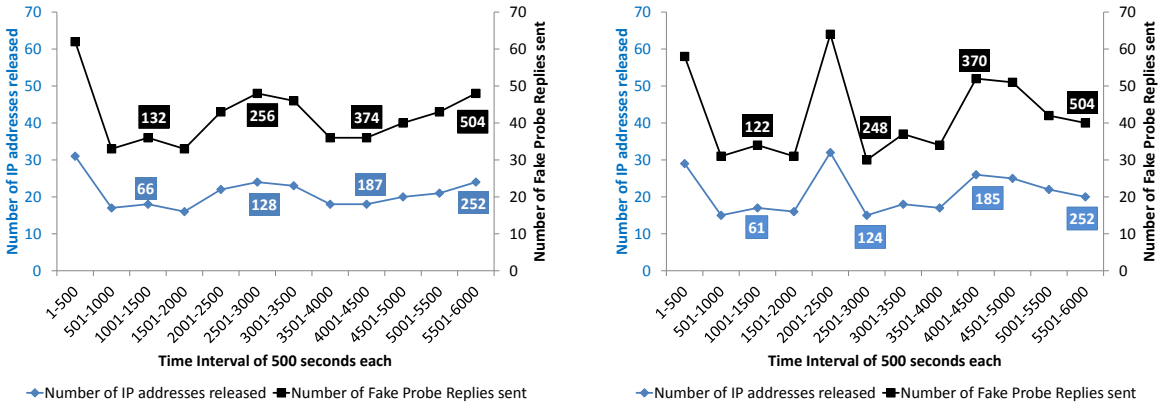

(a) (b)

Figure 3.14: IP address allocation and fake ARP replies sent over time with (a) single victim (b) multiple victim clients in wired network

also notice that it took around 2600 seconds and 300 seconds to exhaust the address pool in one victim and multiple victims scenarios respectively.

### 3.5.2 Experiments with Variant-2

We also tested the second variant of Induced DHCP starvation attack in our wireless and wired testbeds. ISC DHCP server configured in our testbeds sent ICMP echo requests as probes to detect IP address conflicts in the network. To exploit this server's conflict detection scheme, our designated malicious client sniffed the probes and sent corresponding fake probe replies using a C program. In both the testbeds, we designated only one victim client that was prevented from obtaining an IP address from the server. Figures 3.15a and 3.15b show the number of IP addresses released (blue line) and the number of fake probe replies (black line) that malicious client sent over time in order to target the victim client in wireless and wired network respectively. We can notice that, in approximately 6000 seconds, all the IP addresses were released in case of both wireless and wired networks. We can also notice that the number of fake probe replies sent by the malicious client is twice the number of consumed IP



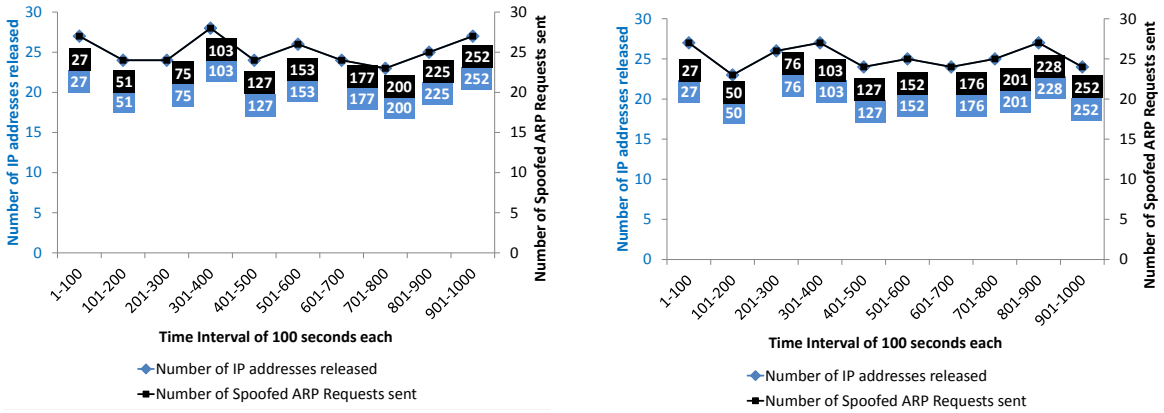

(a) (b)

Figure 3.15: IP address allocation and fake probe replies sent over time in (a) wireless and (b) wired network in case of one victim client

addresses in both wireless and wired networks. This was because, in order to consume one IP address, the malicious client needed to send two fake probe replies (1 fake ARP and 1 fake ICMP reply). Thus, a total of 504 fake probe replies were sent to consume 252 IP addresses in case of both wireless and wired networks.

### 3.5.3 Experiments with Variant-3

We also tested the third variant of proposed Induced DHCP starvation attack in our wireless and wired testbeds. However, we used only four computers in both the testbeds for this experiment such that one computer was designated as malicious client while remaining other three computers were designated as victim clients. Moreover, we configured the malicious client manually with IP address '10.200.1.2'. To sniff ARP requests having source IP address "0.0.0.0" and to send corresponding spoofed ARP requests from the malicious client, a C program was written. Figures 3.16a and 3.16b show the number of IP addresses released (blue line) and the number of spoofed ARP requests (black line) that malicious client sent over time in order to target all the three victim clients in wireless and wired network respectively. We can notice that, in approximately 1000 seconds, all the IP addresses were released in case of both wireless and wired networks. We can also notice that the number of spoofed ARP requests




(a) (b)

Figure 3.16: IP address allocation and spoofed ARP requests sent over time in (a) wireless and (b) wired network in case of three victim clients

sent by the malicious client is equal to the number of consumed IP addresses in both wireless and wired networks. This was because, in order to consume one IP address, the malicious client needed to send only one spoofed ARP request. Thus, a total of 252 spoofed ARP requests were sent to consume 252 IP addresses in both wireless and wired networks.

### 3.5.4 Discussion

As discussed earlier in Section 3.4, Induced DHCP starvation attack exhausts the IP address pool present at the DHCP server by exploiting IP conflict detection schemes present in both client and server sides. In this subsection, we describe the rate with which the attack can exhaust the IP address pool and also compare the traffic generated while launching the classical and Induced DHCP starvation attacks.

**I. Rate of IP Address Pool Exhaustion:** In case of Induced DHCP starvation attack, the rate of pool exhaustion depends upon the rate with which new clients join the network. The reason is malicious client injects probe requests or replies only when it receives corresponding probe requests sent by client or server during IP address allocation process. If the rate with which clients join the network

increases, the rate of occurrence of *DORA* processes and probe generation also increases that leads to injection of more number of fake probe requests or replies. This eventually causes rapid pool exhaustion.

**II. Comparing Traffic Overhead in Induced and Classical DHCP Starvation Attacks:** In order to compare the overhead on malicious client to launch both Induced and classical DHCP starvation attack, we generated all the three Induced DHCP starvation attack variants and classical DHCP starvation attack in a wired network and calculated the number of messages sent by a malicious client in case of a single victim and multiple victims scenarios. We do not present the overhead comparison in wireless network because of the difficulty of creating classical starvation attack in wireless networks.

**A. Single Victim:** To consume one IP address, the first and third variant of Induced DHCP starvation attack requires sending only one fake ARP message while the second variant requires sending either one (ARP message) or two (one ARP and one ICMP) fake messages depending on the type of probe used by DHCP server for conflict detection. On the other hand, classical DHCP starvation attack requires sending two (one DHCPDISCOVER and one DHCPREQUEST) messages to consume one IP address. As a result, in the single victim scenario of our experiments discussed earlier in Section 3.5.1, first and third variant of Induced DHCP starvation attack requires sending 211 fake ARP messages while the second Induced DHCP starvation attack variant and classical DHCP starvation attack requires sending 422 messages to consume pool of 256 IP addresses available at the ISC DHCP server configured in our testbed setup.

**B. Multiple Victims:** We compared different Induced DHCP starvation attack variants and classical DHCP starvation attack in terms of the number of messages required to consume address pool in case of multiple victims scenario also. The total number of messages sent by malicious client to exhaust 256 IP addresses was 252 in case of first and third variant of Induced DHCP starvation attack and 504 in case of second variant of Induced DHCP starvation attack and classical DHCP starvation

attack.

Table 3.4 summarizes the comparison of Induced and classical DHCP starvation attack in terms of traffic overhead.

Table 3.4: Induced v/s classical DHCP starvation attack

| **Scenario** | **Variant-1** | **Variant-2** | **Variant-3** | **Classical DHCP Starvation Attack** |
|---|---|---|---|---|
| Messages required to consume 1 IP | 1 | 2 | 1 | 2 |
| Single Victim | 211 | 422 | 211 | 422 |
| Multiple Victims | 252 | 504 | 252 | 504 |

## 3.6 DNS Spoofing in Local Networks: An Implication of Induced DHCP Starvation Attack

Domain Name System (DNS) [9] is one of the critical protocols of the Internet as it is responsible for the efficient working of various applications. It provides a mapping between domain name and its corresponding IP address, thus, mitigating the need of remembering IP addresses of hosts present on the Internet. DNS is also said to be a database of resource records as it maintains not only the host name to IP mapping but also other records such as nameserver, mail exchanger, etc. In this section, we present an implication of proposed Induced DHCP starvation attack in the form of a targeted DNS spoofing attack in local networks. To launch this attack, a malicious client first exploits the IP conflict detection scheme of a DHCP server using the second variant of Induced DHCP starvation attack. As a result, a victim client is not able to obtain IP address and other network configuration parameters such as DNS server's and default gateway's IP address. After this, malicious client immediately offers a genuine IP address to the victim client and its own IP address as the default gateway's and DNS server's IP address. The victim client ultimately starts using these parameters for

further communication. This allows malicious client to alter or redirect the victim client's traffic as and when required. In next few subsections, we describe the targeted DNS spoofing attack and the experiments performed to test this attack.

### 3.6.1 Proposed Targeted DNS Spoofing Attack

To describe the working of proposed attack, we consider a network topology similar to the one shown in Figure 3.17. In this network, there are four devices namely a DHCP server, a malicious client, a victim client and a switch connecting all these devices. The notations used to describe the attack steps below are shown in Table 3.2.

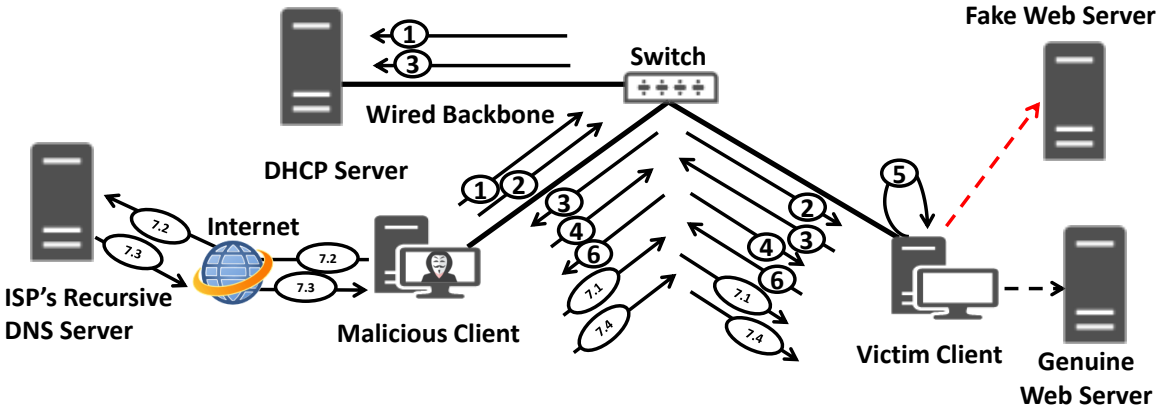



Figure 3.17: Attack description

**1. Exploiting DHCP Server's IP Conflict Detection Scheme:** As soon as a DHCP server receives a DHCPDISCOVER message sent by a victim client to obtain an IP address, DHCP server sends a broadcast probe to check if an IP address IP_SELECTED it chose to offer to the victim client is not being used by any other client in the network. On receiving this broadcast probe, a malicious client responds back with a fake probe reply. Due to this reply, DHCP server is not able to offer IP_SELECTED to the victim client and thus, it selects next available IP address from the pool and starts performing conflict detection for this IP. DHCP server tries to offer an IP address unless it receives the broadcast DHCPREQUEST message sent

by the victim client in $3^{rd}$ step.

**2. DHCPOFFER Message by Malicious Client:** After sending fake probe reply in previous step, malicious client immediately offers an IP address, IP_SELECTED_MALICIOUS, to the victim client by sending a DHCPOFFER message. IP_SELECTED_MALICIOUS belongs to the same IP address pool range as configured on genuine DHCP server.

**3. DHCPREQUEST Message by Victim Client:** As soon as victim client receives DHCPOFFER message, it sends a DHCPREQUEST message to the malicious client in order to request to use IP_SELECTED_MALICIOUS. Since DHCPREQUEST message is a broadcast message, genuine DHCP server also receives it. This message informs genuine DHCP server to stop trying to offer an IP address to victim client as it has already been offered an IP address from the malicious client.

**4. DHCPACK Message by Malicious Client:** In response to the DHCPREQUEST message sent by the victim client in previous step, malicious client sends a DHCPACK message confirming the allocation of IP_SELECTED_MALICIOUS to victim client. Other network configuration parameters like default gateway's and DNS server's IP address is also sent in this DHCPACK message. In order to receive traffic (including DNS queries) coming from and going to victim client, malicious client claims its own IP address as default gateway's and DNS server's IP address.

**5. IP Address and Other Network Parameters Configuration by Victim Client:** As soon as victim client receives DHCPACK message sent by malicious client in previous step, it configures its interface with IP_SELECTED_MALICIOUS and other network parameters.

**6. DNS Query by Victim Client:** To resolve a domain name, $D_N$, to corresponding IP address, victim client sends a DNS query to the malicious client and this client will have option to manipulate the replies.

**7. DNS Response by Malicious Client:** On receiving the DNS query sent by the victim client in the previous step, the malicious client can choose to either forward the query to the ISP's DNS server or simply send back a forged DNS response. If $D_N$ belongs to lucrative domains like banking or e-commerce websites, the malicious

client can send fake DNS responses (shown in Step 7.1 of Figure 3.17) to redirect the client to a fake page for capturing credentials. However, if $D_N$ belongs to other domains like search engine or educational website, the malicious client can send the query to genuine DNS server (shown in Step 7.2 of Figure 3.17) to obtain the resolved IP address (shown in Step 7.3 of Figure 3.17) and forward it to the victim client (shown in Step 7.4 of Figure 3.17) instead of sending fake response, thereby, achieving selective DNS spoofing.

**Targeting a Specific Client:** Using targeted DNS spoofing, malicious client can target a specific client also without affecting other clients. To do so, malicious client first captures the broadcast DHCPDISCOVER message sent by the client to be targeted and then captures the broadcast probe sent by DHCP server for precautionary checking of IP address usage. After this, the malicious client immediately sends back a probe response to prevent DHCP server from offering IP address to target client. Malicious client further offers IP address and other network parameters to target client, thereby, completing the attack procedure. In this way, malicious client can easily target a specific client without disturbing other clients present in the network.

### 3.6.2 Experiments with Targeted DNS Spoofing Attack

In order to demonstrate the execution of proposed attack, we created a network setup similar to the one shown in Figure 3.18. This setup was having a malicious client, few victim clients and a D-Link DIR-600M router having built-in DHCP server. The server was configured with a pool of 14 IP addresses ranging from 192.168.0.1 to 192.168.0.14. The server used ARP requests for probing in order to detect IP address conflicts. The malicious and victim clients were running Ubuntu 16.04 and Windows 7 Service Pack 1 operating systems respectively. We configured an ISC DHCP server [75] on malicious client to offer IP address and other network configuration parameters to the victim clients. The address pool on this server was also ranging from 192.168.0.1 to 192.168.0.14 so that victim client could communicate with other on-link devices

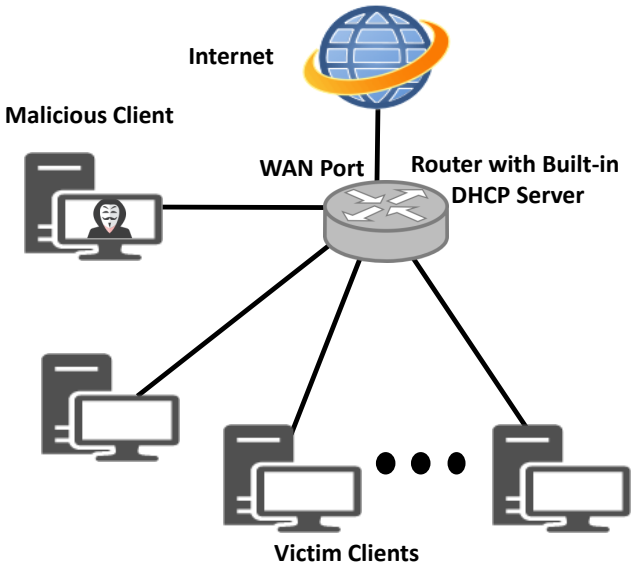



Figure 3.18: Network setup

without any issue. However, this DHCP server was configured to offer malicious client's IP address as default gateway's and DNS server's IP address to the victim clients. We executed two C programs on malicious client to launch the proposed attack. First program was used to sniff ARP requests coming from DHCP server's source IP address and also to send spoofed ARP replies. These two modules were implemented as two threads. One thread was used to sniff ARP requests while other thread generated spoofed ARP replies. Second program was used to send either fake or correct DNS reply depending on the domain type. To do so, we implemented three threads in the second program. First and second threads were used to sniff DNS queries from victim clients and send fake DNS responses respectively while third thread was used to communicate with ISP's DNS server for resolving the victim clients' DNS queries and also to send genuine DNS responses back to the victim clients.

Using this setup, we launched the proposed attack by targeting upto 5 victim clients at a time. We created 5 scenarios such that only one victim client is targeted in first scenario, two victim clients in second scenario and so on. From our experiments, we observed that DHCP server could perform maximum three attempts to offer an IP address to the victim client before it received broadcast DHCPREQUEST message

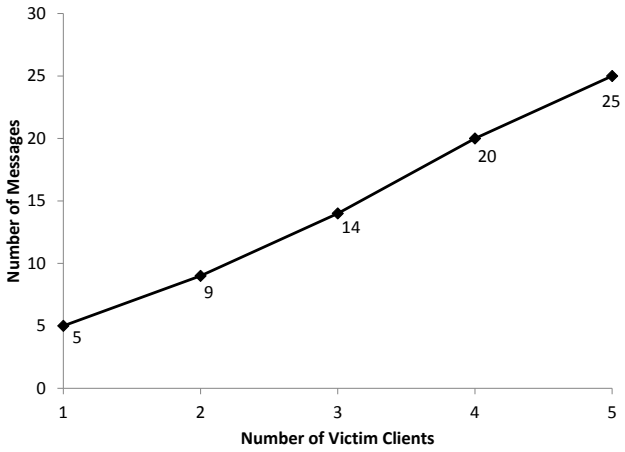



Figure 3.19: Number of messages required to target different number of clients

sent by the victim client. Thus to prevent DHCP server from offering the IP address, malicious client required to send upto three fake probe replies. Figure 3.19 shows the number of messages sent by malicious client in order to target different number of victim clients. Figure 3.20 shows the screenshot of one of the victim clients that tried to access the domain *www.iiti.ac.in*. However, it was redirected to another fake web


Figure 3.20: Victim client redirected to another web server

server configured by us on the Internet. Thus, we can notice that the proposed attack could successfully deceive victim clients by launching targeted DNS spoofing attack.

## 3.7 Conclusion

DHCP starvation attack is a popular DoS attack to prevent clients from acquiring IP address. In this chapter, we highlighted few experimental difficulties in generating classical DHCP starvation attack in wireless networks. Subsequently, we proposed Induced DHCP starvation attack that exploits IP conflict detection schemes present at DHCP clients and DHCP servers. We compared the proposed attack with classical DHCP starvation attack and showed that the proposed attack is easier to launch in wireless networks and requires less number of messages as compared to classical starvation attack. We also presented an implication of Induced DHCP starvation attack in the form of a targeted DNS spoofing attack that can alter or redirect a victim client's traffic for obvious incentives such as capturing login credentials.

Since different variants of the proposed Induced DHCP starvation attack can disrupt the services in a local network to a large extent, detecting the proposed attack is necessary. In the next chapter of this thesis, we propose appropriate anomaly detection schemes to detect these attacks.

# Chapter 4

# Detecting Induced DHCP Starvation Attack

## 4.1 Introduction

In the last chapter, we described DHCP starvation attacks which can prevent a client from obtaining IP address and also its implication in the form of targeted DNS spoofing. We also showed limitations of classical DHCP starvation attack in wireless networks. In contrast to classical DHCP starvation attack, our proposed Induced DHCP starvation attack is effective and easier to launch in both wired and wireless networks. Thus, the proposed attack must be detected using appropriate detection methods.

In this chapter, we first describe previously known defense methods to counter DHCP starvation attacks and show the ability of these methods to mitigate classical and Induced DHCP starvation attacks. Subsequently, we also propose anomaly detection systems which can detect Induced DHCP starvation attack proposed in the last chapter. Our contributions in this chapter are:

**1.** We show how previously known defense methods, though, can mitigate classical DHCP starvation attack but can not counter different variants of Induced DHCP starvation attack. We also test various security features of modern network switches and show that they also can not mitigate different variants of Induced DHCP

starvation attack.

**2.** We propose three different detection techniques to detect all variations of DHCP starvation attack.

**3.** We evaluate the detection performance of proposed scheme in a network.

Rest of the chapter is organized as follows. We describe the previously known defense methods to counter starvation attacks in Section 4.2. We present three different detection schemes in Section 4.3. The experiments performed to test the detection ability of proposed detection schemes are described in Section 4.4. Finally, we conclude the chapter in Section 4.5.

## 4.2 Prior Work

Different mitigation techniques have been proposed to counter classical DHCP starvation attack. In this section, we review their working principles and ability to detect/mitigate both classical DHCP starvation attack and proposed Induced DHCP starvation attack variants.

### 4.2.1 Security Features in Switches

Modern network switches are equipped with built-in and easy-to-configure security features such as port security [23], DHCP snooping [24] and Dynamic ARP Inspection (DAI) [25]. These security features are mostly used in tandem with each other to provide defense against variety of attacks in a local network. We discuss the working of these security features in next three paragraphs.

**1. Port Security:** Port security, implemented in modern network switches, limits the number of MAC addresses seen on a port of a network switch. Once this limit is reached, the port is automatically disabled and an SNMP trap is sent. In [36], authors discuss that DHCP starvation attack can be mitigated by limiting the number of permissible MAC addresses per port in a network switch. Since a malicious client generates multiple random MAC addresses to launch classical DHCP starvation

attack, port security can immediately disable the port as soon as more than one MAC address is detected at the port to which malicious client is connected. Thus, port security can easily mitigate Type-1 classical DHCP starvation attack where malicious client sets the source MAC address of Ethernet frames carrying DHCPDISCOVER messages to the randomly generated MAC address (see Figure 3.3 in Chapter 3). However, Type-2 classical DHCP starvation attack can not be mitigated using port security feature because malicious client uses its real MAC address in Ethernet frames carrying DHCPDISCOVER messages and random MAC address is present only in DHCP header (see Figure 3.5 in Chapter 3). Since port security does not analyze application layer protocols' headers, it is unable to mitigate Type-2 classical DHCP starvation attack. As Induced DHCP starvation attack does not require MAC address spoofing and the malicious client uses its own MAC address for communication, port security sees only one MAC address at the port. Thus, Induced DHCP starvation attack can not be mitigated by port security.

**2. DHCP Snooping:** This feature available with network switches can notice the difference in MAC address present in the Ethernet header and *chaddr* field of a DHCPDISCOVER message. Thus, it can easily mitigate Type-2 classical DHCP starvation attack. However, since this feature does not limit the number of permissible MAC addresses per port in a network switch, it can not mitigate Type-1 classical DHCP starvation attack. Since the proposed Induced DHCP starvation attack does not manipulate the header of DHCP messages, it can not be mitigated using DHCP snooping also.

**3. DAI with DHCP Snooping:** DAI determines the validity of an Address Resolution Protocol (ARP) [26] message by checking valid IP-MAC bindings stored in DHCP snooping database. If a DAI enabled switch receives an ARP message that induces such an IP-MAC binding which contradicts with an IP-MAC binding present in the DHCP snooping database, the ARP message is dropped. Since Type-1 and Type-2 classical DHCP starvation attack do not require sending spoofed ARP replies, they can not be mitigated using DAI. In case of first and second variant of Induced DHCP starvation attack where a malicious client sends fake ARP replies in response

to IP conflict detection probes sent by victim client and DHCP server respectively, these fake replies are dropped by the DAI enabled switch and thus, attack is not successful. However, if the first variant of Induced DHCP starvation attack is launched in a network topology similar to the one shown in Figure 4.1, the fake probe replies sent remain local to the access point and does not traverse the DAI enabled switch. Thus, first variant of Induced DHCP starvation attack can not be mitigated using DAI

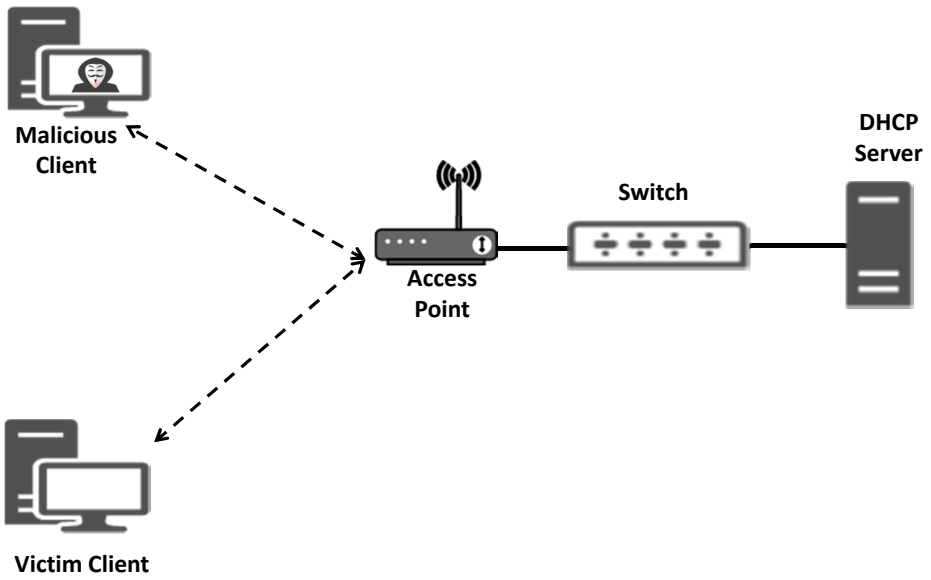



Figure 4.1: A network topology

in such network topologies. If second variant of Induced DHCP starvation attack is launched in such topologies, it is mitigated by DAI as the fake probe replies are sent in response to the DHCP server's probe request and thus, the fake replies have to traverse the network switch. In case of third variant of Induced DHCP starvation attack, since a malicious client manually configures its interface with an IP address without the intervention of DHCP server, snooping database does not contain an entry for an IP address bind to malicious client's MAC address. As a result, DAI does not drop the fake ARP requests sent by the malicious client while launching the attack. Thus, the third variant of Induced DHCP starvation attack can be successfully launched independent of the network topology. In Section 4.4.7, we also present the experiments performed to test these security features against different variants of Induced DHCP

starvation attack.

### 4.2.2 Limiting Number of IP Addresses on a Switch Port

There are few approaches which suggest to put a limit on number of IP addresses that can be assigned to a client connected to a particular port of a switch. One such approach is presented in [22] in which a relay agent includes the port number and the switch ID in the relay agent information option field of DHCP header before forwarding the DHCP messages to a DHCP server. Using this information, the DHCP server verifies that the client connected to the port has not exceeded the maximum number of IP addresses allowed per port. Another technique proposed in [21] estimates the number of contending users at each port of a switch and thereafter proposed a way to fairly allocate IP addresses for each port. In this way, these approaches prevent the consumption of IP address pool available at the server. However, a malicious client can simply connect to a switch port and consume fair share of IP addresses allotted at that port. This leads to starvation for upcoming legitimate clients which are going to connect to the port, the malicious client is already connected with. Thus, limiting number of IP addresses on a switch port does not completely mitigate any of the DHCP starvation attacks.

### 4.2.3 Cryptography- and Certificate-based Techniques

DHCP starvation attacks are possible because of no built-in authentication in DHCP protocol. Various cryptographic techniques [27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37] are proposed in the literature to enforce authentication in DHCP to mitigate starvation attacks. These techniques can mitigate any identity spoofing based attacks including the DHCP starvation attacks. However, these techniques have several issues and limitations. First, implementation of these techniques in a local network is very complex. Moreover, some of them require intervention of network administrator to distribute the certificates and shared keys beforehand using an out-of-band mechanism. The requirement of human intervention straightaway conflicts the purpose of

using DHCP which was designed for automatic IP address allocation. Another issue is the addition of significant complexity and traffic load due to the involvement of various third party modules. Due to such complex infrastructural requirement, the cryptographic techniques are rarely implemented in local networks.

Table 4.1 shows a summary of different mitigation techniques and their ability to counter the classical and Induced DHCP starvation attacks. A (✓) sign indicates

Table 4.1: Techniques to mitigate DHCP starvation attacks

| **Attack** | **Cryptography-based** | **Port Security** | **DHCP Snooping** | **DAI** | **Limit on IP Addresses** |
|---|---|---|---|---|---|
| Classical: Type-1 | ✓ | ✓ | ✗ | ✗ | ✗ |
| Classical: Type-2 | ✓ | ✗ | ✓ | ✗ | ✗ |
| Induced: $1^{st}$ Variant | ✓ | ✗ | ✗ | ✗ | ✗ |
| Induced: $2^{nd}$ Variant | ✓ | ✗ | ✗ | ✓ | ✗ |
| Induced: $3^{rd}$ Variant | ✓ | ✗ | ✗ | ✗ | ✗ |

that the technique can mitigate the corresponding starvation attack while a (✗) sign indicates that the technique can not mitigate the corresponding starvation attack. We can see that the classical DHCP starvation attack can be mitigated when security features in switches are used in tandem with each other. However, first and third variant of Induced DHCP starvation attack can not be mitigated by any of the security features. Moreover, though second variant of Induced DHCP starvation attack can be mitigated using DAI, this attack can not be detected in a wireless network topology similar to the one shown in Figure 4.2 where there is no switch and thus, DAI can not be used.

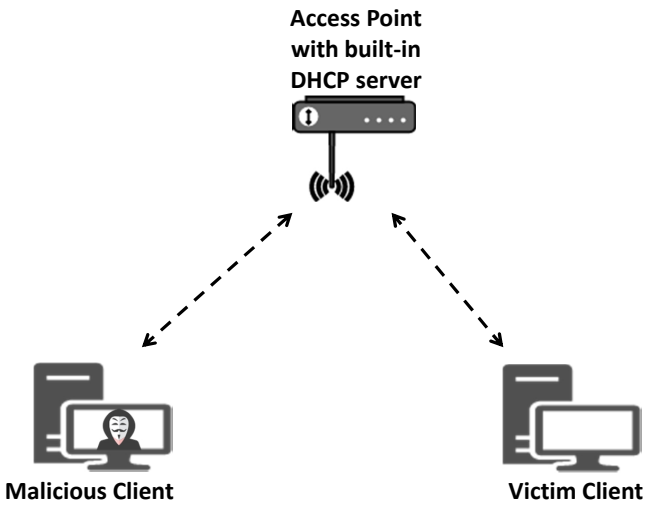



Figure 4.2: A network topology

## 4.3 Detecting Induced DHCP Starvation Attack

In the last chapter, we described that the Induced DHCP starvation attack is effective and easier-to-launch in both wired and wireless networks. Moreover, in Section 4.2 of this chapter, we also described that security features available with modern switches and other detection/mitigation techniques proposed in literature fall short in detecting the variants of Induced starvation attack. Hence we propose three detection approaches to detect this attack. The first and second detection approaches involve comparison of DHCP traffic profiles generated during training and testing phases while the third approach uses various machine learning algorithms to detect anomalies in the DHCP traffic and hence Induced DHCP starvation attack. These detection approaches are described in next three subsections.

### 4.3.1 Detection Approach-1

In this subsection, we describe a statistical abnormality measurement technique to detect the different variants of Induced DHCP Starvation attack. In particular, we treat the normal behavior of DHCP operation as a probability distribution comprising of DHCPDISCOVER, DHCPOFFER, DHCPREQUEST, DHCPACK and DHCPDECLINE messages. This detection method is motivated by the fact that various DHCP

messages have strong correlation between them[1] in a normal scenario. For example, every DHCPDISCOVER message is most likely followed by a DHCPOFFER message and subsequently with a DHCPREQUEST and a DHCPACK messages. This balance is disturbed due to either increased number of DHCPDECLINE messages in case of first and third variant of Induced DHCP Starvation attack or absence of all types of DHCP messages except DHCPDISCOVER in case of second variant of Induced DHCP starvation attack. We exploit this change in observation to detect the different variants of Induced DHCP starvation attack.

Our detection method has two phases of operation as training and testing phases. In the training phase, we collect normal DHCP traffic and generate normal DHCP profile (or probability distribution graph) and in the testing phase, we compare the profile generated using current DHCP traffic with the normal DHCP profile generated during training phase to detect the attack. For this comparison, we calculate Hellinger Distance [18] (See Appendix A) between the DHCP profiles generated during the training phase and different time intervals of the testing phase to detect different variants of Induced DHCP starvation attack.

**Adapting Hellinger Distance to Detect Induced DHCP Starvation Attack:** Hellinger Distance can easily be adapted to determine if there is a significant deviation in the DHCP traffic profiles $\mathcal{P}$ and $\mathcal{Q}$ generated during training phase and a particular time interval of the testing phase respectively. Here $\mathcal{P}$ and $\mathcal{Q}$ are $N$ dimensional vectors with each component of the vector representing the probability of occurrence of a particular type of DHCP message during training phase and one of the intervals of the testing phase respectively. Hellinger Distance between two profiles $\mathcal{P}$ and $\mathcal{Q}$ is calculated as in Equation 4.1. In this equation, $N$ denotes the number of message types and $P_i$ and $Q_i$ indicate the probability of occurrence of $i^{th}$ type of message in $\mathcal{P}$ and $\mathcal{Q}$ respectively.

$$d_H = (\frac{1}{2}\Sigma_{i=1}^{N}(\sqrt{P_i} - \sqrt{Q_i})^2)^{\frac{1}{2}} \tag{4.1}$$

[1]Authors in [76] present similar observation in the context of SIP.

The value of $d_H$ ranges between 0 and 1 with 0 representing perfect similarity and 1 representing maximum dissimilarity between $\mathcal{P}$ and $\mathcal{Q}$.

**Normal DHCP Profile Generation during Training Phase:** In order to generate a profile of normal DHCP operation, we create a probability distribution using DHCP messages over a period of $W$ observation intervals, each of duration $\Delta T$. The profile $\mathcal{P}$ generated in the training phase has 5 attributes as $\{P_{DISC}, P_{OFFER}, P_{REQ}, P_{ACK}, P_{DECL}\}$ where $P_{DISC}, P_{OFFER}, P_{REQ}, P_{ACK}$ and $P_{DECL}$ represent the probability of occurrence of DHCPDISCOVER, DHCPOFFER, DHCPREQUEST, DHCPACK and DHCPDECLINE messages respectively. Algorithm 4.1 describes the procedure of generating the required training profile $\mathcal{P}$. This

**Algorithm 4.1** Training phase for the first detection approach

**Input:** $\Delta T$ - Time period in hours
**Input:** $W$ - Number of intervals (each of $\Delta T$ hours)
**Result:** $\mathcal{P}$ - Probability distribution
1: $Sum = 0$
2: Create an array $Message\_Count[1, \cdots, 5]$
3: **for** $w = 1$ to $W$ **do**
4: **for** $t = t_{start}$ to $(t_{start} + \Delta T)$ **do**
5: $Message \leftarrow$ New message of type $m$ received
6: $Message\_Count[m] \leftarrow Message\_Count[m] + 1$ // where $m = 1, \cdots, 5$
7: **end for**
8: **end for**
9: **for** $m = 1$ to 5 **do**
10: $Sum \leftarrow Sum + Message\_Count[m]$
11: **end for**
12: **for** $m = 1$ to 5 **do**
13: $\mathcal{P}[m] \leftarrow \frac{Message\_Count[m]}{Sum}$
14: **end for**
15: Return $\mathcal{P}$

algorithm takes the time period $\Delta T$ of the training phase and $W$ time intervals as input and counts the number of occurrences of different message types during $\Delta T$ time interval. This is repeated for every $\Delta T$ time period of the training phase. For example, if $W$=16, the number of occurrences of different message types is counted in all these 16 time intervals. After that, the number of occurrences of different message types is divided by the total number of occurrences of all the message types in these 16 time intervals to obtain the probability of occurrence of different message types in a time interval of the training phase. This probability of occurrence of each message

type in a time interval represents the normal DHCP training profile.

**Testing Phase**: Once the system is trained and $\mathcal{P}$ is created, it can be deployed to detect different variants of Induced DHCP starvation attack from $(W+1)^{th}$ interval of duration $\Delta T$. Algorithm 4.2 describes procedure to detect anomalies in different time intervals of the testing phase. This algorithm takes $\Delta T$ time period

**Algorithm 4.2** Testing phase for the first detection approach

**Input:** $\Delta T$ - Time period in hours
**Input:** $\mathcal{P}$ - Probability distribution generated during training phase
**Output:** Detection of Induced starvation attack

**while** Not interrupted **do**
  Create an array $Message\_Count\_Test[1, \cdots, 5]$
  $Sum = 0$
  **for** t=$t_{start}^{test}$ to $(t_{start}^{test} + \Delta T)$ **do**
    $Message \leftarrow$ New message of type $m$ received
    $Message\_Count\_Test[m] \leftarrow Message\_Count\_Test[m] + 1$ // where $m = 1, \cdots, 5$
  **end for**
  **for** $m = 1$ to 5 **do**
    $Sum \leftarrow Sum + Message\_Count\_Test[m]$
  **end for**
  **for** $m = 1$ to 5 **do**
    $\mathcal{Q}[m] \leftarrow \frac{Message\_Count\_Test[m]}{Sum}$
  **end for**
  $HD \leftarrow$ Calculate Hellinger Distance between $\mathcal{P}$ and $\mathcal{Q}$
  **if** $HD \geq \delta$ **then**
    Attack detected
  **end if**
**end while**

and probability distribution $\mathcal{P}$ generated during training phase as input and creates a profile $\mathcal{Q}$ after collecting all types of DHCP messages in $\Delta T$. The algorithm then calculates the Hellinger Distance between $\mathcal{P}$ and $\mathcal{Q}$ and if the calculated distance is greater than a predefined threshold $\delta$, attack is detected.

**Choosing Threshold $\delta$:** Network administrators need to choose threshold Hellinger Distance $\delta$ wisely as smaller $\delta$ values may lead to increased number of false positives while larger $\delta$ values may result into increased number of false negatives. Thus, the best practice is to choose a threshold marginally greater than the Hellinger Distances from normal DHCP profiles.

### 4.3.2 Detection Approach-2

Similar to the previous detection approach, this approach also compares the probability distributions of various DHCP message types generated during training and testing phases. However, unlike the previous case, this method generates probability distributions taking into account the time of day. This makes the algorithm robust against variations in traffic profiles and can adopt to changes like day and night traffic and also week days and weekend traffic patterns. If the probability distribution of a DHCP message type generated during a time interval, say 12:00pm - 12:10pm, of a day of training phase is reasonably different from the probability distribution of the same DHCP message type generated during the time interval 12:00pm - 12:10pm of the testing phase, anomaly is detected in that interval.

**Probability Distribution Generation of DHCP Messages during Training Phase:** In the training phase, the normal probability distribution of different types of DHCP messages are generated. Algorithm 4.3 describes the procedure for generating probability distribution of a DHCP message type $m$ in the training phase. This algorithm takes time period $\Delta T$ and number of days $d$ in the training phase as input and first derives the number of time intervals ($W$) in a day. It then counts the number of occurrences of a message type $m$ during different time intervals of a day of training phase and stores the counts and clock times when different time intervals started and ended (in arrays $COUNT$, $Initial\_Clock\_Time$ and $Final\_Clock\_Time$ respectively shown in Algorithm 4.3). This is repeated for each day of the training phase. The algorithm then computes the probability of occurrence of $m$ in a particular time interval of a day of the training phase as the ratio of number of occurrences of $m$ in that particular time interval to the total number of occurrences of $m$ over training phase. This generates a probability distribution of $m$ for the entire day which is the normal profile of our detection system. This profile represents the likelihood of receiving $m$ in a particular time interval of the day.

**Algorithm 4.3** Training phase for the second detection approach

**Input:** $\Delta T$ - Time period in hours
**Input:** $d$ - Training phase in days
**Result:** $PD$ - Probability distribution

$Sum = 0$
$W \leftarrow \frac{24}{\Delta T}$
Create three arrays $COUNT[1, \cdots, W]$, $Initial\_Clock\_Time[1, \cdots, W]$ and $Final\_Clock\_Time[1, \cdots, W]$
$Initial\_Clock\_Time[1] \leftarrow$ Current Clock Time
$Final\_Clock\_Time[1] \leftarrow Initial\_Clock\_Time[1] + \Delta T$
**for** $d = 1$ to $d$ **do**
    **for** $w = 1$ to $W$ **do**
        $Message\_Count = 0$
        **for** $t = Initial\_Clock\_Time[w]$ to $Final\_Clock\_Time[w]$ **do**
            $Message \leftarrow$ New message of type $m$ received
            $Message\_Count \leftarrow Message\_Count + 1$
            **if** $w$ is not equal to $W$ **then**
                $Initial\_Clock\_Time[w + 1] \leftarrow Final\_Clock\_Time[w]$
                $Final\_Clock\_Time[w + 1] \leftarrow Final\_Clock\_Time[w] + \Delta T$
            **end if**
        **end for**
        $COUNT[w] \leftarrow COUNT[w] + Message\_Count$
    **end for**
**end for**
**for** $i = 1$ to $W$ **do**
    $Sum \leftarrow Sum + COUNT[i]$
**end for**
**for** $i = 1$ to $W$ **do**
    $PD[i] \leftarrow \frac{COUNT[i]}{Sum}$
**end for**
Return $PD$, $Initial\_Clock\_Time$ and $Final\_Clock\_Time$

**Testing Phase:** Algorithm 4.4 describes procedure to detect anomalies in different time intervals of the testing phase. This algorithm takes time period $\Delta T$, number of days $d$ in the training phase, total number of occurrences of $m$ during training phase and $PD$ generated during training phase as input. It then counts the number of occurrences of $m$ in the time interval $Initial\_Clock\_Time$ - $Final\_Clock\_Time$ of the testing phase. After this, detection system triggers another procedure to count the number of occurrences of $m$ expected during the same time interval $Initial\_Clock\_Time$ - $Final\_Clock\_Time$ of a day in training phase. This procedure is described in Algorithm 4.5. Algorithm 4.5 first retrieves the probability of occurrence of $m$ in the time interval $Initial\_Clock\_Time$ - $Final\_Clock\_Time$ of a day of the training phase from probability distribution generated during the training phase. The retrieved probability is then multiplied with the average number of occurrences of $m$ in one day of the training phase to get the number of occurrences of $m$ during the interval $Initial\_Clock\_Time$ - $Final\_Clock\_Time$ of the day of

**Algorithm 4.4** Testing phase for the second detection approach

**Input:** $\Delta T$ - Time period in hours
**Input:** $Sum$ - Total number of occurrences of $m$ during training phase
**Input:** $PD$ - Probability distribution of $m$ generated during training phase
**Input:** $d$ - Training phase in days
**Output:** Detection of Induced DHCP starvation attack

```
while Not interrupted do
    Message_Count = 0
    Initial_Clock_Time ← Current Clock Time
    Final_Clock_Time ← Initial_Clock_Time + ΔT
    for t = Initial_Clock_Time to Final_Clock_Time do
        Message ← New message of type m received
        Message_Count = Message_Count + 1
    end for
    Message_Count_train ← GetCount(Initial_Clock_Time, Final_Clock_Time, PD, Sum, d)
    if Message_Count ≥ Message_Count_train + β then
        Attack detected
    end if
end while
```

**Algorithm 4.5** GetCount($Initial\_Clock\_Time$, $Final\_Clock\_Time$, $PD$, $Sum$, $d$)

```
PD_test ← PD during Initial_Clock_Time - Final_Clock_Time
AvgSum ← Sum / d
Message_Count_train ← PD_test * AvgSum
Return Message_Count_train
```

training phase. This count, $Message\_Count\_train$, is returned back to Algorithm 4.4. Subsequently, Algorithm 4.4 computes the difference between the number of occurrences of $m$ seen during the interval $Initial\_Clock\_Time$ - $Final\_Clock\_Time$ of the training and testing phases. If the difference is more than a predefined threshold $\beta$, the detection system raises the alarm and alerts the network administrators.

**Choosing Threshold $\beta$:** Similar to the Hellinger Distance based detection approach, this approach also requires a network administrator to carefully set a $\beta$ threshold as smaller $\beta$ values may lead to higher number of false positives while larger $\beta$ values may result into higher number of false negatives. Thus, if a network is highly dynamic where number of clients joining and leaving the network at a particular time of a day differs vastly on a daily basis, chosen $\beta$ value should be relatively larger. However, networks where the number of clients connected to the network at a particular time of a day remains relatively same on a daily basis, choosing a smaller $\beta$ value is more appropriate. Also, it is to be noted that DHCP traffic patterns in a network during week days and weekend may vary a lot due to which a DHCP profile

generated during week day should not be compared with the one generated during weekend as it leads to missed detections. The network administrators can overcome this issue by simply generating different DHCP profiles for different days of a week and use these profiles when required. We should also notice that $\beta$ value need not be changed for different time intervals of a day as number of occurrences of $m$ in a particular time interval of testing phase is always compared with the number of occurrences of $m$ in the same time interval of training phase also.

### 4.3.3 Detection Approach-3

Our third anomaly detection system uses machine learning algorithms to detect different variants of Induced DHCP starvation attack. In particular, we compare the performance of six popular one-class classification algorithms to detect the attack. One-class classification algorithms (see Appendix B) are found to be highly useful in the scenarios when information of only one of the classes, the target class, is available. The task of classification involves defining such boundary around target class so as to cover as much of the target objects as possible and at the same time to minimize the chance of accepting outlier objects. The reason behind choosing different one-class classification algorithms is to test the detection performance of these algorithms to detect different variants of Induced DHCP starvation attack. We compare the performance of following one-class classification algorithms to detect different attack variants:

**1. Gaussian Density based One-class Classifier [77]:** Out of different density based models, Gaussian density [77] is the most simple model. This algorithm first estimates density of the training data and then set a threshold on this density. If this threshold is crossed during testing phase, the sample is detected as anomaly otherwise normal.

**2. Naive Bayes One-class Classifier [78]:** Naive Bayes one-class classifier belongs to the category of probabilistic classifiers [78] based on the application of Bayes' theorem with strong independence assumptions between the features. In this case also, the algorithm sets a threshold based on the probabilities obtained during training phase.

During testing phase, if probability of a testing sample is greater than this threshold, the sample is considered as anomaly, otherwise, it is considered as belonging to normal class.

**3. $K$-Nearest Neighbour (KNN) algorithm [79]:** $K$-Nearest Neighbour (KNN) algorithm is considered as a *lazy* algorithm which takes decision by considering entire training dataset. While using KNN for classification purpose, the output is calculated as the class with the highest occurrence from $K$-nearest neighbours. Each of the instance votes for the class to which it belongs and the class with most votes is taken as the prediction. This algorithm calculates outlier score as the ratio between two distances for each testing sample. The first distance is the distance between the test sample $x_i$ and its $k^{th}$ nearest neighbour in the training set NN($x_i$, k). The second distance is the distance between the $k^{th}$ nearest training data point and its $k^{th}$ nearest neighbour. Based on the outlier score, the classifier decides whether the testing sample belongs to anomalous or normal class.

**4. $K$-means clustering algorithm [77]:** $K$-means clustering algorithm partitions the data into a group of data points. These groups of data points are known as clusters. Assuming we have $n$ data points $x_i$ where $i = 1, 2, \ldots, n$ which are to be partitioned in $k$ different clusters such that each data point is assigned to a cluster. The algorithm sets a threshold value based on the distance between the training samples and cluster centers. During testing phase, if the distance between the testing sample and cluster centers exceeds the threshold, the sample is considered as belonging to anomalous class.

**5. Principal Component Analysis (PCA) [77]:** Principal Component Analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. The algorithm decides a threshold value based on the reconstruction error obtained by PCA. If reconstruction error of a testing sample exceeds this threshold, the sample is considered as anomalous one.

**6. Support Vector Data Description (SVDD) [80]:** Support Vector Machine (SVM) based one-class classifier, called as Support Vector Data Description (SVDD)

[80], aims to obtain a spherical boundary instead of a planar one around the data in feature space. The goal is to minimize the volume of hypersphere in such a way that maximum number of the normal samples can be covered without incorporating anomalies in the solution. If a sample falls outside the sphere, it is treated as an anomalous sample.

Like previous two detection approaches, this detection approach has also two phases of operation as training and testing phases. Algorithm 4.6 shows the procedure of training the detector model during training phase. This algorithm takes

**Algorithm 4.6** Training phase for the third detection approach

**Input:** $\Delta T$ - Time period in hours
**Input:** $d$ - Training phase in days
**Result:** Trained detection model
$W \leftarrow \frac{24}{\Delta T}$
Create two dimensional arrays $Feature[1, \cdots, W][1, \cdots, 4]$ and $Avg\_Feature[1, \cdots, W][1, \cdots, 4]$
**for** $d = 1$ to $d$ **do**
    **for** $w = 1$ to $W$ **do**
        **for** $t = t_{start}$ to $(t_{start} + \Delta T)$ **do**
            $Message \leftarrow$ New message of type $m$ received
            $Feature[w][m] \leftarrow Feature[w][m] + 1$ // where $m = 1, \cdots, 4$
        **end for**
    **end for**
**end for**
**for** $w = 1$ to $W$ **do**
    **for** $m = 1$ to $4$ **do**
        $Avg\_Feature[w][m] \leftarrow \frac{Feature[w][m]}{d}$
    **end for**
**end for**
$Training\_dataset \leftarrow Avg\_Feature, Normal$
$\mathcal{M} \leftarrow (Training\_dataset, classifiers)$
Return $\mathcal{M}$

Table 4.2: Candidate features to detect attacks

| Features | Description | Type |
|---|---|---|
| Feature-1 | Number of DHCPDISCOVER messages in $\Delta T$ | Numeric |
| Feature-2 | Number of DHCPREQUEST messages in $\Delta T$ | Numeric |
| Feature-3 | Number of DHCPDECLINE messages in $\Delta T$ | Numeric |
| Feature-4 | Number of ARP probe requests in $\Delta T$ | Numeric |

time period $\Delta T$ of training phase and number of training days $d$ as input and creates data records after every $\Delta T$ time duration during training phase. Each data record consists of features representing the characteristics of network traffic during

$\Delta T$. These features along with their types are shown in Table 4.2[2]. Few examples of data records generated during training phase are shown in Table 4.3 where network traffic in time intervals $T_1$, $T_2$ and $T_3$ belong to the normal class. Algorithm 4.6 marks these data records with normal label and generates the required training dataset. Using this dataset, the detection model is trained which is then tested with new or unknown dataset containing both normal and anomalous data records. Algorithm 4.7 describes the procedure of how anomalies are detected in different time intervals of the testing phase. This algorithm generates data record from the DHCP and ARP

Table 4.3: Sample data records generated during training phase

| Interval | Data of 4 features | Class |
|---|---|---|
| $T_1$ | 69, 250, 1, 1286 | Normal |
| $T_2$ | 49, 260, 0, 1435 | Normal |
| $T_3$ | 58, 127, 0, 1150 | Normal |

**Algorithm 4.7** Testing phase for the third detection approach

**Input:** $\Delta T$ - Time period in hours
**Input:** $\mathcal{M}$ generated during training phase
**Output:** Detection of Induced DHCP starvation attack
1: $W \leftarrow \frac{24}{\Delta T}$
2: Create an array $Features\_Test[1, \cdots, 4]$
3: **for** $t = t_{start}$ to $(t_{start} + \Delta T)$ **do**
4: $\quad Message \leftarrow$ New message of type $m$ received
5: $\quad Feature\_Test[m] \leftarrow Feature\_Test[m] + 1$ // where $m = 1, \cdots, 4$
6: **end for**
7: $Decision \leftarrow \mathcal{M}(Feature\_Test[m])$
8: Return $Decision$

traffic collected in $\Delta T$ time interval of the testing phase and passes it to the trained detector model $\mathcal{M}$ generated during training phase. Eventually, $\mathcal{M}$ returns back the decision whether the traffic present in $\Delta T$ time interval is anomalous or not.

[2]Instead of unicast messages (DHCPOFFER, DHCPACK and ARP reply), we consider counts of only broadcast messages (DHCPDISCOVER, DHCPREQUEST, DHCPDECLINE and ARP requests) as features. Thus, even if proposed anomaly detection framework can not receive unicast communication, it can still detect anomaly in the network.

## 4.4 Experimental Results

In this section, we describe the experiments conducted to evaluate the detection performance of proposed approaches. We first describe the testbed setup used to collect the required network traffic in order to generate the dataset for training and testing purpose. We then present the detection results of different approaches and subsequently, we also describe the experiments conducted to test the ability of various security features of network switches to detect different variants of Induced DHCP starvation attack.

### 4.4.1 Testbed Setup for Traffic Collection

We used our institute network which connects approximately 1000 heterogeneous clients like mobile phones, laptops and desktops as shown in Figure 4.3 for DHCP and ARP traffic collection. There were four types of devices in the network as DHCP

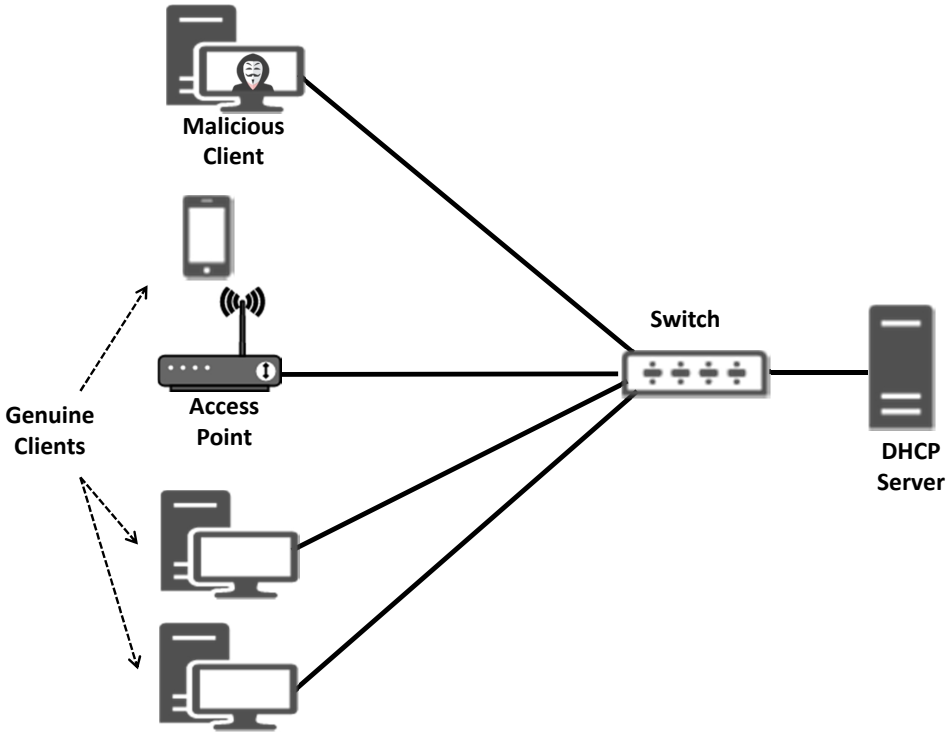



Figure 4.3: Testbed setup for traffic collection

server, malicious client, switch and genuine clients. The DHCP server used in the

institute network was ISC DHCP server. This server was configured to offer 65,536 IP addresses[3] in the range of 10.100.0.0 to 10.100.255.255. The malicious client was having 4 GB of physical memory and Core i5 quad core processor and using Kali 2.0 operating system.

To collect normal DHCP and ARP traffic for training purpose, we installed tcpdump, a packet sniffer, on DHCP server and used appropriate capturing filters so that only the required traffic was collected. In order to collect mixed (normal and anomalous) traffic for testing phase, we used python scripts on the malicious client to send fake requests/replies to launch different variants of Induced DHCP starvation attack against only one victim client. Thus, as discussed in Section 3.5.4 of Chapter 3, the attack rate was dependent only on how frequent the victim client was trying to obtain IP address. It should be noted that while the attack was going on, other genuine clients in the network were successfully obtaining IP addresses. In this way, we collected mixed DHCP and ARP traffic for testing purpose.

### 4.4.2 Training and Testing Dataset

We collected 9 days of DHCP and ARP traffic from the DHCP server. Out of that, the traffic collected during the first 3 days was normal while in the remaining 6 days of traffic collection, we launched one variant of Induced DHCP starvation attack such that each variant lasted for 2 days. For training purpose, we used first day normal traffic while remaining 8 days of traffic was used for testing purpose. Out of these 8 days of traffic, we had 2 days of normal traffic and 6 days of attack traffic. We divided the collected traffic into smaller time intervals of $\Delta T = 10$ minutes. As a result, we obtained 144 smaller time intervals for training purpose while 1152 time intervals for testing purpose. Out of these 1152 time intervals, there were 288 normal and 864 attack time intervals. The details of the training and testing dataset are shown in Table 4.4. We tested the performance of different proposed detection approaches using this dataset and the results obtained are presented in next few subsections.

[3]Among 65536, one each was allotted to malicious client and server itself. Other two IP addresses, 10.100.0.0 and 10.100.255.255 were Network and Broadcast address respectively and were not used.

Table 4.4: Training and testing dataset details

| Details | Training Dataset | Testing dataset |
|---|---|---|
| Number of time intervals without attack | 144 | 288 |
| Number of time intervals with attack | 0 | 864 |
| Total intervals | 144 | 1152 |
| Time period (in minutes) | 10 | 10 |
| Total number of DHCPDISCOVER messages | 16110 | 176672 |
| Total number of DHCPOFFER messages | 15678 | 165728 |
| Total number of DHCPREQUEST messages | 16398 | 171488 |
| Total number of DHCPACK messages | 15678 | 165728 |
| Total number of DHCPDECLINE messages | 36 | 30820 |
| Total number of ARP requests | 188223 | 1586230 |
| Total number of messages | 252123 | 2296666 |

### 4.4.3 Evaluation of Detection Approach-1

As discussed earlier in Section 4.3.1, there is a strong correlation between occurrences of DHCPDISCOVER, DHCPOFFER, DHCPREQUEST and DHCPACK messages. To establish this fact, we used 24 normal time intervals of training dataset to plot a graph of these four DHCP messages along with the DHCPDECLINE messages as shown in Figure 4.4. We can see from this figure that the occurrences of

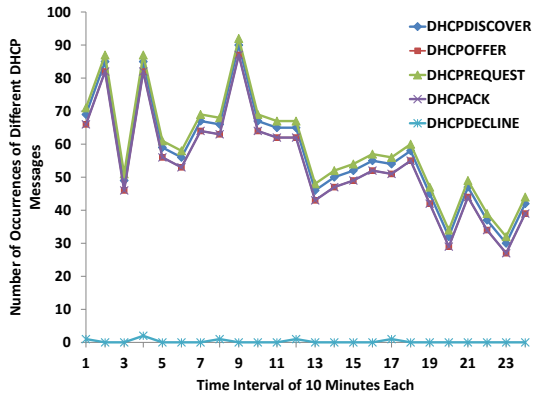



Figure 4.4: Correlation of different DHCP messages

4 out of 5 message types (except DECLINE) are strongly correlated with each other, however, the occurrence of DHCPDECLINE messages do not follow this correlation as address conflicts in a network during the normal operation of DHCP are quite rare and thus, the number of occurrences of DHCPDECLINE messages is very small.

**Training Phase:** During training phase, we created the normal DHCP profile using the training dataset containing 144 normal time intervals of $\Delta T = 10$ minutes each. Figure 4.5 shows the generated training profile. This profile represents

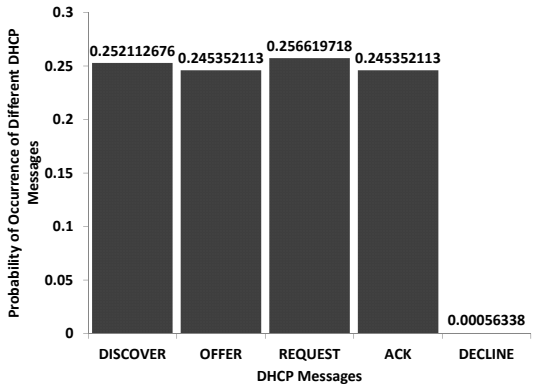



Figure 4.5: Normal DHCP training profile

the normal behavior of DHCP traffic. We can notice that the probabilities of occurrence of DHCPDISCOVER, DHCPOFFER, DHCPREQUEST and DHCPACK messages are almost equal but the probability of occurrence of DHCPDECLINE messages is very small as compared to the probabilities of other DHCP message types.

**Testing Phase:** By using the dataset generated for testing phase, we generated traffic probability distributions in different intervals and calculated the Hellinger Distance between the interval under consideration with that of training profile. A sample profile generated during a time interval of testing phase containing normal traffic is shown in Figures 4.6a. Similarly, sample profiles generated during three different time intervals of the testing phase containing the traffic of first, second and third variant of Induced DHCP starvation attack are shown in Figures 4.6b, 4.6c and 4.6d respectively. Corresponding to these sample time intervals, the number

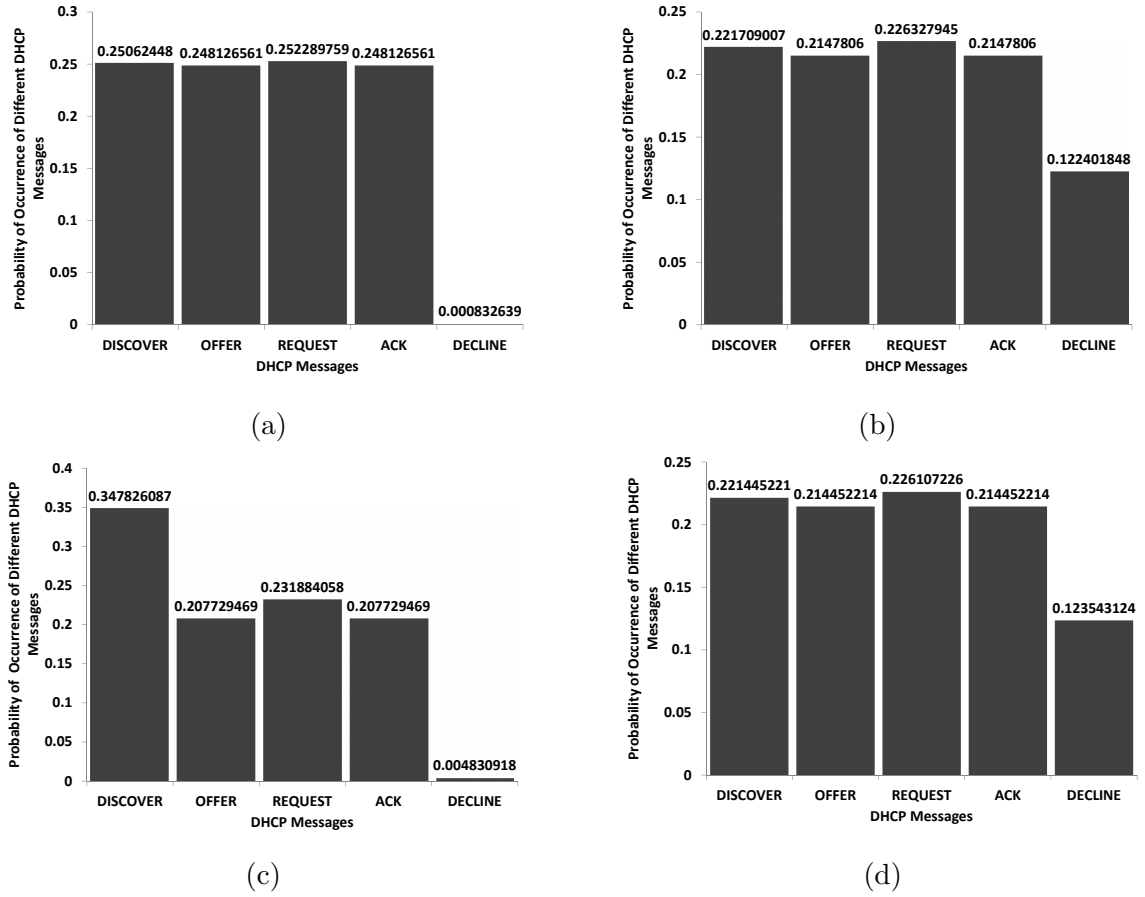



Figure 4.6: Profile with (a) normal, (b) variant-1, (c) variant-2 and (d) variant-3 traffic

of different DHCP message types, number of attempts made by the victim client to acquire an IP address and number of times it was prevented from acquiring the address are shown in Table 4.5. We also notice that Figure 4.6a is almost similar to the normal DHCP training profile shown in Figure 4.5. Table 4.6 shows the obtained minimum and maximum Hellinger Distances between the training profile and the profiles with normal traffic and traffic of variant-1, variant-2 and variant-3 launched against only one victim client during testing phase. To detect different attack variants, we chose the thrice of mean Hellinger Distance obtained by comparing normal DHCP profiles as the threshold. To obtain the mean Hellinger Distance, we first collected 2 more hours of normal DHCP traffic, divided it into smaller intervals of 10 minutes each and created DHCP profiles in each time interval. We then calculated Hellinger Distances between training profile and profiles generated during these time

intervals. In this way, we calculated mean and used it as the threshold. We used Recall and FPR as the performance metrics of the detection approach. Recall and FPR are given by Equation 4.2 and 4.3 respectively.

$$Recall = \frac{tp}{tp + fn} \tag{4.2}$$

$$FPR = \frac{fp}{tn + fp} \tag{4.3}$$

where $tp$, $tn$, $fp$ and $fn$ are as follows:

True Positive ($tp$): Intervals correctly detected as containing attack traffic

True Negative ($tn$): Intervals correctly detected as containing normal traffic

False Positive ($fp$): Intervals incorrectly detected as containing attack traffic

False Negative ($fn$): Intervals incorrectly detected as containing normal traffic

Table 4.5: Details of four time intervals of testing phase

| ***Number of-*** | **Normal Traffic** | **Variant-1 Traffic** | **Variant-2 Traffic** | **Variant-3 Traffic** |
|---|---|---|---|---|
| DHCPDISCOVER messages | 301 | 96 | 72 | 95 |
| DHCPOFFER messages | 298 | 93 | 43 | 92 |
| DHCPREQUEST messages | 303 | 98 | 48 | 97 |
| DHCPACK messages | 298 | 93 | 43 | 92 |
| DHCPDECLINE messages | 1 | 53 | 1 | 53 |
| Attempts by victim client | 1 | 53 | 23 | 53 |
| Failed attempts | 0 | 53 | 23 | 53 |

The $tp$, $tn$, $fp$, $fn$, $Recall$ and $FPR$ obtained in this case were 639, 284, 4, 225, 73.96 % and 1.39 % respectively.

Table 4.6: Min/max Hellinger Distances

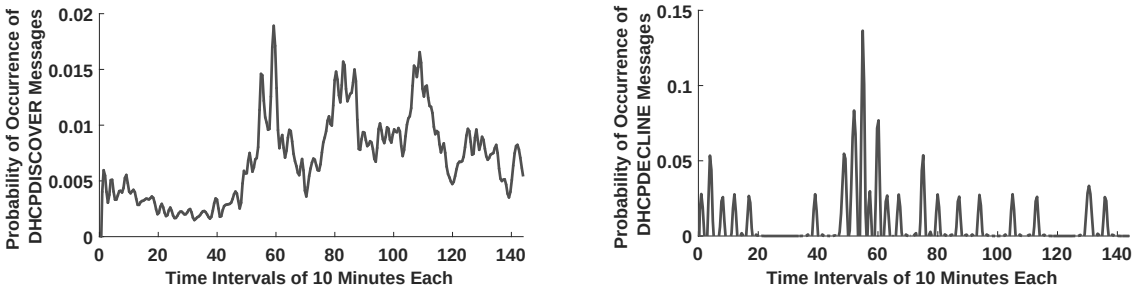
| **Scenario** | **Min/max Hellinger Distance** |
|---|---|
| Intervals containing normal traffic | 0.0056/0.0903 |
| Intervals containing variant-1 traffic | 0.1163/0.2567 |
| Intervals containing variant-2 traffic | 0.0128/0.1265 |
| Intervals containing variant-3 traffic | 0.1183/0.2551 |

### 4.4.4 Evaluation of Detection Approach-2

As discussed earlier in Section 4.3.2, this detection approach first generates probability distributions of various DHCP messages during training phase and based on these distributions, the expected counts of different DHCP message types in different time intervals of a day are estimated. If the counts of different DHCP message types observed during a particular time interval of testing phase exceed the counts of the corresponding DHCP message types expected during that time interval, anomaly is detected in that interval. To detect first and third variant of Induced DHCP starvation attack, we make use of profile of DHCPDECLINE message as the difference between the number of occurrences of these messages during normal scenario and attack scenario is quite large and thus, attacks can be detected with high accuracy. To detect second variant of Induced DHCP starvation attack, we make use of profile of DHCPDISCOVER message as this is the only message type whose count gets increased in case of second variant of Induced DHCP starvation attack. We describe the training and testing phases of this detection approach in subsequent paragraphs.

**Training Phase:** During training phase, we created the normal profile of DHCPDISCOVER and DHCPDECLINE messages using our training dataset containing 144 normal time intervals of $\Delta T = 10$ minutes each. Profile generation requires estimating the probability of occurrence of a DHCP message type in a particular time interval of a day which is the ratio of number of occurrences of the message type in that

interval to the total number of occurrences of that message in the day. The generated normal training profiles of DHCPDISCOVER and DHCPDECLINE messages over a period of 144 normal time intervals of 10 minutes each are shown in Figures 4.7a and 4.7b respectively. X-axis in these figures represents time intervals of the training phase while Y-axis represents the probability of occurrence of corresponding DHCP message type in different time intervals of a day of training phase.




(a) (b)

Figure 4.7: Normal training profile of (a) DHCPDISCOVER and (b) DHCPDECLINE message

**Testing Phase:** During testing phase, we used the training profiles of DHCPDISCOVER and DHCPDECLINE messages to detect the presence of different attack variants' traffic in 1152 time intervals of our testing dataset. To detect the second variant, we counted the number of occurrences of DHCPDISCOVER messages in a particular time interval of the testing phase and then compared it with the number of occurrences of DHCPDISCOVER messages that was expected in that time interval. This expected number of occurrences of DHCPDISCOVER message in that time interval was calculated by taking the product of total number of occurrences of DHCPDISCOVER messages in a day of the training phase with the probability of occurrence of DHCPDISCOVER messages in that time interval retrieved from Figure 4.7a. Similarly, to detect the first and third variants, we also compared the number of occurrences of DHCPDECLINE messages in a time interval of a day of the training phase with the number of its occurrences in the same interval of the testing phase.

Table 4.7 shows the minimum and maximum number of occurrences of different DHCP message types in different time intervals of the testing phase containing either normal traffic or the traffic generated by launching different variants of Induced DHCP starvation attack. The table also shows the minimum and maximum number of attempts made by the victim client to acquire the IP address and failed attempts to obtain the address eventually. To detect different attack variants, we chose a

Table 4.7: Minimum and maximum message count

| ***Min/Max number of-*** | **Normal Traffic** | **Variant-1 Traffic** | **Variant-2 Traffic** | **Variant-3 Traffic** |
|---|---|---|---|---|
| DHCPDISCOVER messages | 27/313 | 80/366 | 53/339 | 81/355 |
| DHCPOFFER messages | 24/302 | 74/364 | 46/335 | 80/352 |
| DHCPREQUEST messages | 30/315 | 98/368 | 54/343 | 86/359 |
| DHCPACK messages | 24/302 | 74/364 | 46/335 | 80/352 |
| DHCPDECLINE messages | 0/3 | 53/56 | 0/3 | 53/58 |
| Attempts by victim client | 1/1 | 53/53 | 20/26 | 53/53 |
| Failed attempts | 0/0 | 53/53 | 20/26 | 53/53 |

threshold such that the number of occurrences of a message type (DHCPDISCOVER for second variant and DHCPDECLINE for first and third variant) in a time interval of a testing phase should not be greater than the twice of number of occurrences of the same message type in the same time interval of a day of training phase. Based on this, the detection performance of this approach in terms of $tp$, $tn$, $fp$, $fn$, Recall and FPR obtained in this case were 621, 287, 1, 243, 71.87 % and 00.35 % respectively.

### 4.4.5 Evaluation of Detection Approach-3

As discussed in Section 4.3.3, our third detection approach uses various one-class classification algorithms to detect different variants of Induced DHCP starvation attack. The training and testing phases of this detection approach is as follows:

**Training Phase:** During training phase, we trained the selected one-class classifiers as a detection model using our training dataset containing 144 normal data records (time intervals). Moreover, all these 144 records were labeled as normal for this training.

**Testing Phase:** During testing phase, the trained detection model was used to test 1152 data records (time intervals) in our testing dataset. Table 4.8 shows detection performance of various one-class classifiers to detect different variants of Induced DHCP starvation attack in terms of Recall and FPR. We can notice from the

Table 4.8: Detection performance of various one-class classifiers

| **Classifier** | **tp** | **tn** | **fp** | **fn** | **Recall (in %)** | **FPR (in %)** |
|---|---|---|---|---|---|---|
| Gaussian Density based Classifier | 597 | 810 | 54 | 267 | 69.10 | 6.25 |
| Naive Bayes Classifier | 600 | 261 | 27 | 264 | 69.44 | 9.38 |
| KNN Classifier | 603 | 268 | 20 | 261 | 69.79 | 6.94 |
| $K$-means Clustering | 605 | 263 | 25 | 259 | 70.02 | 8.68 |
| PCA | 595 | 267 | 21 | 269 | 68.87 | 7.29 |
| SVDD | 615 | 251 | 37 | 249 | 71.18 | 12.85 |

table that SVDD gave the best detection results in terms of Recall but at the same time, the false positive rate was also increased due to misclassifications of normal intervals by SVDD.

### 4.4.6 Sensitivity Analysis

The detection performance of the proposed schemes depend on the rate with which different attacks are launched, the time period ($\Delta T$) chosen during training and testing phase and also the predefined threshold. In this subsection, we first describe the effects of varying the attack rate and time period on the detection performance of proposed schemes and subsequently, we also describe the effect of varying threshold on their detection performance.

#### 4.4.6.1 Varying Attack Rate and Time Period

We performed an experiment by varying attack launching rate and the time period $\Delta T$ to analyze the sensitivity of the proposed schemes to these two parameters. We collected another 45 hours of mixed traffic from the DHCP server of our testbed setup by injecting different attack variants at five different rates as shown in Table 4.9. Thus, there were 5 different scenarios with 9 hours of mixed DHCP traffic collected

Table 4.9: Different attack rates for sensitivity analysis

| | **Number of victim clients** | | | |
|---|---|---|---|---|
| **Scenario** | **Variant-1** | **Variant-2** | **Variant-3** | **Overall Attack Rate** |
| 1 | 0 | 1 | 0 | 1 |
| 2 | 0 | 2 | 0 | 2 |
| 3 | 1 | 2 | 1 | 4 |
| 4 | 1 | 3 | 2 | 6 |
| 5 | 2 | 4 | 2 | 8 |

in each scenario. We varied the time intervals in step size of 5 minutes ranging from 5 minutes to 15 minutes to test the effect of varying time period on the detection approaches. We describe the results obtained from this experiment while considering different detection approaches in the next three paragraphs.

**Detection Approach-1:** We calculated Hellinger Distances between profiles

generated during different time intervals of each scenario and the profile generated during training phase. Figure 4.8 shows the effect of varying attack rate and time period on the detection performance of the scheme in terms of Recall. We can

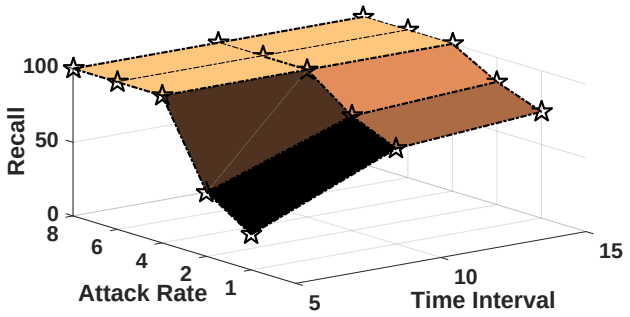



Figure 4.8: Effect of varying time period and attack rate at approach-1

notice that Recall increases with the increase in attack rate independent of time period. A Recall of 100% was achieved when at least 4 clients were targeted at a time independent of the time period. However, we can notice that if number of clients being targeted were less than 4, Recall increased with the increase in time period. Thus, a larger time period improves the detection performance of the scheme. However, choosing a larger time period results into late detection of possible ongoing attacks. Thus, time period should be chosen wisely.

**Detection Approach-2:** We compared the number of occurrences of DHCPDISCOVER and DHCPDECLINE messages in different time intervals of each scenario and the number of occurrences of DHCPDISCOVER and DHCPDECLINE messages in the same time intervals of a day of training phase. Figure 4.9 shows the effect of varying attack rate and time period on the detection performance of the scheme in terms of Recall. From Figures 4.8 and 4.9, we can notice that the effect of varying attack launching rate and the time period was same for both first and second detection approach.

**Detection Approach-3:** We also tested the effect of varying attack launching rate and the time period on our third detection approach. Since SVDD gave best

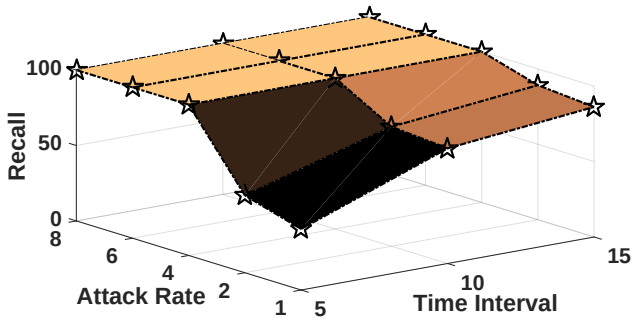



Figure 4.9: Effect of varying time period and attack rate at approach-2

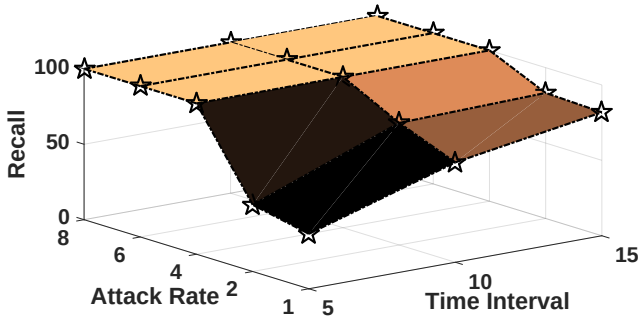



Figure 4.10: Effect of varying time period and attack rate at approach-3

detection results in our experiments, we show the effect of varying these parameters on the detection performance of SVDD in Figure 4.10. We can notice that varying both parameters causes the same effect on this detection approach also.

#### 4.4.6.2 Varying Predefined Threshold

To test the effect of varying the predefined threshold on the detection performance of proposed schemes, we used our training and testing dataset described in Section 4.4.2. We describe the effect of varying predefined threshold on the detection performance of our first and second approach (as we set predefined thresholds only in these two methods) in next two paragraphs.

**Detection Approach-1:** We tested the effect of varying predefined threshold on the detection performance of our first approach. For this purpose, we used five different thresholds as shown in Table 4.10. In this table, *Mean* corresponds

Table 4.10: Effect of varying threshold at approach-1

| **Threshold** | **tp** | **tn** | **fp** | **fn** | **Recall** | **FPR** |
|---|---|---|---|---|---|---|
| $Mean$ | 862 | 230 | 58 | 2 | 99.77% | 20.14% |
| $Mean * 2$ | 704 | 278 | 10 | 160 | 81.48% | 03.47% |
| $Mean * 3$ | 639 | 284 | 4 | 225 | 73.96% | 01.39% |
| $Mean * 4$ | 613 | 287 | 1 | 251 | 70.95% | 00.35% |
| $Mean * 5$ | 584 | 288 | 0 | 280 | 67.59% | 00.00% |

to the mean Hellinger Distance obtained by comparing normal DHCP profiles as discussed earlier in Section 4.4.3. We can notice from Table 4.10 that as the threshold for Hellinger Distance increases, FPR decreases but at the same time, Recall also decreases. Thus, the threshold should be chosen wisely to minimize FPR but maximize Recall at the same time.

**Detection Approach-2:** To test the effect of varying predefined threshold on our second detection approach, we used three different thresholds as shown in Table 4.11. The first threshold shown in this table represents that if the number of occur-

Table 4.11: Effect of varying threshold at approach-2

| **Threshold** | **tp** | **tn** | **fp** | **fn** | **Recall** | **FPR** |
|---|---|---|---|---|---|---|
| Expected Count | 757 | 193 | 95 | 2 | 87.62% | 32.99% |
| Twice of Expected Count | 621 | 287 | 1 | 243 | 71.87% | 00.35% |
| Thrice of Expected Count | 581 | 288 | 0 | 283 | 67.25% | 00.00% |

rences of a message type (DHCPDISCOVER for second variant and DHCPDECLINE for first and third variant) in a time interval of testing phase was greater than the expected number of occurrences of the same message type in the same time interval of a day of training phase, Induced DHCP starvation attack was detected in that interval. We can notice from the table that similar to our first approach, FPR and Recall of this approach also decreases as the threshold increases.

### 4.4.7 Testing Various Security Features

As described earlier in Section 4.2.1, security features such as port security, DHCP snooping and DAI available with modern network switches can be used to mitigate DHCP starvation attacks. In this subsection, we describe the experiments performed to test the ability of these security features to detect different variants of Induced DHCP starvation attack.

**Testbed Setup:** We created a testbed setup similar to the one shown in Figure 4.11. This setup was having a HP Aruba 2920 layer-3 managed switch and two computers. One computer was designated as malicious client while other one as victim client. Malicious client was using Ubuntu 16.04LTS while victim client was using Windows 7 operating system. Both malicious and victim clients were having 4 GB of physical memory and Core i5 quad core processor. The switch's built-in DHCP server was enabled and configured with a pool of 256 IP addresses. Using this setup, we tested the security features and the experimental observations are presented in subsequent paragraphs.

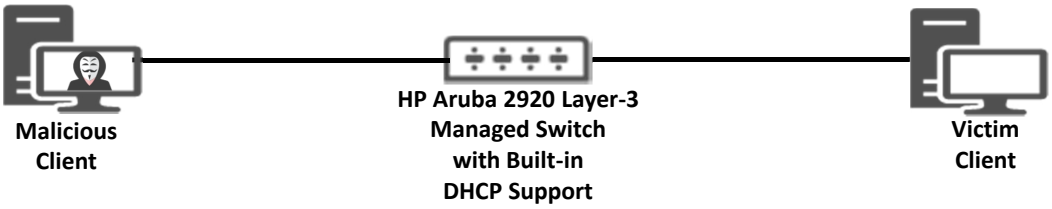



Figure 4.11: Testbed setup for testing security features

**Testing Port Security:** We enabled port security on the switch port to which malicious client was connected by limiting the number of MAC addresses allowed at that port. We also enabled SNMP trap feature in the switch which sends a trap message when port is disabled. After this, we released and then renewed the IP address lease of victim client by executing "ipconfig /release" and "ipconfig /renew" commands on the victim client respectively. After obtaining an IP address from the

built-in DHCP server of the switch, victim client performed conflict checking by sending a broadcast ARP request. On receiving this request, malicious client sent a fake ARP reply. This ARP reply was not dropped by the switch due to which attack was successful and victim client was not able to configure its interface with IP address. Similarly, we tested the second and third variant of Induced DHCP starvation attack and observed that port security was not able to mitigate these two variants also.

**Testing DHCP Snooping:** We enabled DHCP snooping globally on the switch and designated the built-in DHCP server of the switch as an authorized DHCP server. After this, we released and renewed the IP address lease of victim client. As soon as the victim client sent DHCPDISCOVER message to obtain IP address from the DHCP server, the server sent a broadcast ARP request for IP conflict detection. On receiving this request, malicious client sent fake ARP reply. Since DHCP snooping did not drop the fake ARP reply sent by malicious client, proposed attack was successful due to which victim client was not able to configure its interface with an IP address. Similarly, we tested the first and third variant of Induced DHCP starvation attack and observed that DHCP snooping was not able to mitigate these two variants also.

**Testing DAI with DHCP Snooping:** We further tested the DAI security feature on the switch by first configuring switch ports as trusted because ARP messages received on untrusted ports were not forwarded in the network due to which victim client could not perform IP conflict checking. DHCP snooping was also enabled along with DAI as DAI used the trusted DHCP snooping database to check if IP-MAC bindings present in ARP messages were genuine. We then released and renewed the IP address lease of victim client and launched the first variant of Induced DHCP starvation attack from malicious client by sending fake ARP reply in response to the ARP request sent by victim client for conflict checking. However, the switch dropped the ARP reply as the IP-MAC binding induced due to this ARP reply contradicted the binding of the MAC address of the malicious client

and IP allotted to it. Thus, first variant of Induced DHCP starvation attack was mitigated using DAI. Nevertheless, if we consider a network topology similar to the one shown in Figure 4.1, DAI can not detect first variant as the fake ARP replies do not traverse the DAI enabled switch. We also tested the second variant of Induced DHCP starvation attack. However, the fake ARP reply sent by the malicious client in response to the ARP request sent by the DHCP server was dropped by the switch due to which the second variant was also mitigated.

We further launched the third variant of Induced DHCP starvation attack using similar approach. In this case, we observed that the switch did not drop the fake ARP request sent by the malicious client in response to the ARP request sent by the victim client for conflict detection purpose. This was because malicious client configured its interface with a manual IP address and thus, DHCP snooping did not have any binding information of malicious client's MAC address with an IP address. As a result, the third variant of Induced DHCP starvation attack was successful and victim client was not able to configure its interface with IP address.

Table 4.12 summarizes the detection ability of different security features available with network switches to detect Induced DHCP starvation attack variants. A (✓) sign indicates that the security feature can mitigate the corresponding attack variant while a (✗) sign indicates that the security feature can not mitigate the corresponding attack variant. We can notice that port security and DHCP snooping could not mitigate either of the three variants whereas DAI could mitigate only the first variant of Induced DHCP starvation attack.

Table 4.12: Detecting Induced DHCP starvation attack variants using security features available with switches

| ***Attack Variant-*** | **Port Security** | **DHCP Snooping** | **DAI** |
|---|---|---|---|
| 1 | ✗ | ✗ | ✓ |
| 2 | ✗ | ✗ | ✓ |
| 3 | ✗ | ✗ | ✗ |

## 4.5 Conclusion

In this chapter, we proposed three different detection schemes to detect different variants of Induced DHCP starvation attack. The first two approaches involve comparison of DHCP traffic profiles generated during training phase and testing phase while the third approach makes use of various machine learning algorithms to detect different variants of Induced DHCP starvation attack. We evaluated the detection performance of these proposed detection schemes using real DHCP traffic captured from a local network and showed that the proposed detection schemes can detect different variants of Induced DHCP starvation attack with good detection rate. Also we evaluated the detection ability of built-in security mechanisms available with modern network switches in countering induced DHCP starvation attack.

# Chapter 5

# Slow HTTP DoS Attacks against HTTP/1.1 and Detection

## 5.1 Introduction

Hypertext Transfer Protocol (HTTP/1.1) is used by millions of users across the globe for web communication and thus, is one of the core protocols of the Internet. However, there are few known DoS attacks against this protocol impacting its working. Low Rate HTTP DoS attack [56] is an example of DoS attack. In this attack, a malicious client sends a reasonably large number of HTTP requests intelligently to occupy all available HTTP connections that are permitted on a web server. As a result, the server stops processing further requests sent by other genuine clients, resulting in DoS. Several previous works [81, 82, 83, 84, 85, 86] studied the effects of Low Rate HTTP attacks and proposed countermeasures for this. However recently, a relatively newer class of DoS attack known as Slow HTTP DoS attack is discovered. To launch this attack, malicious client creates enough number of connections with the web server and sends specially crafted HTTP requests from each of these connections. Since these connections interact with server very slowly, the server maintains their state information in a connection queue till these connections are completely served. Once all the available space in the connection queue is occupied, no legitimate connections are entertained by server, thereby, causing DoS attack.

In this chapter, we focus on two popular Slow HTTP DoS attacks - *Slow Header Attack* and *Slow Message Body Attack* [87]. We present a detailed study of these attacks by analyzing behaviour of different types of web servers against these attacks and live websites[1] against these attacks. We also describe a statistical abnormality measurement technique to detect these attacks. Our contributions in this chapter are:
**1.** We study the impact of Slow HTTP DoS attacks on widely used web servers and report the results.
**2.** We perform an empirical study of Slow HTTP DoS attacks against a large number of websites falling into 4 categories and furnish the obtained results. These websites are hosted on servers present in different parts of the globe.
**3.** Based on our empirical study, we conclude that Slow HTTP DoS attacks are a real threat and hence believe that detecting these attacks is important.
**4.** We subsequently propose a detection scheme to detect these attacks.
**5.** We evaluate the detection performance of proposed scheme using both simulated HTTP traffic collected from our testbed and real HTTP traffic collected from a public web server.

Rest of this chapter is organized as follows. In Section 5.2, we describe basics of HTTP/1.1 protocol. In Section 5.3, we discuss Slow HTTP DoS attacks and the previously known detection and mitigation techniques. In Section 5.4, we propose an anomaly detection scheme to detect these attacks. We present empirical study of Slow HTTP DoS attacks and experimental evaluation of proposed detection scheme in Section 5.5 followed by conclusion in Section 5.6.

## 5.2 HTTP/1.1 Protocol

HTTP/1.1 is a protocol used for web communication and its working is described in RFC 2616 [16]. HTTP/1.1 is the successor of original HTTP/1.0 protocol and the major difference between these two HTTP versions is that HTTP/1.0 needs to establish a new connection each time a request/response is exchanged between client and

[1]These are our partner websites and are tested as part of vulnerability assessment.

server while HTTP/1.1 has the provision to exchange one or more request/response on a single connection. Some of the features of HTTP/1.1 are as follows:

**1. Connectionless:** After sending a HTTP request to server, client disconnects from the server and waits for a response. The server then processes the request and re-establishes the connection to send the response back.

**2. Staless:** As HTTP is connectionless, it is also a stateless protocol. Both the client and the server are aware of each other only during a current request.

**3. Media Independent:** HTTP/1.1 allows any type of data that can be sent using it as long as both the client and the server know how to process the data. For this purpose, HTTP/1.1 requires that both the client and the server must specify the content type that is being sent.

### 5.2.1 Methods in HTTP/1.1

Table 5.1: HTTP/1.1 methods

| **Method** | **Description** |
|---|---|
| GET | Used to retrieve data from a server using a given URI. |
| POST | Used to send data to the server, for example, file upload, customer information, etc. using a HTML form. |
| HEAD | Similar to GET method except that server sends only response line and headers but not the entity-body. |
| PUT | Used to request a server to store the included entity-body at a location specified by the given URL. |
| DELETE | Used to request a server to delete a file specified by the given URL. |
| CONNECT | Used to establish a network connection to a web server over HTTP. |
| OPTIONS | Used to find out different types of HTTP methods that a web server supports. |
| TRACE | Used to echo the contents of an HTTP request back to the client. This method is typically used for debugging purposes at the time of development. |

RFC 2616 defines different types of methods (or requests) to be performed on the resource identified by the Request Uniform Resource Indicator (URI). Table 5.1

shows a set of these methods and their description. In this chapter, we focus only on *GET* and *POST* requests as the attacks discussed in this chapter are related to these methods.

**1. GET Requests:** HTTP GET request is used to retrieve data, e.g. index page, identified by Request-URI from a web server. HTTP protocol, by its design, requires HTTP headers to be completely received before they are processed. According to RFC 2616, "$\backslash r \backslash n$" denotes the end of a line in HTTP header while "$\backslash r \backslash n \backslash r \backslash n$" denotes the end of a complete HTTP header. Figure 5.1 shows an example of complete HTTP GET header.

```
GET /codes/ HTTP/1.1\r\n
Host: 192.168.0.6\r\n
User-Agent:Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:31.0) Gecko/20100101
Firefox/31.0\r\n
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language:en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Connection: keep-alive\r\n\r\n
```

Figure 5.1: A complete HTTP GET request

**2. POST Requests:** POST request is used to send some data to the web server, e.g. data sent during a form submission. To inform the recipient about the size of

```
POST /form/contact-form-handler.php HTTP/1.1\r\n
Host: 192.168.0.3\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:31.0) Gecko/20100101
Firefox/31.0\r\n
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Referer: http://192.168.0.3/form/contact-form.html\r\n
Connection: keep-alive\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Content-Length: 13\r\n\r\n
message=hello
```

Figure 5.2: A complete HTTP POST request with the message body

the data (in bytes) present in the payload, HTTP defines a *Content-Length* field in the protocol header. Figure 5.2 shows an example of HTTP POST request with the *Content-Length* field showing the size of message is 13 bytes (in the first box in Figure

5.2) and the message itself (in the second box in Figure 5.2).

## 5.3 Prior Work

Various Slow HTTP DoS attacks have been disclosed since last decade. The most popular among them are *Slow Header*, *Slow Message Body* and *Slow Read* attacks. In this section, we first describe these attacks and then elaborate previously known detection and mitigation techniques to counter these attacks.

### 5.3.1 Slow HTTP DoS Attacks

There are three types of Slow HTTP DoS attacks [5] - *Slow Header Attack*, *Slow Message Body Attack* and *Slow Read Attack*. These attacks are described as follows:

**1. Slow Header Attack:** HTTP protocol requires GET headers to be completely received before they are processed [16] by a web server. If an incomplete HTTP GET header is received, the server assumes that the client is operating from a slow Internet connection and thus keeps its resources busy waiting for the rest of the header. In a

```
GET /codes/ HTTP/1.1\r\n
Host: 192.168.0.6\r\n
User-Agent:Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:31.0) Gecko/20100101
Firefox/31.0\r\n
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language:en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Connection: keep-alive\r\n
XYZ: abc\r\n
C:d\r\n
```

Figure 5.3: An incomplete HTTP GET request with bogus header fields

Slow Header attack, a malicious client sends parts of a HTTP GET request header and thus prolongs the connection as whenever server receives parts of HTTP request, it resets the connection expiry timer. In order to generate this attack, the malicious client sends incomplete requests at regular intervals such that these requests do not contain "$\backslash r \backslash n \backslash r \backslash n$" string. Figure 5.3 shows example of an incomplete HTTP GET

header with bogus header fields. From Figure 5.3, we can also notice the absence of "$\backslash r \backslash n \backslash r \backslash n$" sequence in case of incomplete HTTP GET header. The malicious client creates enough connections to web server and sends incomplete HTTP GET requests from each of them to fill the connection queue of the web server so that legitimate clients can not connect to the server. Tools like Slowloris [88] and SlowHTTPTest [89] have modules that can be used to generate this type of attack.

**2. Slow Message Body Attack:** In this attack, the malicious client sends complete HTTP POST header however it sends the message body in very small chunks, thereby, forcing server to wait to receive complete message body. To generate this attack, the malicious client sends HTTP messages having higher *Content-Length* value than the actual size of message body. Figure 5.4 shows such HTTP POST message having

```
POST /form/contact-form-handler.php HTTP/1.1\r\n
Host: 192.168.0.3\r\n
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:31.0) Gecko/20100101
Firefox/31.0\r\n
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
Accept-Language: en-US,en;q=0.5\r\n
Accept-Encoding: gzip, deflate\r\n
Referer: http://192.168.0.3/form/contact-form.html\r\n
Connection: keep-alive\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Content-Length: 200\r\n\r\n
message=hello
```

Figure 5.4: An incomplete HTTP POST request

*Content-Length* value of 200 while the actual message body "message=hello" is of only 13 bytes (characters). When web server receives this message, it assumes that only a part of the message body is received and remaining part is still pending. Thus, server keeps its resources busy waiting for the rest of the message. The malicious client creates enough number of connections to web server and sends such HTTP messages from each of them to fill the connection queue of the web server, thereby, causing a DoS scenario. There are open source tools such as RUDY [90] and SlowHTTPTest [89] that can be used to create this type of attack.

**3. Slow Read Attack:** Slow Read attack against a HTTP/1.1 server requires a malicious client to send a legitimate GET request after TCP 3-way handshake and then immediately advertises a receiver window size of zero byte. As a result, server stops sending data although it holds the connection in the hope of receiving a non-zero window size advertisement from the client. Client on the other hand, never advertises non-zero window size, thus, forcing server to wait for indefinite time. Nevertheless, web server implementations have been patched to immediately close the connection on which a receiver window of size zero is advertised and thus, the servers are no longer vulnerable to Slow Read attack. Thus, in this chapter, we discuss the effects of Slow Header and Slow Message Body Attacks only on different web servers. Moreover, the detection scheme is proposed with keeping in mind only these two attacks.

### 5.3.2 Known Detection and Mitigation Techniques

The research community has invested considerable effort to detect Low Rate HTTP DoS attacks and thus, a comprehensive literature is available on this class of attack [91]. Here, we discuss schemes that focus to combat Slow HTTP DoS attacks.

**1. Anomaly-based Techniques:** Anomaly detection techniques first learn behaviour of a normal traffic pattern and then checks for any significant deviation from this pattern to detect anomalies in the future. There are many anomaly detection techniques that have been proposed to detect anomalies in web traffic by simply looking at deviations from normal flows statistics [57]. Authors of [58] proposed a scheme that extracts traffic features like amount of time required to generate a HTTP request. Based on this feature, a comparison is made between sampling distribution generated from unknown traffic, potentially anomalous, and distribution generated from legitimate HTTP traffic to detect Slow Header attack. However, authors did not discuss how the proposed scheme can be adapted to detect Slow Message Body attack. In another work, Giralte et al. [59] used three types of analyzers wherein the

first analyzer performs statistical analysis of HTTP flow such as request and response size. The second analyzer checks access behaviour of users using a graph, to model the several paths of the web server as well as the different costs of navigating through the server. The last analyzer counts the frequency of HTTP operations carried on the web server by a particular user. However, the proposed technique can not detect Slow HTTP *Distributed* DoS (DDoS) attack if number of involved bots is large and each bot contributes very less traffic to evade detection. Moreover, to avoid detection, a determined adversary can simulate the behaviour of normal users by using techniques such as Web Usage Mining (WUM) which involves the application of machine learning techniques on web data to extract behavioural patterns of web users [60]. The authors of [61, 19] presented a technique that tracks number of packets a web server receives in a given period of time. This feature is monitored in two subsequent temporal periods to detect normal or anomalous behavior. The weakness of this technique lies in the feature selection itself. Monitoring number of packets received at a web server can cause high false positive rate because if a high burst of traffic is received due to normal scenarios like Flash Event [62], the detection technique notifies this as an attack traffic. Moreover, these techniques can not detect Slow HTTP DDoS attacks [63].

**2. Implementation Modules:** Few modules [64, 65, 66, 67] are made available for Apache server to mitigate Slow HTTP DoS attacks. A summary of few mitigation modules and their description is given in Table 5.2. However, these modules either have tendency to block legitimate requests from being served or they are unable to mitigate Slow HTTP DDoS attacks. Moreover, authors of [68] showed that a web server configured with these modules, though, repulsed the attack; there was a noticeable delay in the server's response when attack was launched by 4 bots. Thus, it is possible that a more massive attack scenario can affect the server's performance.

**3. Other Detection and Mitigation Schemes:** In [92], authors proposed neural network models for the analysis of HTTP traffic to detect HTTP DoS attacks. With the help of experimental results, authors concluded that none of the applied neural models were able to differentiate normal from anomalous traffic. Authors of [93] proposed a lightweight method to detect DDoS attack by using various traffic

Table 5.2: Various implementation modules and their description

| **Module** | **Description** |
|---|---|
| Core [64] | This module buffers entire HTTP requests at the kernel level. Once an entire request is received, the kernel then sends it to the server. This ensures that incomplete HTTP requests do not go to the server's connection queue. |
| Antiloris [65] and Limitipconn [66] | This module limits the number of incoming complete/incomplete requests on a per IP basis. |
| mod_reqtimeout [67] | RequestReadTimeout directive of this module allows a server administrator to receive a complete request and/or complete message body within a predefined amount of time. |

flow features such as number of packets per flow, number of bytes transmitted per flow and average duration of each flow. The detection technique was implemented over a NOX-based network, where OpenFlow (OF) switches [94] add statistics about all active flows in a flow table. All feature information needed was accessed in an efficient way by means of a NOX controller [95]. The feature information was then used to train Self Organizing Maps (SOM) [96]. During training phase, SOM created a topological map where different regions represent each kind of traffic. Once SOM was trained, it was then put to use to classify attack and normal traffic.

Although we discussed several previously known defense methods to counter attacks against HTTP protocol, these methods are either not adapted to detect all types of Slow HTTP DoS attack [58] or unable to detect Slow HTTP DDoS attacks launched using several compromised bots [59] or known to generate large number of false positives [61, 19] in case of scenarios such as Flash Event [62]. Thus, in the next section, we describe a detection approach that can detect both types of Slow HTTP DoS attacks with good detection rates even when the attack traffic is coming from different sources.

## 5.4 Proposed Detection Scheme

Since Slow HTTP/1.1 DoS attacks pose a serious threat against a large number of web servers [5] which our empirical study also confirms (see Section 5.5), countering these attacks is important. Thus, we describe an anomaly detection scheme that uses a statistical abnormality measurement technique to detect both Slow Header and Slow Message Body attacks. Our detection scheme involves comparing HTTP traffic profiles (probability distributions) generated during training phase and testing phase on the basis of various request types discussed in the next subsection. In training phase, a normal HTTP profile is generated by collecting legitimate web traffic coming to and going from a server over a period of $n$ observation intervals. In testing phase, detection system compares the current traffic profile with the profile generated during training phase. For this comparison, we calculate Hellinger Distance [18] (See Appendix C) between the training and testing probability distributions to detect the attacks.

### 5.4.1 Request Types

We consider that a HTTP traffic profile comprised of following four types of HTTP requests - *Complete GET*, *Complete POST*, *Incomplete GET* and *Incomplete POST*. In general, a large portion of HTTP traffic consists of complete HTTP headers and complete HTTP message body. A very small portion of HTTP traffic arriving at the server is comprised of incomplete requests. Such incomplete requests are received at the server because of the connections which are not closed gracefully by TCP due to connection errors. Such errors in TCP connections over internet are not rare. In case of Slow HTTP DoS attack, the balance of number of occurrences of different request types is disturbed either due to increased number of incomplete HTTP GET requests in case of Slow Header attack or due to increased number of incomplete POST requests in case of Slow Message Body attack. We exploit this change in observation to detect these attacks. We use following methods to differentiate various request types:

**1. Complete/Incomplete HTTP GET Requests:** If our detection system observes the presence of "$\backslash r\backslash n\backslash r\backslash n$" string in a HTTP GET request, it considers that

request as complete HTTP GET request. Otherwise, request is considered as incomplete GET request.

**2. Complete/Incomplete HTTP POST Requests:** To differentiate between complete and incomplete HTTP POST requests, our detection system compares the value present in *Content-Length* field of HTTP header and actual size of the message body. If both are equal, the request is considered as complete HTTP POST request. Otherwise, request is considered as incomplete POST request.

### 5.4.2 Adapting Hellinger Distance for Slow HTTP DoS attacks Detection

Similar to our first detection approach described in Chapter 4, we use Hellinger Distance [18] to detect Slow HTTP DoS attacks also. Hellinger Distance can easily be adapted to determine if there is a significant deviation in the HTTP traffic profiles generated during training and testing phase. We create traffic profiles during training and testing phases as $\mathcal{P}$ and $\mathcal{Q}$ respectively. Here $\mathcal{P}$ and $\mathcal{Q}$ are $N$ dimensional vectors with each component of the vector representing the probability of an attribute. Hellinger Distance between two profiles $\mathcal{P}$ and $\mathcal{Q}$ is calculated as in Equation 5.1. In this equation, $N$ denotes the number of request types and $P_i$ and $Q_i$ indicate the probability of occurrences of $i^{th}$ type of request in $\mathcal{P}$ and $\mathcal{Q}$ respectively.

$$d_H = (\frac{1}{2}\Sigma_{i=1}^{N}(\sqrt{P_i} - \sqrt{Q_i})^2)^{\frac{1}{2}} \tag{5.1}$$

The value of $d_H$ will be always in between 0 and 1 with 0 representing perfect similarity and 1 representing maximum dissimilarity between $\mathcal{P}$ and $\mathcal{Q}$. Other than Hellinger Distance, there are several popular distance metrics such as Bhattacharyya Distance [97], Total Variation Distance [98], Mahalanobis Distance [99] and Kullback-Leibler Divergence [100]. For detection purpose, we adapted Hellinger Distance rather than other distance metrics as it is light-weight and has natural lower and upper bounds (see Appendix A).

### 5.4.3 Normal HTTP/2 Profile Creation

In order to create a normal HTTP traffic profile, we create a probability distribution using four types of HTTP requests as discussed in previous subsection over a period of $W$ observation intervals, each of duration $\Delta T$. The profile generated in the training phase $\mathcal{P}$ has 4 attributes as $\{P_{GETcomplete}, P_{POSTcomplete}, P_{GETincomplete}, P_{POSTincomplete}\}$ where $P_{GETcomplete}$, $P_{POSTcomplete}$, $P_{GETincomplete}$ and $P_{POSTincomplete}$ represent the probability of occurrence of complete GET, complete POST, incomplete GET and incomplete POST request respectively. Algorithm 5.1 describes the procedure of generating the required training profile $\mathcal{P}$. This algorithm takes the time period $\Delta T$ of the training phase and $W$

**Algorithm 5.1** Training phase

**Input:** $\Delta T$ - Time period in hours
**Input:** $W$ - Number of intervals (each of $\Delta T$ hours)
**Result:** $\mathcal{P}$ - Probability distribution

$Sum = 0$
Create an array $Request\_Count[1, \cdots, 4]$
**for** $w = 1$ to $W$ **do**
    **for** $t = t_{start}$ to $(t_{start} + \Delta T)$ **do**
        $Request \leftarrow$ New request of type $r$ received
        $Request\_Count[r] \leftarrow Request\_Count[r] + 1$ // where $r = 1, \cdots, 4$
    **end for**
**end for**
**for** $r = 1$ to $4$ **do**
    $Sum \leftarrow Sum + Request\_Count[r]$
**end for**
**for** $r = 1$ to $4$ **do**
    $\mathcal{P}[r] \leftarrow \frac{Request\_Count[r]}{Sum}$
**end for**
Return $\mathcal{P}$

time intervals as input and counts the number of occurrences of different request types during $\Delta T$ time interval. This is repeated for every $\Delta T$ time period of the training phase. For example, if $W$=10, the number of occurrences of different request types is counted in all these 10 time intervals. After that, the number of occurrences of different request types is divided by the total number of occurrences of all the request types in these 10 time intervals to obtain the probability of occurrence of different request types in a time interval of the training phase. This probability of occurrence of each request type in a time interval represents the normal HTTP training profile.

### 5.4.4 Testing Phase

Once the system is trained and $\mathcal{P}$ is generated, it can be deployed to detect Slow HTTP/1.1 DoS attacks from $(W+1)^{th}$ interval of duration $\Delta T$. Algorithm 5.2 describes procedure to detect anomalies in different time intervals of the testing phase. This algorithm takes $\Delta T$ time period and probability distribution $\mathcal{P}$ generated during

**Algorithm 5.2** Testing phase

**Input:** $\Delta T$ - Time period in hours
**Input:** $\mathcal{P}$ - Probability distribution generated during training phase
**Output:** Detection of Slow HTTP/1.1 DoS attacks

**while** Not interrupted **do**
  Create an array $Request\_Count\_Test[1, \cdots, 4]$
  $Sum = 0$
  **for** t=$t^{test}_{start}$ to $(t^{test}_{start} + \Delta T)$ **do**
    $Request \leftarrow$ New request of type $r$ received
    $Request\_Count\_Test[r] \leftarrow Request\_Count\_Test[r] + 1$ // where $r = 1, \cdots, 4$
  **end for**
  **for** $r = 1$ to $4$ **do**
    $Sum \leftarrow Sum + Request\_Count\_Test[r]$
  **end for**
  **for** $r = 1$ to $4$ **do**
    $\mathcal{Q}[r] \leftarrow \frac{Request\_Count\_Test[r]}{Sum}$
  **end for**
  $HD \leftarrow$ Calculate Hellinger Distance between $\mathcal{P}$ and $\mathcal{Q}$
  **if** $HD \geq \delta$ **then**
    Attack detected
  **end if**
**end while**

training phase as input and creates a profile $\mathcal{Q}$ after collecting all types of HTTP requests in $\Delta T$. The algorithm then calculates the Hellinger Distance between $\mathcal{P}$ and $\mathcal{Q}$ and if the calculated distance is greater than a predefined threshold $\delta$, attack is detected. Figure 5.5 shows the steps to generate a profile during a particular time in-

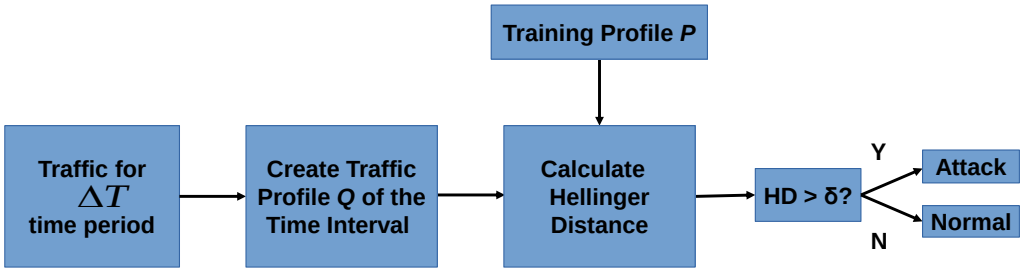



Figure 5.5: Detector working

terval of testing phase and compare it with the normal training profile using Hellinger

Distance. If the calculated Hellinger Distance crosses threshold $\delta$, the detection scheme detects HTTP attack in that time interval.

## 5.5 Experimental Results

We tested impact of Slow HTTP DoS attacks discussed in Section 5.3 not only against four popular HTTP/1.1 web servers but also against 100 live websites hosted on web servers present in different parts of the globe. In this section, we first report the impact of these attacks against different web servers and 100 live websites belonging to four different categories. Subsequently, we also present the experiments performed to evaluate detection performance of proposed anomaly detection system to detect Slow HTTP DoS attacks.

### 5.5.1 Severity of Slow HTTP DoS Attacks

**Testbed Setup:** To evaluate the effectiveness of Slow HTTP DoS attacks against different web servers, we selected four servers - Apache, Microsoft IIS, Nginx and Lighttpd which are amongst most popular web servers according to the Netcraft's survey [101]. The versions of web servers we used for this study were Apache 2.4.18, IIS 7.5, Nginx 1.4.6 and Lighttpd 1.4.33. All these servers were tested in their default configuration. Apache and IIS web servers were installed on a Windows 7 Home Premium machine having Intel Core 2 Duo processor with 4 GB of physical memory while NGinx and Lighttpd servers were installed on a Linux Mint machine having AMD Athlon X2 270 Dual core processor with 4 GB of physical memory. One of the clients running Ubuntu 14.04 having Intel Core 2 Duo processor with 1 GB of physical memory was designated as malicious client. This malicious client was configured with SlowHTTPTest framework [89]. We also designated one computer as genuine client that was used to check the server's availability while attack was going on. We developed a small website with 10 pages and few images and couple of forms with 5 text fields for testing purpose. For each type of Slow HTTP DoS attack, the observations on each of the four web servers were recorded. Findings of Slow HTTP

attack with GET and POST methods are summarized in Table 5.3.

Table 5.3: Vulnerability assessment for different web servers

| **Web Server** | **Slow Header** | **Slow Message Body** |
|---|---|---|
| Apache | Vulnerable | Vulnerable |
| IIS | Not Vulnerable | Vulnerable |
| Nginx | Not Vulnerable | Not Vulnerable |
| Lighttpd | Vulnerable | Vulnerable |

**Launching Slow HTTP/1.1 DoS Attacks:** We used SlowHTTPtest framework configured at the malicious client to first launch Slow Header attack against Apache web server. SlowHTTPtest framework was passed three parameters - i) Establish 1000 connections at ii) the rate of 100 connections per second with iii) the *"IP address"* of the web server. While attack was going on, we sent GET requests from the genuine client to the target web server in order to check the server's availability. We conducted similar experiments by taking other web servers into account. We present the observations made from these experiments in next four paragraphs.

**Apache Web Server:** We observed that Apache web server closes a connection if it does not receive any HTTP header information from that connection for 300 seconds. However, if malicious client continues to send bogus header fields at regular intervals, the server keeps waiting to receive complete HTTP GET Request for up to 990 seconds. After this timeout, the server responds with a 400 *Bad Request* message and finally closes the connection. This large waiting time of 990 seconds in the hope of receiving complete HTTP GET request makes Apache web server vulnerable to Slow Header attack. In case of Slow Message Body attack, we observed that Apache waits for 300 seconds to receive complete message body. However, if malicious client continues to send chunks of message body at regular intervals, Apache web server waits for indefinite amount of time to receive complete message body. As

a result, Apache web server is vulnerable to Slow Message Body attack also[2]. Table 5.4 summarizes the behaviour of Apache web server against Slow HTTP DoS attacks.

Table 5.4: Behaviour of Apache web server against Slow HTTP DoS attacks

| **Parameter** | **Slow Header** | **Slow Message Body** |
|---|---|---|
| Minimum Waiting Time | 300 | 300 |
| Maximum Waiting Time | 990 | Indefinite |
| Number of Connections to create DoS if Vulnerable | 150 | 150 |

**IIS Web Server:** In case of Microsoft's IIS web server, we observed that it closes a connection if it does not receive any data from that connection for 130 seconds. Even if malicious client continues to send bogus header fields at regular intervals, the server waits for just around 130 seconds to receive complete HTTP GET Request and then closes the connection. Thus, in theory, IIS should be vulnerable to Slow Header attack. However, when we sent GET requests from the genuine client while the attack was going on, server was able to properly serve the request and send back the response. This was because IIS does not consider incomplete requests as writable until complete header is received. Such connections are not transferred to server's internal processing queue. As a result, IIS web server is not vulnerable to Slow Header attack. In case of Slow Message Body attack, since complete header (but incomplete message body) of POST requests is sent, these requests are passed to server's internal processing queue. On receiving these requests, we observed that IIS waits for 130 seconds to receive complete message body. We also observed that if malicious client continues to send chunks of message body at regular intervals, IIS web server keeps waiting for indefinite amount of time to receive complete message body. As a result, IIS is vulnerable to Slow Message Body attack. Table 5.5 summarizes the behaviour of IIS web server against Slow HTTP DoS attacks.

[2]We considered a web server vulnerable to the attacks if it waited for more than 30 seconds before closing the connections from which incomplete requests/message bodies were sent.

Table 5.5: Behaviour of IIS web server against Slow HTTP DoS attacks

| **Parameter** | **Slow Header** | **Slow Message Body** |
|---|---|---|
| Minimum Waiting Time | 130 | 130 |
| Maximum Waiting Time | 130 | Indefinite |
| Number of Connections to create DoS if Vulnerable | Not Vulnerable | 100 |

**Nginx Web Server:** In case of Nginx web server, we observed that it closes a connection if it does not receive complete HTTP header within 60 seconds. In case of Slow Message Body attack, we observed that Nginx waits for 60 seconds to receive complete message body. However, if malicious client continues to send chunks of message body at regular intervals, the web server keeps waiting for indefinite amount of time to receive complete message body. Thus, in theory, it should be vulnerable to both Slow Header and Slow Message Body attack. However, when we sent GET requests from the genuine client while the attack was going on, server was able to properly serve the request and send back the response. The reason is Nginx uses a highly scalable event-driven asynchronous architecture that makes use of small but predictable amounts of memory under load. According to Nginx technical

Table 5.6: Behaviour of Nginx web server against Slow HTTP DoS attacks

| **Parameter** | **Slow Header** | **Slow Message Body** |
|---|---|---|
| Minimum Waiting Time | 60 | 60 |
| Maximum Waiting Time | 60 | Indefinite |
| Number of Connections to create DoS if Vulnerable | Not Vulnerable | Not Vulnerable |

specifications guide [102], it can handle upto 1 million concurrent connections. Also, Nginx handles requests using a single (or at least, very few) thread, thus consumes very less RAM unlike process-based servers like Apache that requires a thread for each simultaneous connection. Due to these reasons, Nginx is not vulnerable to both

Slow Header and Slow Message Body attack. Table 5.6 summarizes the behaviour of Nginx web server against Slow HTTP DoS attacks.

**Lighttpd Web Server:** While analyzing the behavior of lighttpd web server, we observed that it closes a connection if it does not receive any data from that connection for around 60 seconds. However, if malicious client continues to send bogus header fields at regular intervals, the server keeps waiting for indefinite time to receive complete HTTP GET Request. This indefinite waiting to receive complete HTTP header makes lighttpd vulnerable to Slow Header attack. In case of Slow Message Body attack, we observed that if malicious client continues to send chunks of message body at regular intervals, lighttpd web server keeps waiting for indefinite amount of time to receive complete message body. As a result, lighttpd web server is vulnerable to Slow Message Body attack too. Table 5.7 summarizes the behaviour of Nginx web server against Slow HTTP DoS attacks.

Table 5.7: Behaviour of Lighttpd web server against Slow HTTP DoS attacks

| **Parameter** | **Slow Header** | **Slow Message Body** |
|---|---|---|
| Minimum Waiting Time | 60 | 60 |
| Maximum Waiting Time | Indefinite | Indefinite |
| Number of Connections to create DoS if Vulnerable | 480 | 480 |

### 5.5.2 Empirical Evaluation of Slow HTTP DoS Attacks

In order to assess the impact of Slow HTTP attacks in the wild, we conducted a pilot study on 100 websites from our partners as part of a vulnerability assessment exercise. These websites are hosted on different types of web servers and are located in different parts of the world. For this pilot study, we used only Slow HTTP GET attack as HTTP POST request requires a custom message to be used (the number of fields and their types are different for each form). The websites chosen

for this study were carefully selected from four different categories - i) Category-1: Educational institutions' websites, ii) Category-2: Scientific organizations' websites, iii) Category-3: Banking organization's websites and iv) Category-4: News' websites and we selected 25 websites from each of these four categories.

**Testbed Setup:** For the evaluation, we used two machines - one as malicious client to launch the attack and another genuine client to check availability of website under consideration during attack period. The malicious client was configured with SlowHTTPTest framework [89] and the availability testing client was configured with one of the most popular benchmarking software, JMeter [103]. This software is an open source pure Java application designed by Apache itself for the purpose of load and performance testing. It simulates virtual users and generates web traffic towards a web server on which performance testing is going on. We connected these two clients to Internet using services of two different Internet Service Providers (ISPs). Thus, we used different Internet connections to mount the attack and test the servers' ability to serve request while the attack is under progress. This was because of possible presence of a Web Application Firewall (WAF) [104] at server side that may blacklist an IP address from which it receives number of connections more than a particular threshold. Using SlowHTTPTest [89], we launched Slow Header attack on each of these 100 websites at a rate of 200 connections per second for a time period of 1 minute. Simultaneously, Jmeter was started on genuine client for testing availability of these websites during this period. We considered all those websites as vulnerable which suffered either from Reduction of Quality (RoQ) of service or DoS. From this experiment, we observed that the most vulnerable class of websites was category-1 with 14 out of 25 websites vulnerable. Each of category-2 and category-3 had 8 out of 25 websites vulnerable. Category-4 was the least vulnerable with only 4 out of 25 websites vulnerable. Figures 5.6a, 5.6b, 5.6c and 5.6d show pie-charts for DoS vulnerability assessment of websites belonging to category-1, category-2, category-3 and category-4 respectively. In summary, we can notice that a large portion of web servers deployed in the wild are still vulnerable to Slow HTTP DoS attacks. Thus,

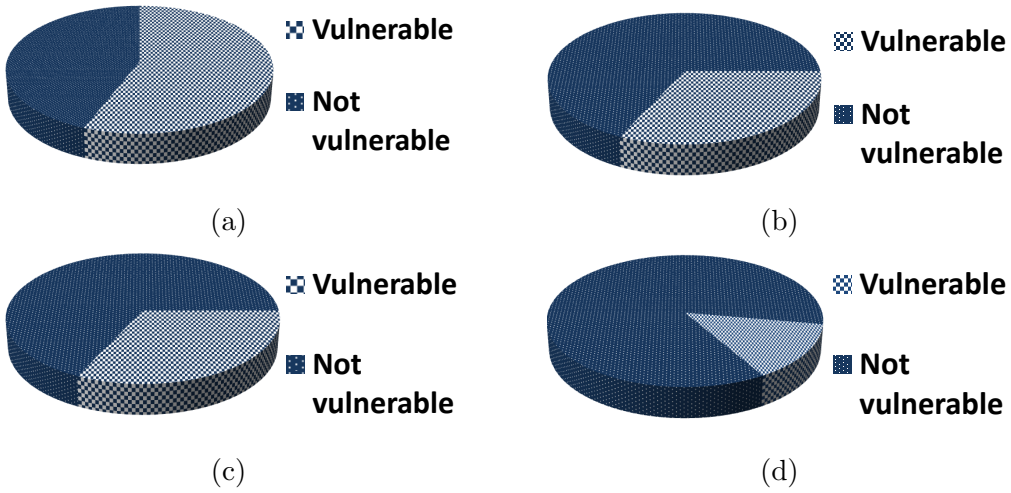



Figure 5.6: (a) Educational, (b) Banking, (c) Scientific and (b) News websites

given the ease with which these attacks can be launched, it is important to detect these attacks.

### 5.5.3 Comparison between Slow HTTP DoS Attacks and Low Rate HTTP DoS Attack

Both Low Rate HTTP DoS and Slow HTTP DoS attacks are aimed at creating DoS scenario. In Low Rate HTTP DoS attack, complete HTTP requests are sent with the intention of being served. However, in case of Slow HTTP DoS attack, incomplete HTTP requests are sent with bogus headers with the intention of forcing server to wait for complete HTTP requests. Though both these attacks are intended to consume available connection queue space, Slow HTTP DoS attacks are more effective as compared to Low Rate HTTP DoS attacks. This is due to the fact that after serving a request, the amount of time for which a web server waits[3] before closing the connection is lesser than the amount of time it waits to receive a complete HTTP request. For instance, Apache web server after serving a request, waits for 26 seconds before closing the connection. On the other hand, it waits for around 300 seconds to receive a complete HTTP request. As a result, connections with partial HTTP

[3]Here we assume persistent HTTP connections.

requests are able to occupy the connection queue space for a longer time, thereby making Slow HTTP DoS attacks more effective.

### 5.5.4 Experiments with Anomaly Detection Scheme

In Section 5.4, we presented our anomaly detection scheme to detect Slow HTTP DoS attacks. In this subsection, we describe the experiments conducted to test the performance of proposed detection scheme and also present the obtained results. For the experiments, we used two different sources for HTTP data collection. Our first source was a web server present in one of our departmental network which receives HTTP traffic by simulating few virtual users. The second source was our institution's public web server that hosts institute official website. This server receives HTTP traffic from several clients present in different time zones. In the next few subsections, we present the detection performance of proposed scheme using simulated and real HTTP traffic and subsequently, we compare our scheme with some of the previously known schemes to detect Slow HTTP DoS attacks.

#### 5.5.4.1 Detection Performance using Simulated HTTP Traffic

In this subsection, we first describe our testbed setup used to collect simulated HTTP traffic and then discuss the detection performance of proposed approach on the resulting dataset.

**Testbed Setup:** We collected simulated HTTP traffic from a server present in one of our departmental network. This machine was running Apache server software v2.4.17 on Linux Mint operating system and having AMD Athlon X2 270 Dual core processor with 4 GB of physical memory. We designed a sample website having 7 web pages and 1 HTML form with 9 fields and hosted it on the web server. In the same network, we designated one of the client as virtual users simulator that generates legitimate HTTP traffic towards configured web server. This client was running Windows 7pro (service pack 1) having Intel Core2 Duo processor with 4 GB

of physical memory. This client was equipped with Apache JMeter benchmarking tool to simulate 150 concurrent virtual users accessing the sample website. Each of the virtual user was configured to generate 8 HTTP GET and 1 HTTP POST request. Moreover, all these virtual users were configured to use non-persistent HTTP connections. As a result, once request from a connection was served, the client closed the connection without any delay. Also, these virtual users were kept in an uninterrupted loop so that they continued to send HTTP requests unless they were manually terminated. JMeter provides various timer modules to integrate delay mechanisms while simulating virtual users. We used one such timer module called Uniform Random Timer. This timer generates time delays according to the Equation 5.2.

$$Delay = \alpha * \beta + \gamma \tag{5.2}$$

where $\alpha$ is a randomly generated value between 0 and 1 (inclusive), $\beta$ is the Random Delay Maximum and $\gamma$ is the Constant Delay Offset. Table 5.8 shows the different values of these variables that we used for each of the HTTP requests. We used default

Table 5.8: Values for uniform delay timer

| **Request** | $\beta$ **(in seconds)** | $\gamma$ **(in seconds)** |
|---|---|---|
| HTTP GET Request-1 | 101 | 6 |
| HTTP GET Request-2 | 101 | 5 |
| HTTP GET Request-3 | 101 | 8 |
| HTTP GET Request-4 | 101 | 6 |
| HTTP GET Request-5 | 101 | 11 |
| HTTP GET Request-6 | 101 | 5 |
| HTTP GET Request-7 | 101 | 16 |
| HTTP GET Request-8 | 101 | 21 |
| HTTP POST Request | 101 | 26 |

value of $\beta = 101$ and was kept constant for all HTTP requests and in order to make these virtual users behave quite similar to real users, we chose values of $\gamma$ according to the web page size. If size of the requested web page of the sample website is larger, a

higher value of $\gamma$ was chosen. For example, a real user, in general, needs more time to grasp contents of a webpage having more information as compared to a web page with very small information [105]. Also, a real user needs more time to fill a HTML form and so, we chose a higher value, $\gamma = 26$ for HTTP POST request as shown in Table 5.8.

**Simulated HTTP Traffic for Training Phase:** From the testbed setup described earlier, we collected 10 hours of simulated HTTP traffic. We used the first 5 hours' traffic for training purpose i.e. to create distribution profile $\mathcal{P}$. The details of the training dataset are shown in Table 5.9. We set the interval duration $\Delta T$ to 10 minutes for our experiments. Thus, there were a total of 30 such intervals from 5 hours of training data. We created the probability distribution graph using these 30 interval data which represent the normal HTTP traffic. Figure 5.7 shows the distribution graph generated from this data. We can notice that probability of incomplete GET and incomplete POST requests is 0 because no incomplete requests were seen in training period. Moreover, we also notice that complete GET requests appeared in majority as compared to complete POST requests, thus, having higher probability. This was due to the large number of pages which had objects and other

Table 5.9: Simulated HTTP training and testing dataset

| **Details** | **Training Dataset** | **Testing Dataset** |
|---|---|---|
| Number of intervals without attack | 30 | 28 |
| Number of intervals with Slow HTTP DoS attacks | 0 | 2 |
| Total number of time intervals | 30 | 30 |
| Time period (in minutes) | 10 | 10 |
| Number of complete GET requests | 38982 | 38836 |
| Number of complete POST requests | 4719 | 4772 |
| Number of incomplete GET requests | 0 | 600 |
| Number of incomplete POST requests | 0 | 600 |
| Total number of requests | 43701 | 44808 |

details and there was only one page having a form which used POST method.

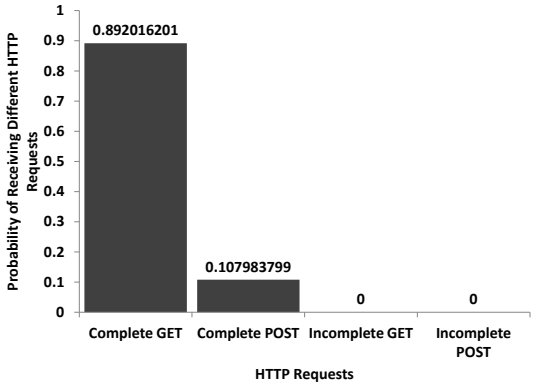



Figure 5.7: Training profile generated from simulated HTTP traffic

**Simulated HTTP Traffic for Testing Phase:** We used remaining five hours' normal HTTP traffic for testing purpose. The details of the testing dataset are also shown in Table 5.9. Similar to training phase, we used intervals of $\Delta T = 10$ minutes in testing phase too. As a result, there were a total of 30 such intervals from these five hours' normal HTTP traffic. Moreover, in two time intervals of testing phase, we launched Slow Header attack and Slow Message Body attack using SlowHTTPTest framework [89] at a rate of 1 connection per second. The sample probability distributions generated from testing an interval without any attack, with Slow Header attack and with Slow Message Body attack are shown in Figure 5.8a, Figure 5.8b and Figure 5.8c respectively. We can notice that Figure 5.8a is almost similar to the profile shown in Figure 5.7. Subsequently, we measured the Hellinger Distance between probability distributions generated during training phase and different intervals of testing phase. Table 5.10 shows the obtained minimum and maximum Hellinger Distances between the training profile and the profiles with normal traffic and traffic of Slow Header and Slow Message Body attack launched during testing phase. We can notice that the Hellinger Distance between two normal profiles is small and it is higher when the intervals contain attack traffic. To detect Slow HTTP DoS attacks, we chose the thrice of mean Hellinger Distance obtained by comparing normal HTTP profiles as the threshold. To obtain the mean Hellinger

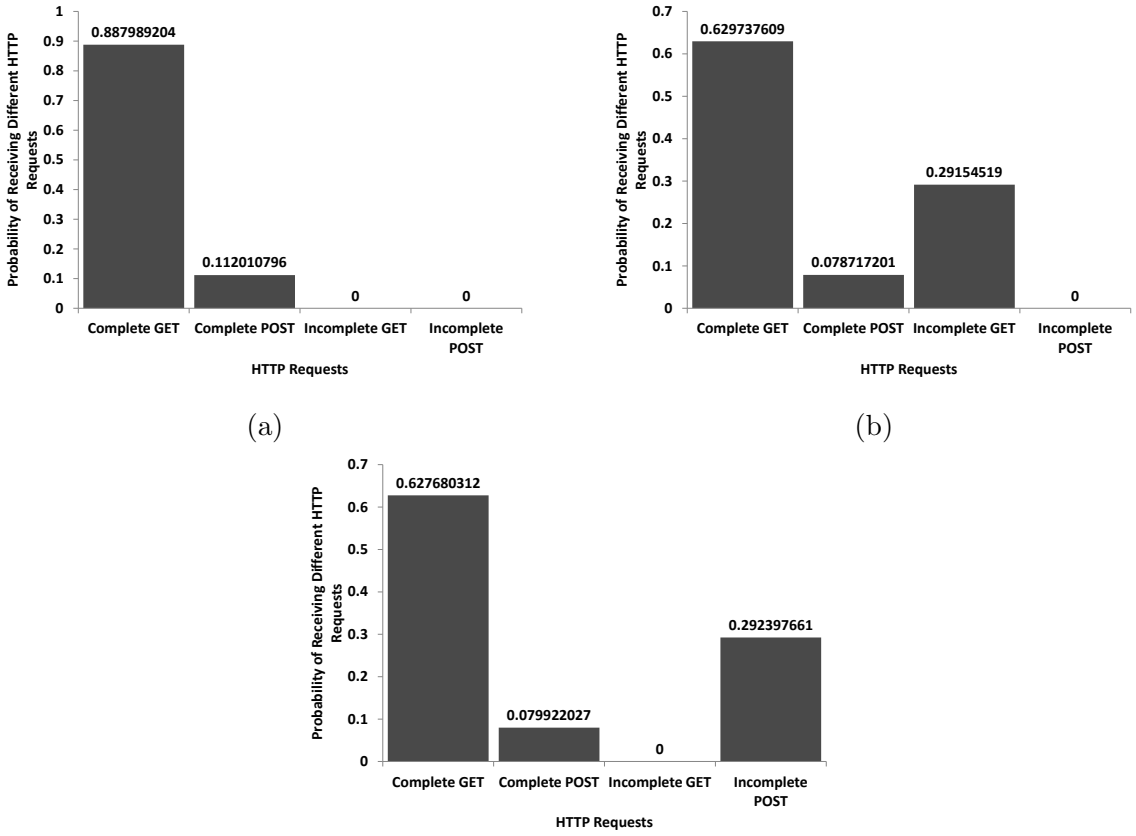



Figure 5.8: Profile with (a) Normal simulated, (b) Slow Header attack and (c) Slow Message Body attack traffic

Distance, we first collected 1 more hour of simulated normal HTTP traffic, divided it into smaller intervals of 10 minutes each and created HTTP profiles in each time interval. We then calculated Hellinger Distances between training profile and profiles generated during these time intervals. In this way, we calculated mean and used it as the threshold. Similar to the previous chapter, we calculated the detection performance of the proposed approach in terms of $tp$, $tn$, $fp$, $fn$, Recall and FPR. The $tp$, $tn$, $fp$, $fn$, Recall and FPR obtained were 2, 28, 0, 0, 100.00 % and 0.00 % respectively.

Table 5.10: Min/max Hellinger Distances

| **Scenario** | **Number of Intervals** | **Min/Max Hellinger Distance** |
|---|---|---|
| Intervals containing normal traffic | 28 | 0.0006/0.0484 |
| Interval containing Slow Header attack traffic | 1 | 0.3979/0.3979 |
| Interval containing Slow Message Body attack traffic | 1 | 0.3985/0.3985 |

#### 5.5.4.2 Detection Performance using Real HTTP Traffic

As mentioned earlier in Section 5.5.4, we chose our institute's public web server that hosts institute's official website as our second source to collect HTTP traffic. We collected traces from this web server so that we could obtain real HTTP traffic over Internet coming from clients all over the globe. This machine was running Apache web server software v2.2.22 on RHEL 6 having Intel Xeon CPU E5-26650 2.4GHz with 8 GB physical memory.

**Real HTTP Traffic for Training Phase:** We collected 16 hours of real HTTP traffic from institute's public web server and used first 8 hours' traffic for training purpose. The details of the training dataset are shown in Table 5.11. In this

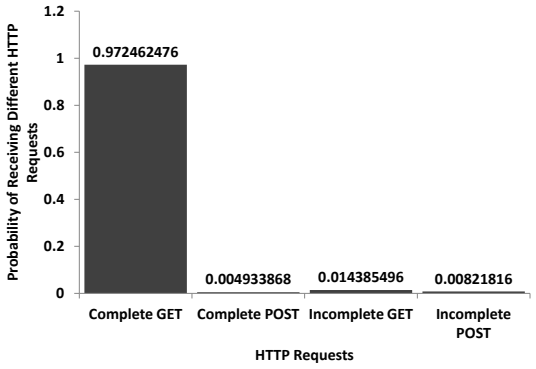



Figure 5.9: Training profile generated from real HTTP traffic

case also, we set the interval duration $\Delta T$ to 10 minutes and thus obtained a total of 48 training intervals. Figure 5.9 shows the distribution graph generated from this

data. We can notice that, unlike simulated HTTP traffic, the real HTTP traffic has few incomplete GET and POST requests as well. This is because of such connections which are not closed gracefully by TCP following some connection error due to which complete GET or POST requests could not be received by server. Such errors in TCP connections over internet are not rare.

Table 5.11: Real HTTP training and testing dataset

| **Details** | **Training Dataset** | **Testing dataset** |
|---|---|---|
| Number of intervals without attack | 48 | 46 |
| Number of intervals with Slow HTTP DoS attacks | 0 | 2 |
| Total number of time intervals | 48 | 48 |
| Time period (in minutes) | 10 | 10 |
| Number of complete GET requests | 65437 | 58065 |
| Number of complete POST requests | 332 | 344 |
| Number of incomplete GET requests | 968 | 1126 |
| Number of incomplete POST requests | 553 | 1005 |
| Total number of requests | 67290 | 60540 |

**Real HTTP Traffic for Testing Phase:** We used remaining eight hours' normal real HTTP traffic for testing purpose. Similar to training phase, we used intervals of $\Delta T = 10$ minutes in testing phase too. As a result, there were a total of 48 such intervals from these eight hours' out of which, in two intervals, we launched Slow Header attack and Slow Message Body attack using SlowHTTPTest framework [89] at a rate of 1 connection per second. Thus there are total of 46 normal HTTP traffic intervals and 2 anomalous intervals. The details of the testing dataset are also shown in Table 5.11. The sample probability distributions generated from an interval without any attack, with Slow Header attack and with Slow Message Body attack are shown in Figure 5.10a, Figure 5.10b and Figure 5.10c respectively. We can notice that Figure 5.10a is almost similar to the profile shown in Figure 5.9. Subsequently, we

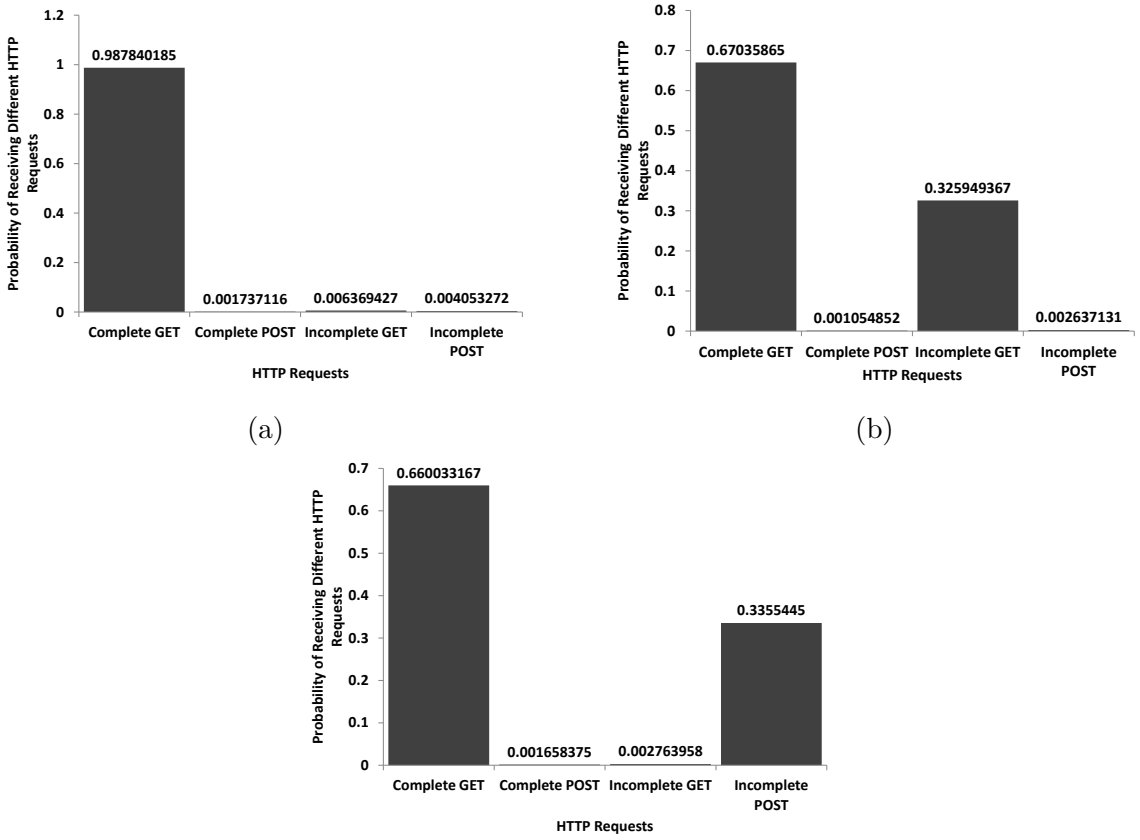



Figure 5.10: Profile with (a) Normal real, (b) Slow Header attack and (c) Slow Message Body attack traffic

measured the Hellinger Distances between probability distributions generated during training phase and different intervals belonging to testing phase. Table 5.12 shows the obtained minimum and maximum Hellinger Distances between the training profile and the profiles with normal traffic and traffic of Slow Header and Slow Message Body attack launched during testing phase. In case of real HTTP traffic also, we chose the thrice of mean Hellinger Distance obtained by comparing normal HTTP profiles as the threshold. To obtain this mean Hellinger Distance, we first collected 1 more hour of real normal HTTP traffic, divided it into smaller intervals of 10 minutes each and created HTTP profiles in each time interval. We then calculated Hellinger Distances between training profile and profiles generated during these time intervals. In this way, we calculated mean and used it as the threshold. The $tp$, $tn$, $fp$, $fn$,

Table 5.12: Min/max Hellinger Distances

| **Scenario** | **Number of Intervals** | **Min/Max Hellinger Distance** |
|---|---|---|
| Intervals containing normal traffic | 46 | 0.0191/0.1273 |
| Interval containing Slow Header attack traffic | 1 | 0.3423/0.3423 |
| Interval containing Slow Message Body attack traffic | 1 | 0.3704/0.3704 |

Recall and FPR obtained were 2, 46, 0, 0, 100.00 % and 0.00 % respectively.

#### 5.5.4.3 Sensitivity Analysis

The detection performance of the proposed scheme depends on the rate with which Slow HTTP DoS attacks are launched, the time period ($\Delta T$) chosen during training and testing phase and also the predefined threshold. In this subsection, we first describe the effects of varying the attack rate and time period on the detection performance of proposed scheme and subsequently, we also describe the effect of varying threshold on their detection performance.

**Varying Attack Rate and Time Period:** We performed an experiment by varying attack launching rate and the time period $\Delta T$ to analyze the sensitivity of the proposed scheme to these two parameters. We collected another 10 hours of traffic from the public web server by injecting both Slow Header and Slow Message Body attack at five different rates as shown in Table 5.13. Thus, there were 5 different scenarios with 2 hours of mixed traffic collected in each scenario. We varied the time intervals in step size of 5 minutes ranging from 5 minutes to 15 minutes and calculated Hellinger Distances between profiles generated during these time intervals and the training profile. Figure 5.11 shows the effect of varying attack rate and time period on the detection performance of the scheme in terms of Recall. We can notice that Recall increases with the increase in attack rate independent of time period. A Recall of 100% was achieved when the attack rate was greater than or equal to 8 incomplete

Table 5.13: Different attack rates for sensitivity analysis

| **Scenario** | **Slow Header Attack Rate** | **Slow Message Body Attack Rate** | **Overall Rate (requests/minute)** |
|---|---|---|---|
| 1 | 1 incomplete GET request after every 30 seconds | 1 incomplete message body after every 30 seconds | 4 incomplete requests/minute |
| 2 | 1 incomplete GET request after every 15 seconds | 1 incomplete message body after every 15 seconds | 8 incomplete requests/minute |
| 3 | 1 incomplete GET request after every 10 seconds | 1 incomplete message body after every 10 seconds | 12 incomplete requests/minute |
| 4 | 1 incomplete GET request after every 5 seconds | 1 incomplete message body after every 5 seconds | 24 incomplete requests/minute |
| 5 | 1 incomplete GET request after every 2 seconds | 1 incomplete message body after every 2 seconds | 60 incomplete requests/minute |

requests/minute and the time period was 15 minutes. We can also notice that when the time period was 5 and 10 minutes, 100% recall was achieved when the attack rate was greater than or equal to 12 incomplete requests/minute. Moreover, when the attack rate was fixed at 8 incomplete requests/minute, higher Recall was achieved when time period was 15 minutes. Thus, a larger time period improves the detection performance of the scheme. However, choosing a larger time period results into late detection of possible ongoing attacks. Thus, time period should be chosen wisely.

**Varying Predefined Threshold:** To test the effect of varying the predefined threshold on the detection performance of proposed scheme, we collected 12 more hours of real HTTP traffic from the public web server. While collecting the traffic, we also launched Slow Header and Slow Message Body attack for 4 hours at the rate of 1 request per second. We set the interval duration $\Delta T$ to 10 minutes and thus, obtained a total of 72 time intervals. Out of these 72 intervals, there were 48 normal intervals and 24 attack intervals respectively. We used five different thresholds and in

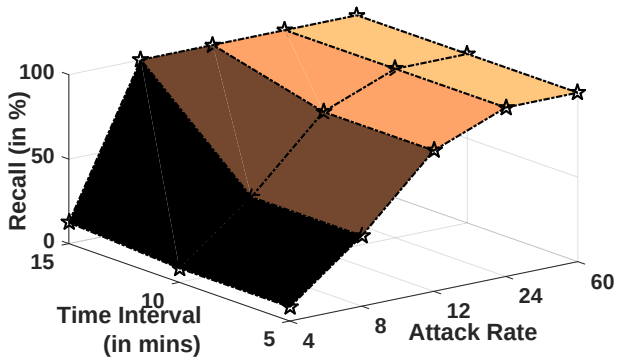



Figure 5.11: Effect of varying time period and attack rate

each case, we calculated the detection performance of proposed detection approach as shown in Table 5.14. In this table, *Mean* corresponds to the mean Hellinger Distance

Table 5.14: Effect of varying threshold

| **Threshold** | **tp** | **tn** | **fp** | **fn** | **Recall** | **FPR** |
|---|---|---|---|---|---|---|
| $Mean$ | 24 | 29 | 19 | 0 | 100% | 39.58% |
| $Mean * 2$ | 24 | 42 | 6 | 0 | 100% | 12.50% |
| $Mean * 3$ | 21 | 48 | 0 | 3 | 87.50% | 00.00% |
| $Mean * 4$ | 18 | 48 | 0 | 6 | 75.00% | 00.00% |
| $Mean * 5$ | 16 | 48 | 0 | 8 | 66.67% | 00.00% |

obtained by comparing normal HTTP profiles as discussed earlier in Section 5.5.4.2. We can notice from Table 5.14 that as the threshold for Hellinger Distance increases, FPR decreases but at the same time, Recall also decreases. Thus, the threshold should be chosen wisely to minimize FPR but maximize Recall at the same time.

#### 5.5.4.4 Comparison of Proposed Detection Scheme with Other Approaches

We compare our detection method with recent anomaly detection schemes [58, 59, 61, 19] proposed in the literature to detect DoS attacks against HTTP protocol. Table 5.15 shows the summary of comparison of our proposed detection technique with these schemes. In [58], authors proposed a scheme that extracts traffic features

like amount of time required to generate a HTTP request and based on this feature, a comparison is made between distribution generated from unknown traffic and legitimate HTTP traffic to detect Slow Header attack. However, authors did not discuss

Table 5.15: Comparison of our scheme with other related works

| **Work** | **Method** | **Strength/Weakness** |
|---|---|---|
| M. Aiello et al.[58] | Compares the time taken to complete a HTTP request in normal and attack scenario. | Can not detect Slow Message Body attack |
| L. C. Giralte et al. [59] | Compares a real user's web access behaviour with web access behaviour of bots. | Can not detect Slow HTTP DDoS attack. |
| M. Aiello et al. [61, 19] | Compares number of packets received in a time period during normal and attack scenario. | High false positive rates in case of Flash Event |
| Proposed scheme | Compares probability of receiving complete and incomplete GET and POST requests in a time period during normal and attack scenario. | High detection accuracy and low false positive rate. |

how the proposed scheme can be adapted to detect Slow Message Body attack. On the other hand, our proposed detection approach can detect both types of DoS attacks as discussed earlier in Section 5.5.4. In another work, Giralte et al. [59] used different analyzers to identify deviations in a user's web access behaviour pattern in order to detect attacks. However, if HTTP DDoS attacks are launched from several compromised bots such that each bot contributes very less traffic, this scheme can not detect the attacks as the web access behaviour pattern of a user remains same (or slightly changed) in this case. Since our proposed detection approach analyzes the overall HTTP traffic received at a web server instead of analyzing the traffic coming from an individual source, it can detect Slow HTTP DDoS attacks also. In [61, 19], authors presented a technique that tracks number of packets a web server receives in

a given period of time. However, monitoring number of packets received at a web server can cause high false positive rate because if a high burst of traffic is received due to normal scenarios like Flash Event [62], the detection technique notifies this as an attack traffic. In case of such Flash Events, the overall HTTP traffic coming to a web server increases due to which the probability of occurrence of a particular type of request (complete/incomplete GET/POST request) in a time interval does not vary much. As a result, our proposed approach does not raise false alarms if the overall traffic on a web server increases. From this comparison, we can conclude that the proposed detection scheme is better equipped to detect the Slow HTTP DoS attacks.

## 5.6 Conclusion

Slow HTTP DoS attacks are a serious threat to web services. In this chapter, we presented an empirical study of Slow HTTP DoS attacks against different web servers and some live websites against Slow HTTP DoS attacks. We presented results to show that Slow HTTP DoS attacks are still a threat to Internet despite of few known detection techniques. Subsequently we proposed a statistical abnormality measurement based detection system that measures Hellinger Distance between two probability distributions generated during training phase and testing phase. This probability distribution comprises of complete and incomplete GET and POST requests. During attack period, the proportion of incomplete requests increases in the overall traffic that causes the corresponding probability distribution deviated from the normal probability distribution, resulting into anomaly. We evaluated the effectiveness of proposed detection technique by collecting simulated HTTP traffic from a web server that received traffic from simulated users and real HTTP traces collected from a public web server too. Our experimental results show that proposed system detects Slow Header and Slow Message Body attacks with high accuracy.

# Chapter 6

# Slow Rate DoS Attacks against HTTP/2 and Detection

## 6.1 Introduction

With the development of highly efficient network infrastructure, research in networking community is now focused towards the development of robust application layer protocols which can utilize the potential and capability of underlying infrastructure to the fullest. It is argued that protocols such as HTTP/1.0 [106] (obsolete now) and HTTP/1.1 [16] are not able to efficiently utilize TCP [17] transmission capacity. In particular, HTTP/1.0 allowed only one request that can be outstanding at a time on an established TCP connection. To overcome this limitation, HTTP/1.1 was facilitated with request pipelining but this could not provide complete request concurrency and still suffers from Head-of-Line (HoL) blocking[1]. Thus, clients using HTTP/1.0 or HTTP/1.1 require multiple connections to make many requests to a server in order to achieve concurrency and thereby reduce latency. This can be viewed as a negative impact on application performance. Such issues motivated researchers to develop HTTP/2 [17]. This protocol not only supports all the basic features of HTTP/1.1 but it is also very efficient in utilizing TCP transmission capability. As HTTP/2 is a

[1]An HTTP request resulting into voluminous response size which blocks servicing other small requests.

new protocol, the research community has not yet paid much attention on the security issues or vulnerabilities in HTTP/2. There are only few works available in literature which discussed some security flaws in HTTP/2 [107], [3], [108]. We believe that a proper understanding of possible threats that may affect normal operation of a widely popular protocol such as HTTP/2 is essential to develop appropriate detection and mitigation methods.

In this chapter, we describe new threat vectors of HTTP/2 which we found after a careful analysis of protocol's working. By exploiting the proposed threat vectors, a malicious client can launch Slow Rate DoS attacks which can thwart a HTTP/2 server from serving legitimate clients. Subsequently we propose a statistical abnormality measurement technique that uses chi-square test to detect deviations in the HTTP/2 traffic profile. This method can identify deviations in network traffic patterns due to presence of anomalous connections established to launch Slow Rate DoS attacks. Our contributions in this chapter are:

**1.** We propose novel Slow Rate HTTP/2 DoS attacks which are effective against majority of popular web servers.

**2.** We also propose a novel detection mechanism to detect these Slow Rate DoS attacks. We test it in a real network setup and show that it can detect anomalies in the HTTP/2 traffic with very high accuracy.

**3.** We perform experiments using both clear text and encrypted HTTP/2 requests and find that the attack is effective in both the cases.

**4.** We perform the vulnerability assessment of both HTTP/1.1 and HTTP/2 protocols and show that HTTP/2 has relatively more number of threat vectors that can be exploited to launch Slow Rate DoS attacks.

Rest of the chapter is organized as follows. We give an overview of HTTP/2 protocol and its working in Section 6.2. We explain the other threat vectors and known attacks against HTTP/2 in Section 6.3. In Section 6.4, we describe the working of proposed Slow Rate HTTP/2 DoS attacks. We describe proposed detection method in 6.5. In Section 6.6, we discuss behaviour of different web server implementations against proposed attacks and experimental evaluation of proposed detection method.

A comparison of vulnerability assessment of HTTP/1.1 and HTTP/2 protocols is presented in Section 6.7 followed by conclusion in Section 6.8.

## 6.2 HTTP/2 Protocol

The working of HTTP/2 protocol is described in RFC 7540 [17] which was standardized recently in May, 2015. The operation of HTTP/2 is significantly different from HTTP/1.1 though the semantics remains the same. HTTP/2 improves the ability of an application to efficiently utilize the transmission bandwidth which its predecessor is lacking. In the next few paragraphs, we discuss some features present in HTTP/2 protocol that makes it more efficient as compared to HTTP/1.1.

**1. Message Multiplexing:** HTTP/2 multiplexes different requests by associating each request with its own stream. Since streams are independent of each other, a blocked request or response does not prevent progress of other streams.

**2. Flow Control and Request Prioritization:** Flow control ensures that sender trasmits only that much data that can be used by receiver to prevent unnecessary retransmissions. Request prioritization ensures that more important requests are processed and served more quickly.

**3. Header Compression:** Since HTTP header fields can contain large amount of redundant data, frames containing them are compressed to reduce the request sizes. This allows many requests to be compressed into one packet.

### 6.2.1 HTTP/2 Components

HTTP/2 supports various types of components such as Connection Preface, frames, streams and flow control mechanism. In this subsection, we describe the working of these components during normal operation of HTTP/2.

**1. Connection Preface:** Upon connection establishment, a HTTP/2 client sends a Connection Preface to a HTTP/2 enabled web server to inform the server that client is going to use HTTP/2 for further communication. After Connection Preface, client can start sending HTTP/2 frames to server. Connection Preface contains a magic string

$PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n$. In case an invalid Connection Preface is received, server treats it as a connection error and closes the connection immediately.
**2. Frames:** The basic protocol unit in HTTP/2 is a frame. RFC 7540 defines different frame types such that each frame type serves different purpose. All frames begin with a fixed header of 9 bytes followed by variable length payload. Figure 6.1 [17] shows the common frame layout for different frame types. Different fields of the frame header are defined below:

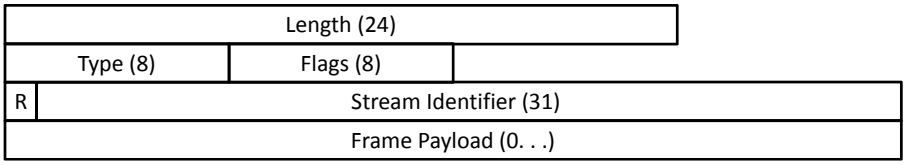



Figure 6.1: HTTP/2 frame layout

**Length:** This is a 24-bit field and represents the length of frame payload. This value does not include 9 bytes of the frame header.
**Type:** This field is of 8 bits and represents the type of frame. The frame type determines the format and semantics of the frame.
**Flags:** This is an 8-bit field reserved for boolean flags specific to the frame type.
**R:** This is a reserved 1-bit field and it must remain unset (0x0) when sending and must be ignored when receiving.
**Stream Identifier:** This is a 31-bit field and uniquely identifies a stream. The value 0x0 is reserved for frames associated with the connection as a whole as opposed to an individual stream.

As described earlier in this section, there are different types of HTTP/2 frames and each frame type is used for different purposes such as negotiating connection parameters, carrying header and data segments and terminating an established connection. These frame types and their purposes are as follows:
**SETTINGS Frame:** This frame is used by both client and server for the purpose of negotiating connection parameters like maximum number of concurrent streams per connection, initial window size and maximum frame size.

**WINDOW_UPDATE Frame:** WINDOW_UPDATE frame is used to indicate the number of bytes that the sender is willing to accept by its peer in addition to the sender's existing capacity to receive data in bytes.

**HEADERS and CONTINUATION Frames:** HEADERS frame is used to open a stream and carries a header block fragment. In case a header block is large enough not to fit in a single HEADERS frame, CONTINUATION frames are used to transmit remaining parts of header block.

**DATA Frame:** This frame is used to carry message body sent by the endpoints (client or server). For example, client uses DATA frame to carry message body of a POST request while server uses DATA frame to carry response to a client's GET request.

**PING Frame:** This frame is used to measure minimal round-trip time from the sender, as well as determining whether an idle connection is still functional.

**GOAWAY Frame:** This frame is used to either tear off a established connection between endpoints or indicate some serious error condition. RFC 7540 defines various error codes that GOAWAY frame carries to convey the reason behind connection or stream error. These error codes and the corresponding reasons are shown in Table 6.1.

**3. Streams:** A stream is an independent, bidirectional sequence of frames exchanged between the client and server within a HTTP/2 connection. One HTTP/2 connection can contain multiple concurrently open streams and these streams can be closed by either endpoint. The endpoint which initiated the stream is responsible for assigning stream identifier to it.

**4. Flow Control:** Flow control scheme in HTTP/2 ensures that streams on the same connection do not interfere with each other. Flow control is used for both individual streams and for the connection as a whole. HTTP/2 uses WINDOW_UPDATE frames to provide flow control. It is the receiver that chooses to set any window size that it desires for each individual stream and for the entire connection and the sender must respect the flow control limits imposed by the receiver. The initial size for the flow control window is 65,535 bytes for both new streams and the overall connection. Of all the frame types defined in RFC 7540, only DATA frames are subject to flow control.

Table 6.1: Various error codes and reason behind error

| Error Code | Reason behind Error |
|---|---|
| NO_ERROR | This code indicates graceful shutdown of a connection. |
| PROTOCOL_ERROR | This code indicates an unspecific protocol error and is usually transmitted when there is no more specific code available. |
| INTERNAL_ERROR | This code is transmitted in case an endpoint encounters an unexpected break in the normal working of protocol. |
| FLOW_CONTROL_ERROR | An endpoint transmits this code if peer violates the flow control negotiations made. |
| SETTINGS_TIMEOUT | In case an endpoint does not receive acknowledgement of a SETTINGS frame sent by it, it sends a GOAWAY frame with SETTINGS_TIMEOUT code. |
| FRAME_SIZE_ERROR | An endpoint transmits this code if it receives a frame with an invalid size. |

### 6.2.2 Normal Operation of HTTP/2

HTTP/2 runs on TCP port number 80 and 443 for *http* and *https* URIs respectively. Once a HTTP/2 connection is established, both client and server can start exchanging frames. Figure 6.2 shows exchange of various frames during normal operation of HTTP/2 protocol. The sequence is as follows:

**1. First Payload from Client to Server:** As soon as TCP connection is established, client creates two streams having stream identifiers 0 and 1 over the same connection such that on stream 0, it sends Connection Preface, SETTINGS frame and WINDOW_UPDATE frame. Using a parameter SETTINGS_INITIAL_WINDOW_SIZE of SETTINGS frame, client informs the server that how much data it can currently receive. Also, client sends a HEADERS frame containing HTTP GET request to the server on stream 1. Both of these streams are part of one HTTP/2 payload.

**2. Second Payload from Server to Client:** As soon as server receives the

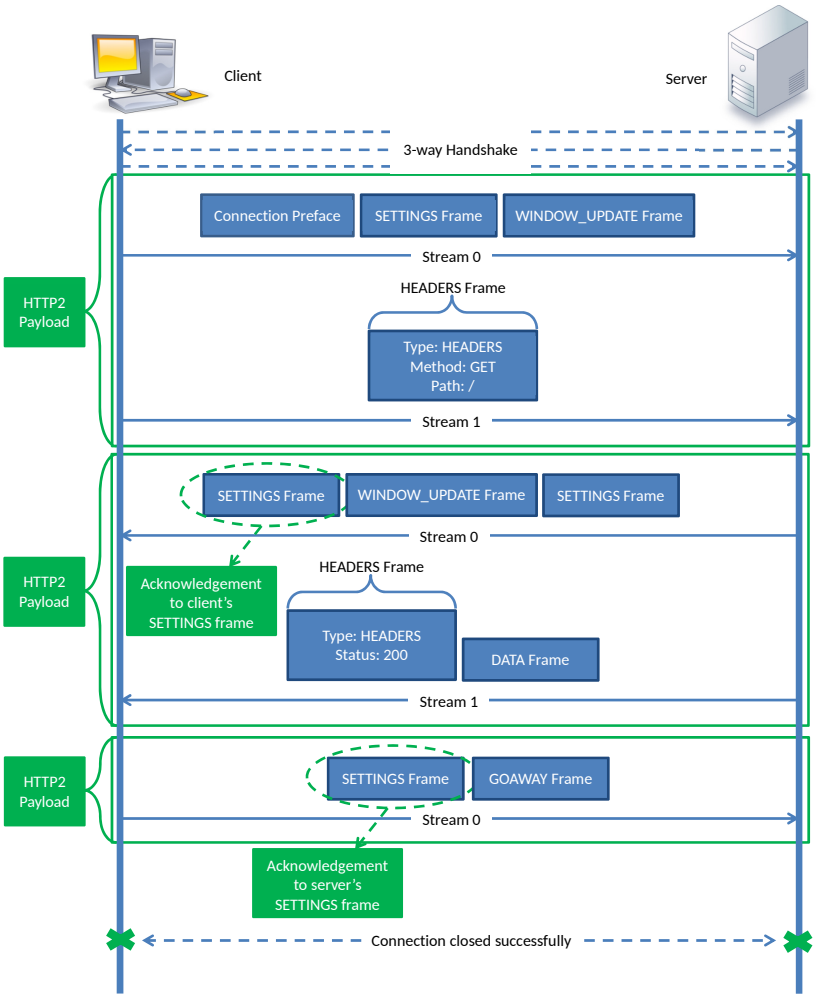



Figure 6.2: HTTP/2 working

HTTP/2 payload sent by the client in the first step, it acknowledges the receipt of SETTINGS frame by sending an empty SETTINGS frame on stream having identification number 0. Along with this frame, server also sends WINDOW_UPDATE frame and a non-empty SETTINGS frame. Similar to the client's SETTINGS frame sent in the previous step, the server's SETTINGS frame also contains SETTINGS_INITIAL_WINDOW_SIZE parameter using which server informs the client that how much data it can currently receive. Moreover, as a response to the GET request, server also sends HEADERS and one or more DATA frames on stream 1 to the client. The HEADERS frame contains the header block the HTTP response while DATA frame contains the message body of the response.

**3. Third Payload from Client to Server:** Once client receives the HTTP/2 payload sent by the server in the second step, it acknowledges the receipt of SETTINGS frame by sending an empty SETTINGS frame on stream having identification number 0. The client also sends a GOAWAY frame on the same stream to close the connection successfully.

## 6.3 Prior Work

Since HTTP/2 is standardized recently in 2015, very few works are available in the literature which discuss security aspects of the protocol. In this section, we describe the HTTP/2 vulnerabilities and attacks found in the literature.

### 6.3.1 Vulnerabilities in HTTP/2 Protocol

Several works have been published on DoS attacks against HTTP/1.1 protocol. However, very few works [2], [3], [107] are available in the literature which discuss DoS/DDoS attacks against HTTP/2. In [2] and [3], authors described five attack scenarios which involve sending of millions of PING and WINDOW_UPDATE frame packets to a HTTP/2 enabled server and then monitoring different resource parameters such as CPU usage, memory consumption, network throughput and packet loss. These attacks require sending a large number of HTTP/2 frames to create DoS scenario. Both of these works mainly involve analyzing effect of DDoS attacks on a HTTP/2 enabled web server. We present a comparison of our proposed attacks and attacks proposed in [2], [3] later in Section 6.6.2 of this chapter.

Imperva [107], a security consultancy firm, proposed four attacks against HTTP/2 protocol- *Slow Read*, *HPACK (Compression)*, *Dependency DoS* and *Stream abuse*. They tested these attacks against five different web servers and showed that all web servers are vulnerable to at least one of the attacks. The working of these attacks are described below:

**1. Slow Read:** In this attack, a malicious client sets very small SETTINGS_INITIAL_WINDOW_SIZE and sends GET requests from several concurrent streams multi-

plexed over a single TCP connection. SETTINGS_INITIAL_WINDOW_SIZE field indicates the sender's currently available capacity for receiving the data in bytes from its peer. Servers like Apache 2.4.17 and Apache 2.4.18 are vulnerable to the attack because these versions dedicate one thread per stream which results into consumption of all worker threads once enough number of streams are created. However, with the release of latest security patch [109], Apache 2.4.20 and later server versions are not vulnerable to this attack as these versions dedicate threads on per connection basis instead of per stream basis.

**2. HPACK (Compression):** HPACK compression algorithm [110] is used by HTTP/2 protocol to compress header size. For compression purpose, sender informs other endpoint about the maximum size of header compression table which can be used to decode header blocks. The encoder, then can select any table size less than or equal to this value. However, RFC 7540 does not put a restriction on the size of each individual header. Thus, size of each header is restricted to the size of header compression table. A malicious client can abuse the working of this algorithm by creating a stream with large header equal to the size of table. After this, multiple streams are created over the same connection and each of them are referenced to the single large header multiple number of times. While processing these streams, server decompresses each request and allocates memory for each large header. As a result, the whole memory gets eventually consumed due to which server becomes unable to serve any further HTTP request.

**3. Dependency DoS:** HTTP/2 facilitates stream dependency such that a stream can depend on another stream. Thus, if a stream $S_2$ depends on a stream $S_1$, stream $S_2$ is not processed unless stream $S_1$ is closed or stream $S_1$ cannot progress due to, for example, flow control or data is not available from backend content server. These dependency links form a tree which is known as dependency tree. The circular dependency is not allowed as it leads to starvation due to which no stream is processed by a server. To launch this attack, a malicious client tries to create scenarios like circular dependency and/or rapid changes in the stream dependencies. These situations lead to high CPU utilization and unreasonable memory utilization and thus affects server's

ability to serve HTTP/2 requests.

**4. Stream Abuse:** This attack involves reusing stream identifiers over the same connection which RFC 7540 strictly prohibits. However, few server implementations like Microsoft's Internet Information Services (IIS) older versions do not follow this mechanism and as a result, they are vulnerable to this attack. After the disclosure of these vulnerabilities, vendors of different HTTP/2 implementations released new versions of server software which are no longer vulnerable to the above described attacks [107].

Yahoo penetration testing team, in their work [108], presented some flaws found in various HTTP/2 implementations like Firefox, Apache Traffic Server (ATS) and Node-http2. These flaws basically caused issues like arbitrary code execution, unreasonable memory allocation due to integer underflow and buffer out of bound reads.

## 6.4 Slow Rate HTTP/2 DoS Attacks

In this section, we present five new Slow Rate HTTP/2 DoS attacks using **i)** *Zero SETTINGS_INITIAL_WINDOW_SIZE*, **ii)** *Complete POST Header*, **iii)** *Connection Preface*, **iv)** *Incomplete GET/POST Header* and **v)** *SETTINGS frame.* These attacks target the number of free connection slots available on a server by establishing enough number of connections and sending specially crafted HTTP/2 requests from each of them. Since these special requests hold back established connections for a long duration, server does not entertain requests sent by genuine HTTP clients for that amount of time which leads to DoS. In next few subsections, we discuss working of proposed Slow Rate HTTP/2 DoS attacks.

### 6.4.1 Attack-1: Using Zero SETTINGS_INITIAL_WINDOW_SIZE

As described earlier in Section 6.2.1, HTTP/2 uses flow control scheme to regulate the amount of data transmitted in either direction. SET-

TINGS_INITIAL_WINDOW_SIZE field in SETTINGS frame is set with appropriate value to indicate the initial amount of memory the sender of this frame has (in bytes) and hence can not receive data more than this capacity. In the subsequent phases, this value is updated by sending WINDOW_UPDATE frames.

In this attack, a malicious client sends a legitimate Connection Preface and a HTTP GET request but deliberately manipulates the SETTINGS_INITIAL_WINDOW_SIZE value of the SETTINGS frame being sent by setting it to zero as shown in Figure 6.3. This is interpreted by the server as client currently does not have any space to receive

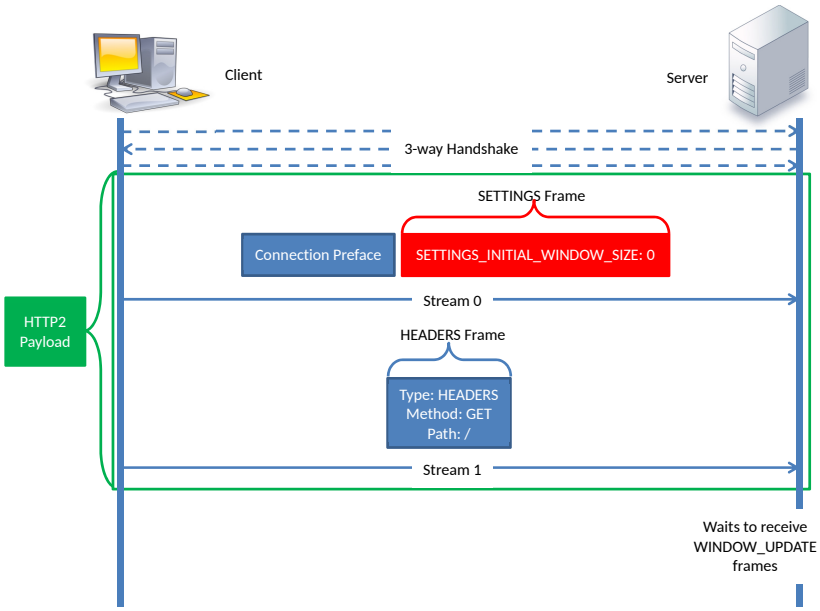



Figure 6.3: Attack-1

data and hence, it should wait to receive a WINDOW_UPDATE frame from the client. However, a malicious client never sends any WINDOW_UPDATE frame to server as its intention is to only hold this connection for long time. Normally, web servers wait for relatively large amount of time anticipating WINDOW_UPDATE frame which never arrives. On the contrary, the malicious client creates many such connections to hold the slots in the connection pool for a long time. This creates a DoS scenario once the server's connection pool is exhausted. Since this attack requires sending only one GET request per connection, malicious client needs to create just one stream per connection

as compared to the Slow Read attack [107] which requires multiple streams over each connection.

### 6.4.2 Attack-2: Using Complete POST Header

As described earlier in Section 6.2.1, HTTP/2 frame structure has an 8-bit *Flags* field and is reserved for boolean flags specific to frame type. For example, a DATA frame defines a *END_STREAM* flag while a HEADERS frame, along with this flag, additionally defines a *END_HEADERS* flag. A DATA frame having END_STREAM flag set indicates that this is the last frame that the client or server is sending for the identified stream. A HEADERS frame having END_STREAM flag set indicates that this is the last header block that the client or server is sending for the identified stream. However, if the header block is large enough not to fit in a HEADERS frame, it is followed by CONTINUATION frames. Thus, CONTINUATION frames can be transmitted even after transmitting a HEADERS frame having END_STREAM flag set. On the other hand, a HEADERS frame having END_HEADERS flag set indicates that this frame contains an entire header block and is not followed by any CONTINUATION frames.

In this attack, a malicious client sends complete header (but not message body) of POST request contained within a HEADERS frame and deliberately manipulates the END_STREAM and END_HEADERS flags of this HEADERS frame by setting them to 0 and 1 respectively as shown in Figure 6.4. As soon as server receives this HEADERS frame, it interprets that it has received complete header block of POST request (due to END_HEADERS flag set) but as END_STREAM flag is reset, the HEADERS frame is going to be followed by one or more DATA frames. However, malicious client never sends the DATA frames due to which, depending on the server's implementation, it waits for a particular amount of time to receive DATA frames before closing the connection.

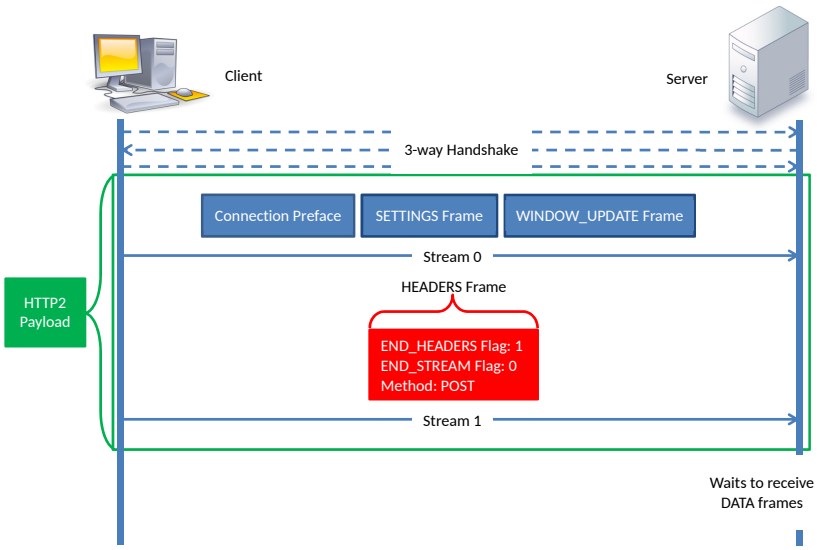



Figure 6.4: Attack-2

### 6.4.3 Attack-3: Using Connection Preface

As discussed earlier in Section 6.2.1, Connection Preface is the first frame sent by a client to a HTTP/2 web server after connection establishment. This message is sent to the server to inform that client is going to use HTTP/2 for further communication over this connection. Connection Preface is also used to inform a large proportion of HTTP/1.1 and HTTP/1.0 servers and intermediaries to not attempt to process further frames received over this connection. After Connection Preface, client can start sending other HTTP/2 frames to the server.

In this attack, a malicious client first sends Connection Preface to the web server as shown in Figure 6.5. On receiving this, the web server honours this message and starts waiting to receive SETTINGS and WINDOW_UPDATE frames on stream 0 and a HEADERS frame containing either GET or POST request on stream 1. However, malicious client never sends these HTTP/2 frames. The malicious client can also repeatedly send the Connection Preface at regular intervals to make server wait for a long time by resetting the timer responsible for closing the connection after timeout. This allows malicious client to get enough waiting time at server to create DoS scenario.

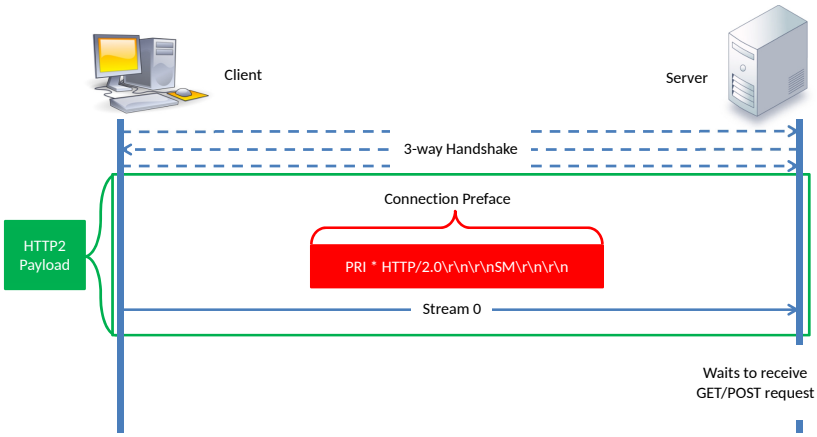



Figure 6.5: Attack-3

### 6.4.4 Attack-4: Using Incomplete GET/POST Header

As described in Section 6.4.2, a HEADERS frame having END_HEADERS flag set indicates that this frame contains the complete header block of a GET or POST request and this frame must not be followed by any CONTINUATION frame. In this attack, a malicious client sends a GET request within a HEADERS frame and deliberately manipulates the END_HEADERS and END_STREAM flags of this HEADERS frame by setting them to 0 and 1 respectively as shown in Figure 6.6. When server re-

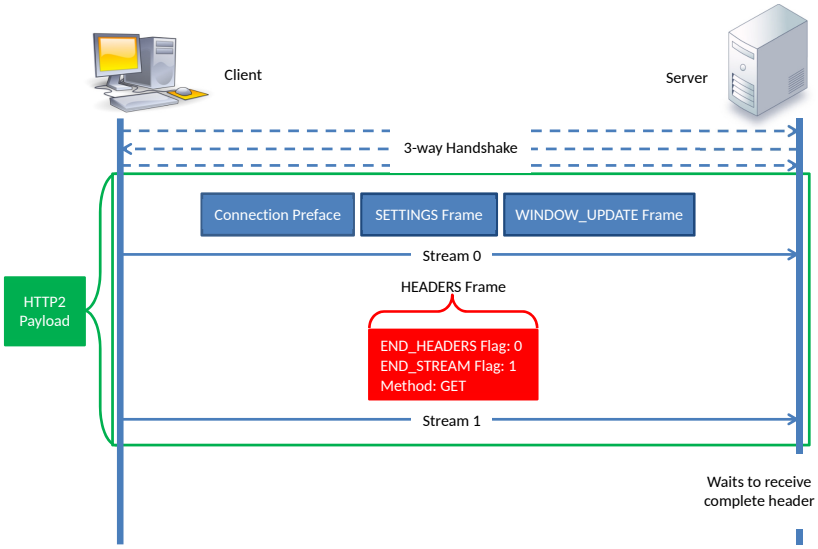



Figure 6.6: Attack-4

ceives this HEADERS frame, server interprets that it has received incomplete header and it should wait to receive CONTINUATION frames. However, malicious client never sends CONTINUATION frames due to which server waits for a particular time period before closing the connection. This attack can also be launched using incomplete POST request as well. In this case, malicious client sends a POST request within a HEADERS frame and deliberately manipulates the END_HEADERS and END_STREAM flags of this HEADERS frame by setting both of them to 0. When server receives this HEADERS frame, server interprets that it has received incomplete POST request and it should wait to receive CONTINUATION frames. However, malicious client never sends CONTINUATION frames due to which server waits for a particular time period before closing the connection. It is to be noted that malicious client has to set the END_STREAM flag also to 0 while launching this attack using incomplete POST request as server terminates a connection if it receives a HEADERS frame containing POST request and having END_STREAM flag set to 1. This is because server assumes that client would be having some DATA frames also to be sent and if END_STREAM flag is 1, client can not send any other frame except CONTINUATION frames (as discussed in Section 6.4.2).

### 6.4.5 Attack-5: Using SETTINGS frame

According to RFC 7540 [17], a SETTINGS frame can be sent by both client or server at the start of a connection and may be sent at any other time by either of them over the lifetime of the connection. Also, a SETTINGS frame sent by one endpoint must be acknowledged by another endpoint. For this purpose, the SETTINGS frame defines an ACK flag. This flag, when set, indicates that this SETTINGS frame is sent to acknowledge receipt and application of the peers SETTINGS frame. When this bit is set, the payload of the SETTINGS frame remains empty. RFC 7540 also suggests that if the sender of a SETTINGS frame does not receive an acknowledgement within a reasonable amount of time, it may close the connection by showing the error code SETTINGS_TIMEOUT. Different server implementations wait for different amount of time before receiving an acknowledgement for the sent SETTINGS frame. In this

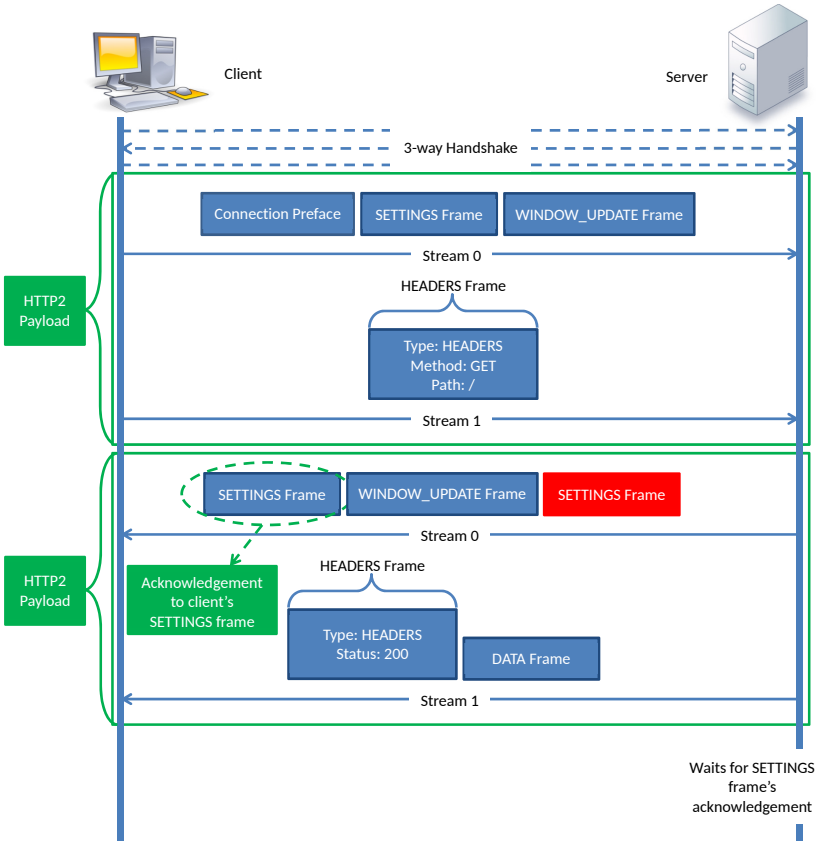



Figure 6.7: Attack-5

attack, a malicious client sends a HTTP/2 payload containing Connection Preface, SETTINGS, WINDOW_UPDATE and a HEADERS frame having legitimate complete GET or POST request as shown in Figure 6.7. On receiving this payload, server sends back a payload containing WINDOW_UPDATE, HEADERS, DATA and two SETTINGS frame. First SETTINGS frame (shown in green colour in Figure 6.7) is sent to acknowledge SETTINGS frames sent by the client in the previous HTTP/2 payload while second SETTINGS frame (shown in red colour in Figure 6.7) includes parameters that server wants to negotiate with the client. The second SETTINGS frame must be acknowledged by malicious client by sending a SETTINGS frame with empty payload and having ACK flag set. However, malicious client does not acknowledge the second SETTINGS frame sent by server. As a result, vulnerable web servers wait for a particular time to receive the acknowledgement. Since this attack can be

launched by sending either GET or POST request, there are two flavours of this attack also similar to Attack-4.

## 6.5 Detecting Slow Rate HTTP/2 DoS Attacks

Slow Rate HTTP/2 DoS attacks proposed in this chapter are effective against majority of web servers (see Section 6.6.1) as these attacks exploit the flaws in specification and are not specific to any implementation errors. For instance, a HTTP/2 server's tendency to wait for receiving a complete message body is not an implementation bug but its intentional behaviour because a large resource can not be sent in a single DATA frame and thus, it is transmitted in several DATA frames. As a result, patching HTTP/2 web servers to mitigate these Slow Rate HTTP/2 DoS attacks is not a feasible solution.

Since the proposed attacks can not be patched, we propose a method to detect these attacks by inspecting HTTP/2 traffic in network. In this section, we propose an anomaly detection scheme to detect these Slow Rate DoS attacks. Our detection scheme involves comparing HTTP/2 traffic profiles generated during training phase and testing phase using a set of features. In training phase, a normal HTTP/2 profile is generated by collecting legitimate web traffic coming to and going from a server over a period of $x$ observation intervals. In testing phase, detection system compares the current traffic profile with the profile generated during training phase. For this comparison, the proposed anomaly detection scheme uses chi-square test [111] (See Appendix C). Chi-square test has been used in the literature [112, 113, 114, 115, 116] to detect anomalous events such as password guessing and attempting to gain an unauthorized remote access on a computer.

In the next subsection, we describe the candidate features used to generate the HTTP/2 traffic profiles during training and testing phase.

### 6.5.1 Feature Selection

As discussed earlier in this section, we generate the HTTP/2 traffic profiles during training and testing phase using a set of features. In particular, these features are number of occurrences of few specific HTTP/2 frames. For example, occurrence of a SETTINGS frame whose SETTINGS_INITIAL_WINDOW_SIZE field is 0. Similarly, few other features are also defined to detect different types of attacks and are shown in Table 6.2. Feature-1 corresponds to number of SETTINGS frame having

Table 6.2: Candidate features

| **Features** | **Description** | **Type** |
|---|---|---|
| Feature-1 | SETTINGS frame having SETTINGS_INITIAL_WINDOW_SIZE set to 0 | Numeric |
| Feature-2 | HEADERS frame having END_STREAM_FLAG Reset | Numeric |
| Feature-3 | Flows having Connection Preface only | Numeric |
| Feature-4 | HEADERS frame having END_HEADERS_FLAG Reset | Numeric |
| Feature-5 | Server's SETTINGS frames which are not acknowledged | Numeric |

SETTINGS_INITIAL_WINDOW_SIZE set to 0. Similarly, Feature-2 and Feature-4 correspond to number of HEADERS frame with either END_STREAM_FLAG or END_HEADERS_FLAG reset respectively. Feature-3 corresponds to number of flows[2] having Connection Preface only while Feature-5 corresponds to number of server's SETTINGS frames which are not acknowledged.

### 6.5.2 Adapting Chi-square Test for Slow Rate HTTP/2 DoS Attack Detection

Chi-square test can easily be adapted to determine if there is a significant deviation between two variables. In our case, these two variables correspond to the HTTP/2 traffic profiles generated during training and testing phase. To adapt chi-square test

[2]Each HTTP/2 flow or connection is uniquely identified by the combination of source IP address and source port number.

for detecting proposed attacks, we also require proposing *null* and *alternate* hypotheses. A null hypothesis states that there is no significant difference between two profiles while an alternate hypothesis states that there is significant difference between the two profiles. In our case, for a particular time interval $\Delta T$ of testing phase, null hypothesis states that $\Delta T$ does not contain anomalous flows and thus, there is no significant difference between the training profile and profile generated during $\Delta T$ period of testing phase. On the other hand, alternate hypothesis states that $\Delta T$ contains anomalous flows and thus, the profile generated during $\Delta T$ period is significantly deviated from training profile. Chi-square test statistic is defined as in Equation 6.1. In this equation, $n$ is the number of features, $O_i$ corresponds to Feature-$i$ received in a particular time interval of testing phase and $E_i$ corresponds to mean of Feature-$i$ received in different time intervals of training phase.

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{6.1}$$

### 6.5.3 Normal HTTP/2 Profile Creation

In order to make our detection system learn normal HTTP/2 behaviour, we collect legitimate HTTP/2 traffic coming to and going from the web server over a period of $x$ time intervals each of duration $\Delta T$. The HTTP/2 profile, $E$, is then created by taking mean of each feature received in different time intervals of training phase. $E$ is a $n$ dimensional vector with each component $E_i$ of the vector representing Feature-$i$. $E_i$ is estimated using Equation 6.2

$$E_i = \frac{\sum_{t=1}^{x} e_{it}}{x} \tag{6.2}$$

where $e_{it}$ corresponds to Feature-$i$ received in $t^{th}$ time interval of training phase and $x$ is the total number of time intervals in the training phase.

### 6.5.4 Comparing Profile Generated during Training and Testing Phase

Once the detection system is trained with the normal HTTP/2 profile, we use it to detect presence of attacks from $(x+1)^{th}$ interval. The steps involved in the working of anomaly detection system are shown in Figure 6.8. We generate a profile, $O$, after

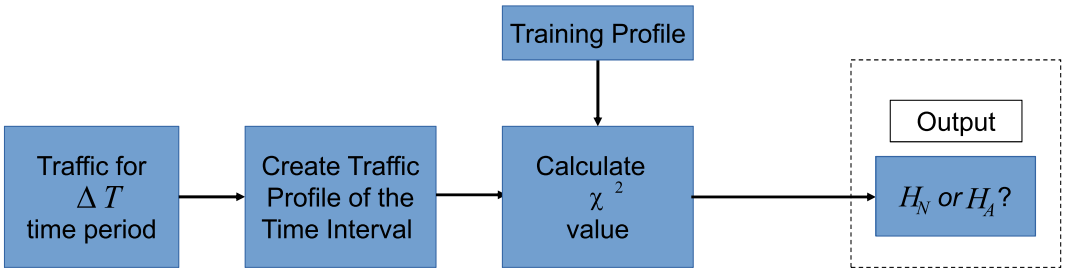



Figure 6.8: Anomaly detection system working

every $\Delta T$ duration. Similar to $E$, $O$ is an $n$ dimensional vector with each component $O_i$ of the vector $O$ representing Feature-$i$ received in a particular time interval of testing phase. $O$ is then compared with $E$ using chi-square test statistic. If obtained $\chi^2$ value is less than the predefined threshold, detector accepts $H_N$. However, if $\chi^2$ value is greater the predefined threshold, detector accepts $H_A$. For each time interval, detector accepts either $H_N$ or $H_A$.

## 6.6 Experimental Evaluation

In this section, we describe the experiments performed to evaluate the proposed attacks and anomaly detection system. We first tested the proposed attacks against four popular web servers and examined the impact of these attacks on different web servers. Subsequently, we also describe the experiments performed to evaluate detection performance of proposed scheme to detect Slow Rate HTTP/2 DoS attacks.

### 6.6.1 Behaviour of Web Servers against Slow Rate HTTP/2 DoS Attacks:

In this subsection, we describe the effect of proposed Slow Rate DoS attacks on different web servers.

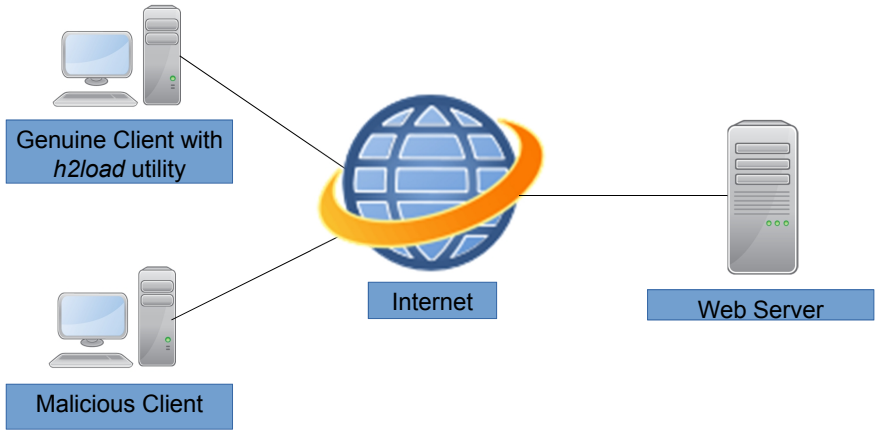



Figure 6.9: Testbed architecture

**Testbed Setup:** To test the proposed attacks, we created a testbed similar to the one shown in Figure 6.9. We installed four popular [117] HTTP/2 supporting web servers namely Apache 2.4.23, Nginx 1.10.1, H2O 2.0.4 and Nghttp2 1.14.0 on a computer with 4 GB of physical memory and a quad core processor and having internet connectivity. This computer was running Kali 2.0 operating system. We hosted a sample website in each server's home directory. This website contains a homepage having fifty images. All four servers were tested in their default configurations and with the latest versions available as on October 24th, 2016. We also designated two other computers in the network as malicious and genuine clients. Each of these clients were having 4 GB of physical memory and dual core processor and running Ubuntu 16.04 LTS operating system. Malicious client was used to launch the proposed attacks while the genuine client was used to check the server's availability while attack was going on. Both malicious client and genuine client were also connected to internet. Due to unavailability of any full-fledged tool that can provide flexibility to modify some parameters of HTTP/2 frames while testing the protocol, we wrote python

scripts which can send custom HTTP/2 requests to the server. We used h2load [118] benchmarking tool to generate and send legitimate requests from genuine client to check the server's availability during attack. In particular, h2load sends a HTTP/2 GET request to retrieve an object from the server. Depending on whether server responds back or not, we conclude that server is available or unavailable respectively.

**Behaviour of Web Servers against Attack-1:** As described in Section 6.4.1, Attack -1 requires sending a HTTP/2 payload having a SETTINGS frame with SETTINGS_INITIAL_WINDOW_SIZE field set to zero and a complete GET request. If such payload is sent from a connection, Apache waits for 300 seconds before closing the connection. After this waiting time, server closes the connection by sending a GOAWAY frame with NO_ERROR code back to malicious client. Both Nginx

Table 6.3: Servers' behaviour against Attack-1

| **Server** | **Waiting Time** | **Connection Closing Behaviour** |
|---|---|---|
| Apache | 300 | GOAWAY frame with NO_ERROR code |
| Nginx | 60 | RST_STREAM frame with PROTOCOL_ERROR code |
| H2O | Indefinite | Never closes the connection |
| Nghttp2 | 60 | RST_STREAM frame with INTERNAL_ERROR code |

and Nghttp2 wait for 60 seconds before closing the connection. After this waiting time, Nginx and Nghttp2 close the connection by sending a RST_STREAM frame with PROTOCOL_ERROR and INTERNAL_ERROR code respectively. H2O is also vulnerable to this attack as it waits for indefinite time to receive WINDOW_UPDATE frame. Table 6.3 summarizes the behaviour of different web servers if the payload corresponding to Attack-1 is sent.

**Behaviour of Web Servers against Attack-2:** In Section 6.4.2, we described that Attack-2 requires sending a complete POST request contained within

a HEADERS frame whose END_HEADERS and END_STREAM flags are set and reset respectively. If such request is sent, servers wait for a particular amount of time to receive DATA frames. However, if malicious client does not send any DATA

Table 6.4: Servers' behaviour against Attack-2

| **Server** | **Min/Max Waiting Time** | **Connection Closing Behaviour** |
|---|---|---|
| Apache | 600/Indefinite | 1. GOAWAY frame with NO_ERROR code after minimum waiting time,<br>2. Never closes connection after maximum waiting time |
| Nginx | 30/Indefinite | 1. GOAWAY frame with PROTOCOL_ERROR code after minimum waiting time,<br>2. Never closes connection after maximum waiting time |
| H2O | 10 Indefinite | 1. GOAWAY frame with NO_ERROR code after minimum waiting time,<br>2. Never closes connection after maximum waiting time |
| Nghttp2 | 10/975 | GOAWAY frame with SETTINGS_TIMEOUT code independent of waiting time |

frame after sending POST request to server, Apache, Nginx and H2O close the connection after 600, 30 and 10 seconds respectively. After this timeout, Apache and H2O send back GOAWAY frame with NO_ERROR code while Nginx sends back GOAWAY frame with PROTOCOL_ERROR code to close the connection. However, if malicious client keeps sending DATA frames having END_STREAM flag reset at regular intervals, all these three servers wait for an indefinite amount of time by resetting their expiry timers. Nghttp2 closes the connection after 10 seconds by sending GOAWAY frame with SETTINGS_TIMEOUT code if no DATA frame is received. In case malicious client tries to extend this timeout duration by sending a DATA frame, then also Nghttp2 closes the connection after 10 seconds by sending GOAWAY frame with SETTINGS_TIMEOUT code. However, if malicious client

acknowledges the SETTINGS frames which was sent by the server after connection establishment, server waits for malicious client to close the connection. If malicious client does not close the connection, Nghttp2 waits for a time period of 975 seconds. After this timeout, Nghttp2 closes the connection without sending any frame back to the client. Table 6.4 summarizes the behaviour of different web servers if the payload corresponding to Attack-2 is sent.

**Behaviour of Web Servers against Attack-3:** As described in Section 6.4.3, to launch Attack-3, malicious client sends only Connection Preface without any GET/POST request. If such a payload is received by the Apache web server, it waits for 300 seconds in the hope of receiving GET/POST request from the malicious client before closing the connection. After this timeout, server closes the connection by sending GOAWAY frame with NO_ERROR code. In case malicious client tries to reset this time counter by sending the Connection Preface again, server immediately closes the connection by sending GOAWAY frame with PROTOCOL_ERROR code. Nginx waits for 30 seconds in the hope of receiving GET/POST request. After this timer expires, server closes the connection by sending GOAWAY frame with PROTOCOL_ERROR code. However, if malicious client keeps sending Connection Preface at regular intervals, Nginx waits for an indefinite amount of time. H2O is not vulnerable to this attack as it closes the connection just after 10 seconds by sending GOAWAY frame with NO_ERROR code. In case malicious client tries to extend this timeout duration by sending Connection Preface again, H2O immediately closes the connection by sending GOAWAY frame with FRAME_SIZE_ERROR code. Nghttp2 also closes the connection after 10 seconds by sending GOAWAY frame with SETTINGS_TIMEOUT code. However, similar to attack discussed in Section 6.4.2, if malicious client acknowledges the SETTINGS frames sent by the server after connection establishment, it waits for 975 seconds in the hope that malicious client will close the connection. After this timeout, Nghttp2 closes the connection without sending any frame back to client. Table 6.5 summarizes the behaviour of different web servers if the payload corresponding to Attack-3 is sent.

Table 6.5: Servers' behaviour against Attack-3

| **Server** | **Min/Max Waiting Time** | **Connection Closing Behaviour** |
|---|---|---|
| Apache | 300/300 | 1. GOAWAY frame with NO_ERROR code after minimum waiting time,<br>2. GOAWAY frame with PROTOCOL_ERROR code after maximum waiting time |
| Nginx | 30/Indefinite | 1. GOAWAY frame with PROTOCOL_ERROR code after minimum waiting time,<br>2. Never closes connection after maximum waiting time |
| H2O | 10/10 | 1. GOAWAY frame with NO_ERROR code after minimum waiting time,<br>2. GOAWAY frame with FRAME_SIZE_ERROR code after maximum waiting time |
| Nghttp2 | 10/975 | 1. GOAWAY frame with SETTINGS_TIMEOUT code after minimum waiting time,<br>2. Closes connection without sending any frame after maximum waiting time |

**Behaviour of Web Servers against Attack-4:** In Section 6.4.4, we described that Attack-4 requires sending of an incomplete GET request contained within a HEADERS frame whose END_HEADERS and END_STREAM flags are reset and set respectively. If malicious client sends such request from a connection, Apache and Nginx wait for 300 and 90 seconds respectively before closing the connection. However, if malicious client keeps sending CONTINUATION frames having END_HEADERS flag reset at regular intervals, Apache waits for an indefinite amount of time by resetting its expiry timer again and again. However, Nginx do not reset the expiry timer and after the waiting time expires, Nginx closes the connection by

Table 6.6: Servers' behaviour against Attack-4

| **Server** | **Min/Max Waiting Time** | **Connection Closing Behaviour** |
|---|---|---|
| Apache | 300/Indefinite | 1. GOAWAY frame with NO_ERROR code after minimum waiting time,<br>2. Never closes connection after maximum waiting time |
| Nginx | 90/90 | GOAWAY frame with PROTOCOL _ERROR code independent of waiting time |
| H2O | 10/Indefinite | 1. GOAWAY frame with NO_ERROR code after minimum waiting time,<br>2. Never closes connection after maximum waiting time |
| Nghttp2 | 10/60 | 1. GOAWAY frame with HEADER_TIMEOUT code after minimum waiting time,<br>2. RST_STREAM frame with INTERNAL_ERROR code after maximum waiting time |

sending GOAWAY frame with PROTOCOL_ERROR code. Both H2O and Nghttp2 just wait for 10 seconds after receiving such HEADERS frame. After timeout, H2O and Nghttp2 close the connection by sending GOAWAY frame with NO_ERROR and HEADER_TIMEOUT code respectively. However, if malicious client keeps sending CONTINUATION frames having END_HEADERS flag reset at regular intervals, H2O waits for an indefinite amount of time while Nghttp2 waits for 60 seconds. After this timeout, Nghttp2 closes the connection by sending RST_STREAM frame with INTERNAL_ERROR code. Table 6.6 summarizes the behaviour of different web servers if the payload corresponding to Attack-4 is sent.

**Behaviour of Web Servers against Attack-5:** As described in Section 6.4.5, Attack-5 requires sending of a HTTP/2 payload containing Connection Preface, SETTINGS, WINDOW_UPDATE and a HEADERS frame having legitimate

complete GET or POST request. On receiving such payload, Apache and H2O send the response back for malicious client's request and close the connection after 5 and 10 seconds respectively by sending back GOAWAY frame with NO_ERROR code without waiting for their SETTINGS frames to be acknowledged. As a result, both these servers are not vulnerable to this attack. However, Nginx waits for 180

Table 6.7: Servers' behaviour against Attack-5

| **Server** | **Min/Max Waiting Time** | **Connection Closing Behaviour** |
|---|---|---|
| Apache | 5/5 | GOAWAY frame with NO_ERROR code independent of waiting time |
| Nginx | 180/180 | GOAWAY frame with NO_ERROR code independent of waiting time |
| H2O | 10/10 | GOAWAY frame with NO_ERROR code independent of waiting time |
| Nghttp2 | 10/975 | 1. GOAWAY frame with SETTINGS_TIMEOUT code after minimum waiting time,<br>2. Closes conenction without sending any frame after maximum waiting time |

seconds to receive the acknowledgement before closing the connection. After this timeout, it closes the connection by sending GOAWAY frame with NO_ERROR code. Nghttp2 waits for 10 seconds after serving a request sent by the malicious client. After this timeout, server closes the connection by sending GOAWAY frame with SETTINGS_TIMEOUT code. However, if the SETTINGS frame sent by server after connection establishment are acknowledged, server waits for 975 seconds in the hope that malicious client will close the connection. After this timeout, Nghttp2 closes the connection without sending any frame back to client. Table 6.7 summarizes the behaviour of different web servers if the payload corresponding to Attack-5 is sent.

Table 6.8 shows number of connections different web servers can handle at a time.

Table 6.8: Number of simultaneous connections servers can handle

| **Server** | **Number of Connections** |
|---|---|
| Apache | 150 |
| Nginx | 2060 |
| H2O | 1024 |
| Nghttp2 | 1142 |

Malicious client needs to establish these number of connections (within their respective timeout periods) in order to create a DoS scenario using either of the proposed attacks.

### 6.6.2 Comparing HTTP/2 DoS Attacks

In [2] and [3], authors described five attack scenarios which involve sending of millions of PING and WINDOW_UPDATE frame packets to a HTTP/2 enabled server and then monitoring four parameters - CPU usage, memory consumption, network throughput and packet loss. These attacks require a large amount of bandwidth to create DoS scenario, thus, can be easily detected. On the contrary, our proposed Slow Rate HTTP/2 DoS attacks require minimal bandwidth and thus, they are more stealthier. Table 6.9 shows the comparison between the attacks proposed by us and the DDoS attacks presented in [2] and [3] in terms of minimum and maximum number of packets required to cause DoS. The attacks proposed in this chapter require minimum 150 (against Apache) and maximum 2060 (against Nginx) connections and hence packets to create DoS scenario at the server. On the other hand, DoS attacks discussed in [2] and [3] require millions of packets. Another limitation with the attacks proposed in [2] and [3] is that these attacks could only achieve Reduction of Quality (RoQ) scenario instead of DoS as they were not able to exhaust the computational resource at web server completely.

Table 6.9: Comparison of proposed attacks with the attacks discussed in [2] and [3]

| | **Minimum Number of Packets Required** | **Maximum Number of Packets Required** |
|---|---|---|
| Proposed Attacks | 150 | 2,060 |
| Attacks in [2] | 2,000,000 | 2,000,000 |
| Attacks in [3] | 1,000,000 | 2,000,000 |

### 6.6.3 Slow Rate HTTP/2 DoS Attacks over TLS

Since a significant portion of HTTP/2 traffic on the Internet is sent using encryption, we test the effectiveness of proposed attacks over TLS also. For this purpose, we repeated the above experiments by configuring web servers to use TLSv1.2 [119]. We used OpenSSL [120] to generate the required digital certificate and private key. All the four servers were configured to use the generated certificate and private key. For the purpose of packet crafting and automated TLS handshake, we wrote a python script that uses scapy-ssl_tls python library [121] configured on malicious client. The script first performed the TLS handshake for negotiating parameters such as session key with server and then encrypted the specially crafted HTTP/2 request using the session key exchanged for the session. We hardcoded the required HTTP/2 payload in *hex* format in the script. The resulting encrypted request was then sent to server. The python script was automatically executed multiple times using a shell script so that malicious client could establish multiple concurrent connections with the server and send an encrypted HTTP/2 request from each of those connections. Once enough number of connections were established, genuine client was used to check the availability of server.

From this experiment, we observed that malicious client required same number of connections as shown in Table 6.8 to create DoS scenario using either clear text or encrypted HTTP/2 requests. We also observed that the time durations for which requests sent during execution of different attacks (as shown in Tables 6.3-6.7) hold the connection queue space was same for clear text and encrypted request. Thus, the

proposed attacks are found to cause similar effects on each server irrespective of clear text or encrypted HTTP/2 requests.

### 6.6.4 Experiments with Anomaly Detection Scheme

Apart from new Slow Rate HTTP/2 DoS attacks, our contribution in this chapter is also an anomaly detection system which is described in Section 6.5. We performed experiments using this detection method in a lab setup. These experiments are described in this subsection.

**Testbed Setup:** We created a testbed similar to the one shown in Figure 6.9. We used Apache web server for our experiments as it is the most popular web server worldwide [117]. We hosted a sample website on this server. This sample website was having 25 web pages containing very brief tutorials of an online course. One out of these web pages was also designed to upload an image of size 617 Kilobytes on the web server and the instructions such as size and format of image to be uploaded was also given on this webpage. We designated one computer as genuine client that simulates the behaviour of web users using a python program and sends legitimate HTTP/2 traffic towards the web server using h2load utility. This utility provides a Console User Interface (CUI) based browser with HTTP/2 support. Using this CUI browser, each thread sent 4 GET requests and 2 POST requests to the server. Using GET requests, a simulated user demanded for different web pages while using POST requests, a simulated user uploaded the image on web server. In order to mimic the real users' web surfing behaviour, some volunteer users accessed this website to read the tutorials and upload a sample image on web server. From this experiment, we then observed the average time delay between each GET and POST request sent by a real user. We found that on an average, users spent approximately 15 seconds on webpage having smallest size while they spent approximately 25 seconds on largest webpage[3]. This is obvious because of the presence of more information on a larger

[3]In [105] also, it was observed that users spend more time on a web page having larger size as compared to a smaller web page.

web page as compared to a smaller one and thus, users need more time to grasp the page contents. We also observed that, on an average, users spent 30 seconds on web page having image upload field. This was probably due to browsing the valid image file on the local computer and then uploading it to web server. Thus, depending on these observations, we chose appropriate sleeping time intervals ranging from 15 to 25 seconds between each GET request sent by a simulated user. We also chose a longer sleeping time interval of 30 seconds between each POST request sent by a simulated user. We also designated one computer as malicious client which generated anomalous flows in different time intervals of testing phase. Using this setup, we generated traffic for training and testing phase. Due to unavailability of any stable HTTP/2 parsing library, we wrote a program using jNetPcap [122] Java library to process the collected traffic and extract the required features.

**Training Phase:** During training phase, we collected 14 hours of normal HTTP/2 traffic from the web server configured in our testbed and used this traffic to generate normal HTTP/2 profile. This HTTP/2 traffic profile is then compared with the

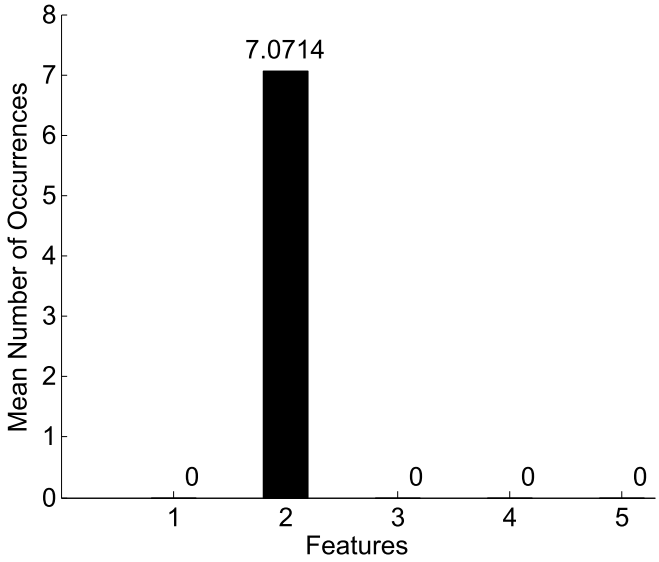



Figure 6.10: Normal HTTP/2 profile

profile generated in a particular time interval of testing phase to detect anomalies. For our experiments, we divided the training phase into smaller time periods, $\Delta T = 10$

minutes which resulted into 84 intervals in total. We created HTTP/2 normal profile using these 84 intervals. Figure 6.10 shows the generated normal HTTP/2 profile. This profile shows the mean occurrences of each feature in a time interval. The x-axis in the figure denotes different features and y-axis shows the occurrence count of each of these features. We can notice that except feature-2 (END_STREAM_FLAG reset), remaining features have a count of zero. The count for Feature-2 is non-zero due to the presence of few HTTP POST requests which could not upload the image successfully on the web server due to non graceful termination of underlying TCP connection due to some error.

**Testing Phase:** We generated another 14 hours of normal traffic from the same setup for testing purpose. We repeated this experiment by injecting different attacks at different rates along with normal requests raised by the script. In total there are seven different scenarios (one normal and remaining six containing attack instances) each with 84 intervals. The first scenario has all 84 normal intervals and other six scenarios have 24 intervals without any attack flows and remaining 60 intervals containing different attack flows. Table 6.10 shows experimental scenarios along with number of intervals with various attack flows injected in each case. For

Table 6.10: Performance of proposed anomaly detection scheme

| **Scenarios** | **Rate (Flows/ attack/ interval)** | **Overall Attack Rate** | **tp** | **fp** | **tn** | **fn** | **Recall** | **FPR** |
|---|---|---|---|---|---|---|---|---|
| Normal | 0 | 0 | 0 | 0 | 84 | 0 | 000.00% | 0.00% |
| Attack-1 and 2 | 3 | 6 | 5 | 0 | 24 | 55 | 008.33% | 0.00% |
| Attack-1 to 4 | 3 | 12 | 8 | 0 | 24 | 52 | 013.33% | 0.00% |
| All 7 Attacks | 2 | 14 | 26 | 0 | 24 | 34 | 043.33% | 0.00% |
| All 7 Attacks | 3 | 21 | 60 | 0 | 24 | 0 | 100.00% | 0.00% |
| All 7 Attacks | 4 | 28 | 60 | 0 | 24 | 0 | 100.00% | 0.00% |
| All 7 Attacks | 5 | 35 | 60 | 0 | 24 | 0 | 100.00% | 0.00% |

example fourth row in the Table indicates that all 7 attacks are injected at the rate of 2 flows per interval.

**Obtained Results:** We used time periods of 10 minutes to assess the detection performance. As described previously in Section 6.5, we compute $\chi^2$ test statistic for each time interval using Equation 6.1 to check for any significant departure of testing profile from the normal one. We can recall that chi-square test statistic computation requires dividing the difference of expected and observed values with expected value. However from the training profile of Figure 6.10 we can notice that 4 out of 5 features have a zero count. This results into an expected count of zero for these features and thus, a divide by zero scenario arises. To avoid this case, we increment these feature count by a small number (5 in this case).

We use Recall and FPR as performance metrics to evaluate detection performance of proposed scheme. Table 6.10 shows the detection performance of proposed scheme. We obtained these results at 0.05 significance level (see Appendix C). If obtained $\chi^2$ values for a particular time interval was greater than the threshold $\chi^2$ value, $H_A$ was accepted for that time interval. However, $H_N$ was accepted in case obtained $\chi^2$ value for that time interval was less than the threshold $\chi^2$ value. We can notice from the table that Recall increases with the increase in attack rate and 100 % recall was obtained when the total number of attack flows in a time interval exceeded 21. We can also notice that if the attack was launched at a very small rate, the detection performance is degraded . This is because the traffic profile of the time interval in which very small number of anomalous flows were present was almost similar to normal HTTP/2 profile. Thus, obtained $\chi^2$ value is less than the threshold value due to which the time interval is considered as normal, though it contains anomalous flows. In order to detect these cases, we can extend the monitoring period for longer duration and count the occurrences of these cases to signal the anomalous scenario.

**Sensitivity Analysis:** The detection performance of the proposed scheme depends on threshold chosen for $\chi^2$ value and also the window period chosen for

monitoring various features. Choosing appropriate $\chi^2$ threshold value and time period ($\Delta T$) is a critical and essential task to minimize and maximize the false and true detection cases respectively. Thus, we performed an experiment by varying both threshold $\chi^2$ value and the time period $\Delta T$ to analyze the sensitivity of the proposed scheme to these two parameters. To do so, we generated another 14 hours of traffic by injecting all 7 attacks at the rate of 1 flow/attack after every 300 seconds. We varied the time intervals in step size of 5 minutes ranging from 5 minutes to 25 minutes and took $\chi^2$ values at 7 different significance levels as $\alpha = 0.005, 0.01, 0.025, 0.05, 0.1, 0.25,$ and 0.5. Figure 6.11 shows that Recall rate increases with increase in both $\Delta T$ time period and significance level $\alpha$. We can notice that 100% Recall rate was obtained for

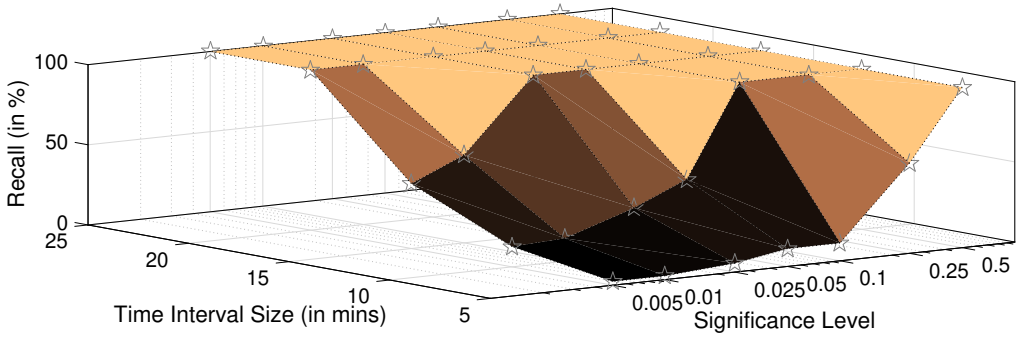



Figure 6.11: Recall

$\Delta T = 20$ and 25 minutes independent of $\alpha$ value while worst Recall rate (=3.18%) was obtained for $\Delta T = 5$ minutes and $\alpha = 0.005$. Figure 6.12 shows that FPR remains 0 for $\Delta T = 5$ minutes independent of $\alpha$ value. However, it increases if both $\Delta T$ time period and $\alpha$ are increased. In our experiments, we obtained worst FPR (=26.34 %) for $\Delta T = 25$ minutes and $\alpha = 0.5$. This analysis helps us in deciding the appropriate time period and threshold significance level. A larger time period provides better Recall rate but it also results in higher FPR rate (for larger significance levels) which is undesirable while a smaller time period badly affects the Recall rate (for smaller significance levels) though it results into very small FPR. Thus, choosing appropriate $\Delta T$ time period and $\chi^2$ value is essential for good detection accuracy of proposed

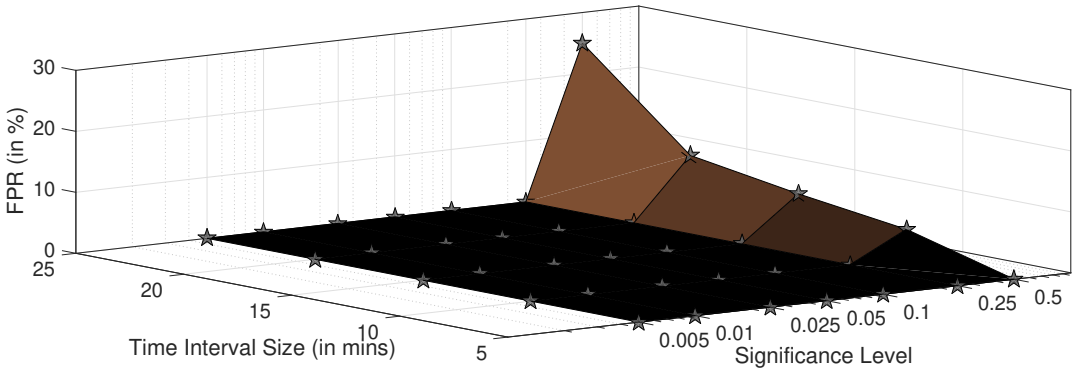



Figure 6.12: False Positive Rate

scheme.

### 6.6.5 Discussion

In this subsection, we discuss how our proposed detection scheme can be extended to handle non-stationarity in the generated HTTP/2 traffic profile. Subsequently, we also discuss how the scheme can be employed to detect Slow Rate HTTP/2 DoS attacks in case of encrypted HTTP/2 traffic using an intercepting proxy.

**Periodic Retraining:** It should be noted that, traffic profile during a busy time interval (belonging to office hour), $E_{busy}$, may significantly vary from the traffic profile during an idle time interval (night intervals), $E_{idle}$. As a result, comparing $E_{busy}$ generated during training phase with $E_{idle}$ generated in an interval during testing phase to detect any deviation may lead to higher false detection rates. To handle this non-stationary data distribution scenario, new HTTP/2 traffic profiles can be generated for different time intervals of a day. For example, two traffic profiles can be generated for a day during training phase - $E_{busy}$ for day time intervals while $E_{idle}$ for night intervals which can then be used for detecting deviations in the traffic received in different time intervals of a day during testing phase. Similarly, different traffic profiles can be generated for week and weekend days. The new traffic profiles can be generated using an online feedback mechanism in proposed

scheme. This mechanism enables the detection framework for its periodic retraining. In [123], authors presented a comprehensive study of various schemes which use different online feedback mechanisms to detect anomalies in wireless sensor networks in non-stationary environment. Feedback mechanisms used in these schemes can easily be adapted for periodic retraining of our proposed detection scheme also.

**Detecting Anomalies in Encrypted HTTP/2 Traffic:** The proposed detection scheme requires clear text HTTP/2 payload for further analysis. Thus if HTTP/2 traffic is encrypted, it must be first decrypted before submitting traces to the detector. Nevertheless, this can be easily achieved by implementing an intercepting proxy [124], [125], [126] before the web server designated to handle all HTTP/2 requests. Nowadays, most of the organizations are following this strategy to intercept the traffic coming to and going from their local network. The intercepting proxy server automatically creates SSL certificate used to encrypt/decrpyt traffic between client and proxy server while the original certificate sent by the web server is used to encrypt/decrypt the traffic between proxy and web server. One such example is shown in Figure 6.13. Using such setup, clear text HTTP/2 traces can be obtained which can then be submitted to detector for further analysis.

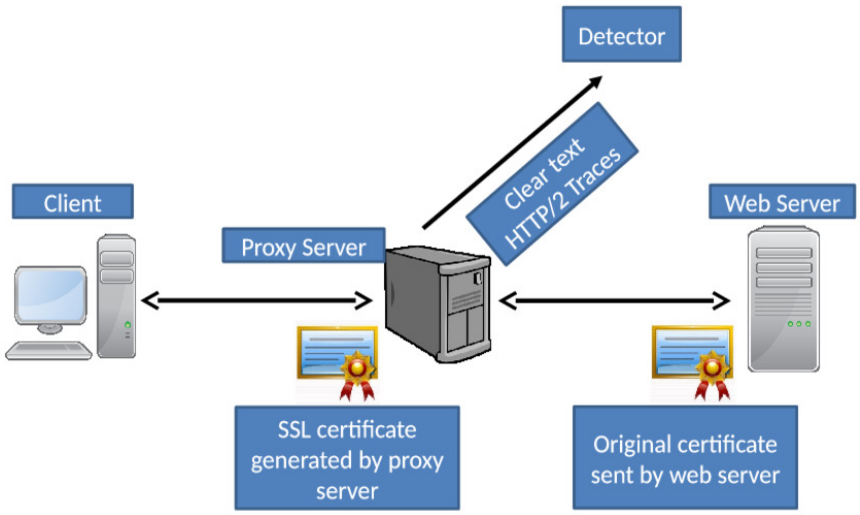



Figure 6.13: Intercepting HTTP/2 traffic using proxy server

There are few approaches in the literature which do not require payload analysis

for anomaly detection and thus, they can be adapted to detect anomalies in encrypted traffic also. These approaches detect anomalies in network traffic by observing hidden periodicities and hard-to-masquerade properties such as inter-packet arrival time, time gap between a request and its corresponding response, etc. These approaches can detect the proposed attack (and any other Slow Rate DoS attacks) as number of requests with no corresponding responses is very high in presence of proposed attacks. One such technique is proposed in [58] where authors collected information such as duration of a request and its corresponding response, the time gap between the end of a request and the start of the relative response and the time gap between the end of a response and the next request on the same established connection. Similar detection scheme is proposed in [19] which analyzes specific spectral features of traffic over small time horizons. In particular, proposed scheme tracks the number of packets received by a web server in a particular time period. In [61] also, authors proposed a scheme that monitors the number of packets received by a web server in different time horizons for anomaly detection. Nevertheless, to determine which of the five Slow Rate DoS attacks proposed in this chapter are launched and using what method, analyzing payload is the only possible solution.

## 6.7 Comparison of Slow Rate DoS Attacks in HTTP/1.1 and HTTP/2

As discussed earlier in Section 6.3.1, several works in the literature performed vulnerability assessment of HTTP/1.1 and presented Slow Rate DoS attacks against the protocol. In this section, we show that three of the attacks proposed in this paper are structurally similar (though not identical) to known Slow Rate DoS attacks against HTTP/1.1 and subsequently, compare both protocol versions based on these attacks. The experiments for the comparison was done while taking Apache 2.4.23 web server into account.

**Slow Read Attack and Attack-1:** Slow Read attack [127] against HTTP/1.1 server requires a malicious client to send a legitimate GET request after 3-way handshake and then immediately advertises a receiver window size of 0 byte. As a result, server stops sending data although it holds the connection in the hope of receiving a non-zero window size advertisement from the client. Client on the other hand, never advertises non-zero window size, thus, forcing server to wait for indefinite time. Similar to Slow Read attack, malicious client sends a SETTINGS frame with SETTINGS_INITIAL_WINDOW_SIZE parameter set to 0 to launch Attack-1. However, Apache HTTP/1.1 server is not vulnerable to Slow Read attack as it immediately closes the TCP connection which advertises a receiver window size of 0 bytes. On the other hand, Attack-1 is effective against Apache HTTP/2 server.

**Slow Message Body Attack and Attack-2:** In order to launch Slow Message Body attack against HTTP/1.1 server, malicious client sends complete POST request but message body is sent in smaller chunks so as to keep server waiting to receive complete message body. Usually, malicious client sends HTTP POST requests with a large *Content-Length* value as compared to the actual size of message body. On receiving this request, web server assumes that only a part of the message is received and rest of the message body is still pending. As a result, server keeps its resources busy waiting for the rest of message. To create DoS, malicious client establishes at least 150 connections and sends such POST requests from each connection. HTTP/1.1 server waits for an indefinite amount of time in the hope of receiving complete message body. This finally leads to DoS. Attack-2 also involves sending of smaller message chunks at regular intervals to maintain the connection in open state. Both Slow Message Body attack and Attack-2 are effective to create DoS scenario in HTTP/1.1 and HTTP/2 respectively.

**Slow Header Attack/SlowReq/Slowcomm and Attack-4:** Slow Header (also known as Slowloris) [5], SlowReq [128] and Slowcomm [129] attacks involve sending incomplete GET requests to the server by not transmitting the string

"\r\n\r\n" which denotes the end of a HTTP header. Instead, it sends bogus header fields at regular intervals to keep the connection alive. The malicious client establishes several connections and sends such incomplete GET requests from each connection to create DoS. Attack-4 against HTTP/2 server also involves sending of incomplete GET request header and then subsequent bogus headers to keep the connection alive. Slow Header attack can create DoS scenario if malicious client establishes at least 150 connections with the server. Before closing the connection, server waits for 300 seconds if no subsequent header parts are sent. However, this waiting time increases to 990 seconds if subsequent header parts are sent at regular intervals. SlowReq and Slowcomm attacks also cause similar effect on Apache server. Similarly, HTTP/2 is also vulnerable to Attack-4 as discussed in Section 6.4.4.

Table 6.11: Comparison of Slow Rate DoS attacks against HTTP/1.1 and HTTP/2

| **HTTP/1.1** | | | **HTTP/2** | | |
|---|---|---|---|---|---|
| **Attack Type** | **Connections Required to Create DoS** | **Max. Waiting Time** | **Attack Type** | **Connections Required to Create DoS** | **Max. Waiting Time** |
| Slow Read | Not Vulnerable | 0 | Attack-1 | 150 | 300 |
| Slow Message Body | 150 | Indefinite Time | Attack-2 | 150 | Indefinite Time |
| - | - | - | Attack-3 | 150 | 300 |
| Slow Header/ SlowReq/ Slowcomm | 150 | 990 | Attack-4 | 150 | Indefinite Time |
| - | - | - | Attack-5 | Not vulnerable | 5 |

Table 6.11 summarizes behaviour of HTTP/1.1 and HTTP/2 enabled web server in presence of different types of Slow Rate DoS attacks. We can notice that HTTP/1.1 is vulnerable to two types of Slow Rate DoS attacks while HTTP/2 is vulnerable to four

types of Slow Rate DoS attacks (For Attack-3 and Attack-5, there are no comparable attacks in HTTP/1.1) which are discussed in this chapter. Thus, it is evident that HTTP/2 has relatively more number of threat vectors as compared to Slow Rate DoS attacks against HTTP/1.1.

### 6.7.1 Comparison of Attacks in HTTP/1.1 and HTTP/2 with Encrypted Requests

We compared threat vectors in HTTP/1.1 and HTTP/2 by considering encrypted requests also. As discussed earlier, proposed attacks behaved similar against HTTP/2 irrespective of clear text or encrypted requests. In case of HTTP/1.1 also, we observed that effectiveness of Slow Rate DoS attacks remains same (independent of the type of request) as the timeout period and number of connections required to create DoS scenario did not depend on the clear text or encrypted requests. Thus, the comparison summary shown in Table 6.11 holds valid for encrypted HTTP requests as well.

## 6.8 Conclusion

In this chapter, we proposed novel Slow Rate DoS attacks against HTTP/2. These attacks can be launched by sending specially crafted HTTP/2 payloads to a web server. We performed an empirical evaluation of these attacks against all major web servers and found that majority of them are vulnerable to these attacks (tested with both clear text and encrypted traffic). Subsequently, we proposed an anomaly detection technique to detect the Slow Rate HTTP/2 DoS attacks which works by measuring the $\chi^2$ value between the expected and observed traffic pattern and showed that it could detect the attacks with high accuracy. We also compared HTTP/1.1 and HTTP/2 based on the similarity of threat vectors and showed that HTTP/2 has more threat vectors as compared to HTTP/1.1.

# Chapter 7

# Preventing Clock Synchronization in NTP's Broadcast Mode and Detection

## 7.1 Introduction

Network Time Protocol (NTP) [10] is one of the oldest protocol of the Internet and is used to synchronize clocks of computer systems on the Internet. Clock synchronization among computers is important for many reasons as inconsistencies of even fractions of a second can affect various core Internet services such as Domain Name System (DNS) resolution, interdomain routing using Resource Public Key Infrastructure (RPKI) and Transport Layer Security (TLS) [130], [131], [132], [14]. Thus, if an adversary is able to manipulate clocks of important systems on the Internet by obstructing NTP's operation, services to a large number of the Internet users will be affected. There have been several incidences in the past where clock synchronization failure led to breakdown of various services on the faulty systems, all at the same time. For example, two important NTP servers of U.S. Naval Observatory (USNO) were sent back in time by about 12 years on November 19, 2012 [133], causing outages at a variety of devices including Active Directory (AD) authentication servers

and routers [134]. NTP's potential for Distributed DoS (DDoS) [8] using reflection and amplification techniques is well studied by the security community. However, researchers have now started paying attention to find vulnerabilities and flaws in the protocol specification and implementation [13], [14], [38] itself.

In this chapter, we describe a new attack that exploits a vulnerability present in NTP's broadcast mode. The proposed attack prevents a NTP client from synchronizing its clock with a NTP broadcast server by sending spoofed NTP packets to the NTP client and the broadcast server. We also show how the proposed attack is effective in both authenticated and unauthenticated broadcast modes of NTP. To the best of our knowledge, this is the first attack proposed against NTP protocol since NTP's reference implementation's most recent version *ntpd* v4.2.8p11 [135] was launched on February 27th, 2018. As a first line of defense, we also propose a technique to detect spoofed NTP packets injected while launching the proposed attack. Our contributions in this chapter are:

**1.** We propose a new attack that can prevent a NTP client from synchronizing its clock with a NTP server.

**2.** We test the proposed attack in a real network and show that it is effective in both authenticated and unauthenticated modes of NTP.

**3.** We show how the proposed attack can prevent the time synchronization of computers configured in multicast mode also.

**4.** We propose a technique to detect spoofed NTP packets and test it in a real network setup by creating several spoofing scenarios and furnish the experimental results.

Rest of the chapter is organized as follows. We give an overview of NTP protocol and its working in Section 7.2. We explain few previously known attacks against NTP and techniques to counter these attacks in Section 7.3. In Section 7.4, we describe the working of proposed attack. We describe the technique to detect spoofed NTP packets in 7.5. In Section 7.6, we present the experiments performed to test the proposed attack against NTP's broadcast mode and the performance of detection technique to detect spoofed NTP packets. Finally, the chapter is concluded in Section 7.7.

## 7.2 NTP Protocol

NTP is in operation since 1985 and thus, is one of the oldest Internet protocols currently being used. The working of earlier versions of the protocol, NTPv1, NTPv2 and NTPv3 were described in RFC 1059 [136], RFC 1119 [137] and RFC 1305 [138] respectively while the working of current version of the protocol, NTPv4, is described in RFC 5905 [10] which is an *Internet Standards Track* document since June 2010. NTPv4 is backward compatible with previous versions of the protocol. NTPv4 includes a modified protocol header to accommodate the Internet Protocol version 6 (IPv6) address family. This version also includes improvements in various algorithms that extend the potential accuracy to the tens of microseconds with modern workstations and fast LANs [10]. Despite of changing the protocol specification over the years to make the protocol more robust and accurate, various NTP implementations today do not completely follow the specification. Instead, it is *ntpd*, the reference implementation of the protocol that practically defines it [14]. *ntpd* has been updated/modified several times in the last few decades to make it secure against various attack vectors found in the implementation over time. *ntpd* v4.2.8p10 and v4.2.8p11 are the two most recent versions launched on March 21st, 2017 and February 27th, 2018 respectively.

### 7.2.1 NTP Packet Structure

Different types of NTP packets exchanged during clock synchronization process have a common packet structure. This structure is shown in Figure 7.1. Various fields of the mode 5 packet are as follows.

**Leap Indicator (LI):** This is a 2-bit field used for warning of an impending leap second to be inserted or deleted in the last minute of the current month. The possible values are 0, 1, 2 and 3 where 0 represents no warning, 1 and 2 represents that the last second of the day has 61 or 59 seconds respectively and 3 represents that the clock is unsynchronized.

**Version (v):** This is a 3-bit field that represents the NTP version number. Currently, the version number is 4.

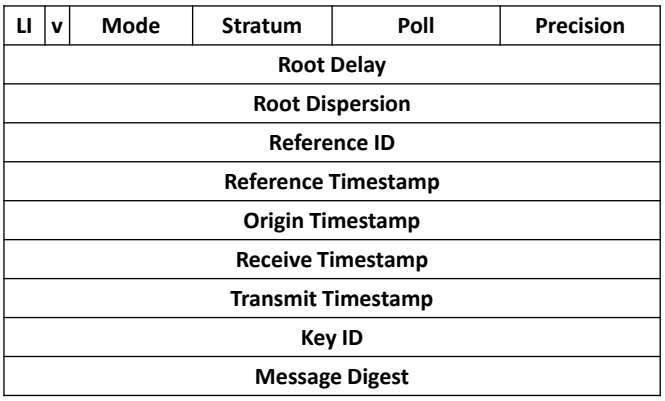



Figure 7.1: NTP packet structure

**Mode:** This is a 3-bit field that represents the NTP mode to which the packet belongs.

**Stratum:** This is an 8-bit field that represents the stratum of the NTP system. Stratum is a simple measure of the synchronization distance from the primary time source and its value ranges from 0 to 16 inclusive. Stratum 0 refers to reference clocks which receive true time from a dedicated transmitter or satellite navigation system. Stratum 1 systems are at the root of the NTP hierarchy and get the time directly from reference clocks while Stratum 16 indicates failure to synchronize.

**Poll:** This is an 8-bit field that represents the maximum interval between successive messages in log base 2 seconds.

**Precision:** This is an 8-bit field that represents the precision of the system clock in log base 2 seconds.

**Root Delay:** This is a 32-bit field that represents the total round-trip delay between the system and the reference clock

**Root Dispersion:** This is a 32-bit field that represents the total dispersion to the reference clock.

**Reference ID:** This is a 32-bit field identifying a particular server or reference clock to which the NTP system is synchronized.

**Reference Timestamp:** This is a 32-bit field that represents the time when the system clock was last updated.

**Origin Timestamp ($T_1$):** This is a 32-bit field that represents the time at the client

when a NTP request departed for the server. In broadcast mode, since the client does not send any NTP request and it is the server that sends unsolicited mode 5 packets at regular intervals, this field is set to NULL.

**Receive Timestamp ($T_2$):** This is a 32-bit field that represents the time at the server when the request arrived from the client. Since there is no corresponding request sent by the client, this field is also set to NULL.

**Transmit Timestamp ($T_3$):** This is a 32-bit field that represents the time at the server when the NTP packet left for the client.

**Key ID:** This is a 32-bit field that identifies the key used to authenticate the packet. There are multiple keys that can be configured on a NTP device and each key is identified by the Key ID.

**Message Digest:** This is a 128-bit field that contains the MD5 hash calculated over the contents of the NTP packet using a symmetric key $k$.

### 7.2.2 NTP Working

NTP is designed to operate under several modes - *client/server*, *broadcast*, *multicast* and *symmetric* modes. Also, all these modes are recommended to be processed by the same codepath of the NTP's implementation. Table 7.1 shows different NTP modes

Table 7.1: Different NTP modes

| **Mode** | **Synchronization** | **Exchanged Messages** |
|---|---|---|
| Client/server | Client synchronizes to lower stratum server | Mode3 and mode 4 |
| Symmetric | Peers at same stratum exchange time with each other | Mode 1 and mode 2 |
| Broadcast | Client synchronizes to a lower stratum broadcast server | Mode 5 |
| Multicast | Client synchronizes to a lower stratum multicast server | Mode 5 |

in which NTP systems can synchronize their clocks to other NTP systems' clocks

using a set of NTP messages. NTP's default mode of operation is client/server mode in which a NTP client sends timing queries to a set of NTP servers. This set of servers is typically static and configured manually. Stratum $s$ NTP systems act as servers that provide time to stratum $s+1$ systems. There are 16 levels in this stratum hierarchy such that stratum 0 refers to reference clocks which receive true time from a dedicated transmitter or satellite navigation system while stratum 1 refers to those NTP systems which are at the root of the NTP hierarchy and get the time directly from reference clocks and thus, are the most accurate NTP systems. Also, a failure to synchronize clock is represented as stratum 16. Thus, as we go down the hierarchy, the time accuracy reduces. This hierarchy is shown in Figure 7.2. In NTP's broadcast

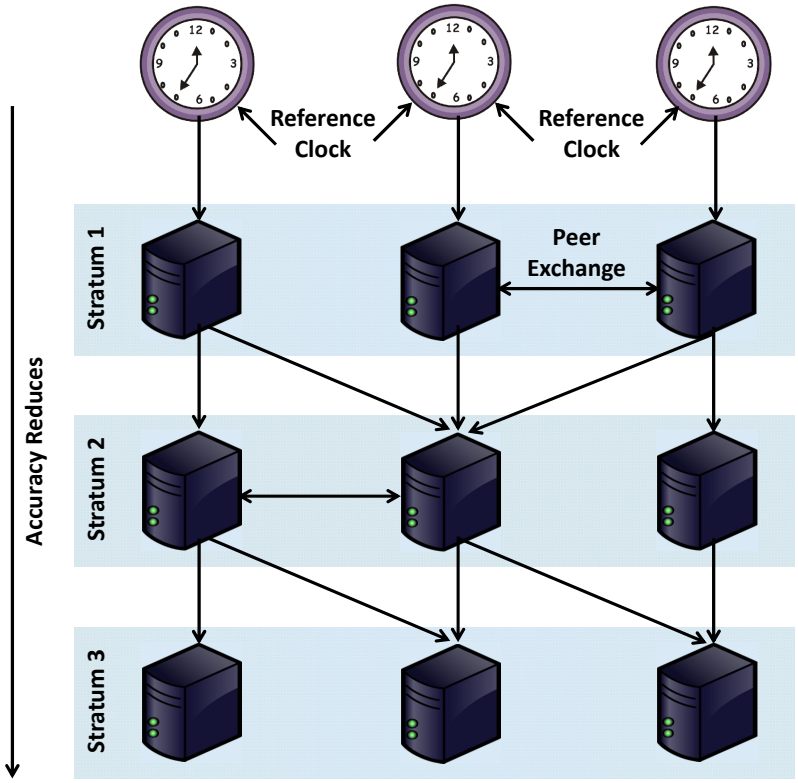



Figure 7.2: NTP hierarchy

mode, a set of clients listen to a broadcast server that periodically broadcasts timing information while in symmetric mode, NTP peers exchange timing information with each other. Since the attack proposed in this chapter exploits a vulnerability present

in the NTP's broadcast mode, we discuss the working of this mode of NTP operation in the next subsection.

### 7.2.3 NTP's Broadcast Mode

NTP's broadcast mode is mostly used in environments having few NTP servers and large number of potential clients. An NTP server can be configured to periodically broadcast what is known as *mode 5* NTP packets in the network. Similarly, NTP clients can be configured to accept and process these mode 5 packets. As soon as an unsynchronized broadcast client receives its first mode 5 packet, it mobilizes broadcast association with the NTP server. This association exists unless there is some error in NTP's broadcast mode operation or timeout. While mobilizing the association, the broadcast NTP client changes its mode to client/server mode and calculates the *path propagation delay* between client and server. On page 7 of RFC 5905 [10], following line is there:

> *"It is useful to provide an initial volley where the client operating in client mode exchanges several packets with the server, so as to calibrate the propagation delay..."*

While calculating path propagation delay in client/server mode, the client sends a *mode 3* NTP query to the broadcast server. This query is sent by a client to obtain clock information from a server in order to synchronize its clock with the server's clock. On receiving the mode 3 query, the server sends *mode 4* response to the client. This response is sent by the server to provide its clock information to the client. Using this response, the client calculates the path propagation delay and reverts back to broadcast mode and creates an ephemeral association with the broadcast server upon receiving further mode 5 packets. After this, the client regularly synchronizes its clock with the help of mode 5 packets periodically sent by the broadcast server.

### 7.2.4 Built-in Security Mechanisms in NTP

NTP facilitates some built-in security mechanisms to counter variety of attacks against the protocol. We discuss some of these security mechanisms in subsequent paragraphs.

**TEST1 [138]:** As discussed earlier in Section 7.2.3, a NTP client sends mode 3 query to a NTP server in order to synchronize its clock. In reply, the server sends mode 4 response to the client. On receiving this response, the client matches the transmit timestamp in the current mode 4 response to that of the last mode 4 response it received. If both timestamps match, the client marks the packet as duplicate and drops it. This precautionary check is known as TEST1. This check is typically performed to resist replay of a server's mode 4 response packet.

**TEST2 [138]:** NTP protocol requires the clients to check that the origin timestamp present in a server's mode 4 response matches with the transmit timestamp present in the corresponding mode 3 query sent by the client earlier. In this way, an off-path malicious client, that cannot observe the communication between the NTP client and the server, does not know the timestamp and thus can not spoof the packets. This is called as TEST2. We should also note that all NTP packets have UDP source port 123 instead of random source port numbers. That is the reason port number can not be used as a nonce to match a mode 4 response with the corresponding mode 3 query.

**Panic Threshold and Step Threshold [10]:** RFC 5905 defines a *panic threshold* PANICT and suggests that if the time difference between a NTP client's clock and a NTP server's clock (known as *offset* and discussed later in Section 7.4.1) to which the client is attempting to synchronize is greater than PANICT, NTP program should quit. On page 45 of RFC 5905, following line is there:

> *"...PANIC means the offset is greater than the panic threshold PANICT (1000 s) and SHOULD cause the program to exit..."*

This threshold prevents such attacks where a malicious client can attempt to tell a client to alter its local clock by a very large value so that the malicious client can create scenarios such as DNS cache and TLS certificate validity expiry which require time to be shifted by big steps (days or even years) in the past or future. RFC 5905 also defines a *step threshold* and suggests that a NTP client should accept a time shift larger than the step threshold (125 ms by default) but smaller than PANICT (1000 s by default).

**Kiss-o'-Death (KoD) Packet [10]:** RFC 5905 describes the use of a special packet called KoD to inform clients to stop sending packets that violate server access controls. For this purpose, the protocol specification defines several kiss codes [10] which must be inspected by recipient clients and take the action accordingly. One such kiss code is 'RATE' which asks a recipient client to reduce the rate with which it is sending mode 3 queries to the server. On page 25 of RFC 5905, following line is there:

> *"Rate exceeded. The server has temporarily denied access because the client exceeded the rate threshold."*

**Authenticated NTP's Broadcast Mode:** An NTP client listens to broadcast mode 5 packets sent by any NTP broadcast server in the network so an adversary can send spoofed mode 5 NTP packets and the client accepts it. Moreover, the origin timestamp field in mode 5 NTP packet sent by broadcast server is always NULL as NTP client does not send any query for broadcast packets. As a result, TEST2, that matches origin timestamp with the transmit timestamp present in the most recent query packet client sent to server, is not applicable in broadcast mode due to which NTP client can not validate the received mode 5 packets. Thus, NTP's broadcast mode is vulnerable to identity spoofing attacks where malicious client can spoof the identity of a broadcast server. To address this issue, RFC 5905 [10] recommends that NTP implementations should have some authentication mechanism to validate mode 5 NTP packets. The protocol standard also recommends that if authentication is

implemented, MD5 hash algorithm must be supported. On page 32 of RFC 5905, following line is there:

> *"There is no specific requirement for authentication; however, if authentication is implemented, then the MD5-keyed hash algorithm described in [RFC1321] must be supported."*

*ntpd* also facilitates provision of an authentication scheme that uses symmetric-key cryptography to validate mode 5 NTP packets. In particular, this authentication scheme appends an MD5 hash obtained by using a symmetric key $k$ and NTP packet contents to the Message Digest field of NTP packet as shown in Figure 7.1.

## 7.3 Prior Work

There are three recent works [13, 14, 38] in the literature that describe new attacks against NTP. These attacks exploit the vulnerability present in either the protocol specification or its implementation. In [13], authors proposed two attacks against NTP's authenticated broadcast mode. These attacks are as follows:

**1. Deja Vu (On-path Time-sticking Attack):** The first attack was a Man-in-The-Middle (MiTM) time-sticking attack (CVE-2015-7973) where it is assumed that a malicious client is present between a NTP broadcast server and a victim client but it does not possess the symmetric key that authenticates broadcast NTP messages. To launch the attack, the malicious client first collects and records a contiguous sequence of NTP server's broadcast packets and then replays this sequence of packets (while dropping the legitimate packets), over and over, to the victim client. As a result, the victim gets stuck at a particular time. Since the malicious client replays a sequence of packets instead of just one packet, the replayed packets pass TEST1 (as the victim client only matches the current transmit timestamp with that of the last packet). After attack disclosure, *ntpd* v4.2.8p6 and later versions were patched to mitigate this attack [13].

**2. Off-path DoS Attack:** The second attack (CVE-2015-7979) was a DoS attack

wherein an off-path malicious client can easily cause an error in the NTP operation of victim client by sending a badly authenticated mode 5 packet. As a result, the victim client is not able to collect enough good NTP packets from the broadcast server to update its local clock. The only challenge for the malicious client is to learn the IP address of the broadcast server to which victim client is synchronized. However, the malicious client can simply send the victim client a mode 3 query, and thus, the victim client sends back a mode 4 response that reveals the IP address of the broadcast server in its reference ID field. Following attack disclosure, appropriate patches have been added to *ntpd* v4.2.8p6 and later versions to mitigate this attack [13].

In another work [14], authors proposed several attacks against NTP's unauthenticated client/server mode. These attacks are as follows.

**1. Small-step-big-step Attack:** In this attack, a malicious client sends bogus small time shifts to a victim client and then, when the malicious client is ready, it can send a big time shift to the victim client, perform harmful activities, and finally send another big time shift that sets the victim client's clock back to normal. To do so, the malicious client prevents the victim client from ever making two contiguous clock updates (one after the other) of less than the step threshold due to which a variable *allow_panic* in the victim client's *ntpd* implementation remains TRUE. As a result, the victim client is always ready to take whatever bogus time the malicious client wants to offer to the victim client. This issue is captured in CVE-2015-5300 following which patches have been added to the vulnerable *ntpd* versions [14].

**2. Off-path DoS Attacks:** To launch these attacks, an off-path malicious client exploits the rate-limiting mechanism of NTP. *ntpd* versions earlier than 4.2.8p4 do not validate the origin timestamp present in the KoD packets (by comparing it with the transmit timestamp of mode 3 queries sent by the victim client to the NTP server earlier) due to which the malicious client can send KoD packets to the victim client which are spoofed to look like they are coming from the NTP server. Moreover, even if *ntpd* of the victim client does validate origin timestamp present in KoD packets, the off-path malicious client can simply send large number of mode 3 queries to the NTP server which are spoofed to look like they are coming from the victim client. As a re-

sult, the NTP server sends genuine KoD packets to the victim client due to which the victim client stops updating its clock, resulting into DoS. These attacks are captured in CVE-2015-7704 and CVE-2015-7705 following which the vulnerable *ntpd* versions were patched so as to validate the origin timestamp present in the KoD packets [14]. However, the attack in which malicious client sends large number of mode 3 queries to the NTP server which are spoofed to look like they are coming from the victim client is not patched at the time of writing [14].

**3. Other Known NTP Vulnerabilities:** In [38], a NTP vulnerability (CVE-2015-8139) was discussed which can be exploited by a malicious client to obtain the transmit timestamp $T_3$ of a pending mode 3 query sent by a client to a NTP server. Once the malicious client obtains this timestamp, it can craft a fake mode 4 response having origin timestamp as $T_3$ and bogus time information. Since this response is having valid origin timestamp, the client accepts and process the response due to which its clock is shifted to a wrong time. This vulnerability is not yet patched [38] and thus, the current versions of *ntpd* are still vulnerable. One more vulnerability (CVE-2015-8138) was discussed in this work wherein a malicious client can bypass TEST2 by spoofing server's mode 4 response packets with their origin timestamp set to zero. Following attack disclosure, this vulnerability has been fixed in *ntpd* v4.2.8p9 and later versions [38].

## 7.4 Proposed Attack

We found a vulnerability (CVE-2018-8956) in NTP's broadcast mode which can be exploited to launch an attack that can prevent a client configured in broadcast mode from synchronizing its clock with a NTP broadcast server. In this section, we first explain the vulnerability and how it can be exploited to launch the proposed attack. Subsequently, we also describe the procedure of attack execution in NTP's unauthenticated and authenticated broadcast mode. We make following two assumptions for the successful execution of proposed attack:

**1.** Ingress or egress filtering is not enabled on edge routers of victim client's or ma-

licious client's network respectively. This is required because the proposed attack requires sending spoofed NTP packets to the broadcast server and the client.

**2.** Malicious client has a prior knowledge of which all clients in the wild are using NTP's broadcast mode. The malicious client can scan IPv4 address space using NTP's *peers* command to check for the presence of NTP broadcast clients in the wild [13].

### 7.4.1 Vulnerability in the Protocol

As discussed earlier in Section 7.2.1, NTP packets exchanged between a client and server include four timestamps - Origin Timestamp ($T_1$) that determines client's time when it sent the request to the server, Receive Timestamp ($T_2$) that determines server's time when it received the request sent by the client, Transmit Timestamp ($T_3$) that determines server's time when it sent the response to the client and Destination Timestamp ($T_4$) that determines client's time when it received the response sent by the server[1]. NTP uses these timestamps to calculate *Offset* ($\theta$) which represents the time difference between the client's and server's clock and is given by

$$\theta = \frac{1}{2}((T_2 - T_1) + (T_3 - T_4)) \tag{7.1}$$

If the calculated offset $\theta$ is greater than the panic threshold PANICT, according to RFC 5905, *ntpd* should quit with a diagnostic message to the system log. However, this creates surface for *Small-step-big-step attack* [14]. This issue is captured in CVE-2015-5300 and *ntpd* v4.2.8p5 and versions released later were patched. We also tested two most recent versions of *ntpd* - v4.2.8p10 and v4.2.8p11 - and found that these versions do not quit on receiving a broadcast mode 5 packet whose $T_3$ is set to a timestamp such that the calculated $\theta >$ PANICT. However, the change in the implementation creates surface for another vulnerability as recent versions start recalculating path propagation delay by exchanging mode 3 and mode 4 packets with the NTP server on receiving mode 5 packets such that calculated $\theta >$ PANICT. Thus, to exploit this vulnerability, a malicious client can send such mode 5 packets

[1] $T_4$ is not present in the NTP packet structure and the client retrieves this timestamp directly from its own clock.

which are spoofed to look like they are coming from the genuine broadcast server. The victim client, on the one hand, starts sending mode 3 packets to recalculate the path propagation delay while on the other hand, the malicious client keeps sending large number of mode 3 packets at a very high rate to broadcast server which are spoofed to look like they are coming from the victim client. As a result, the broadcast server starts responding with KoD packets having 'RATE' code in response to both spoofed and genuine mode 3 packets sent by malicious client and broadcast client respectively. Since broadcast client does not receive mode 4 responses from the server, it is not able to calculate the path propagation delay. As a result, the broadcast client is not able to synchronize clock with the broadcast server.

**Attack Surface:** To quantify the effect of proposed attack, it is important to visualize the attack surface space. Recently, authors in [13] presented measurement results which show that approximately 18K IP addresses on the Internet have at least one broadcast association out of which approximately 9.8K IP addresses use broadcast associations exclusively. Moreover, 1,767 IP addresses were found to use multicast associations. Authors also mentioned that NTP's broadcast mode is typically implemented when the broadcast server and clients are present behind Network Address Translation (NAT) boxes due to which they could not be scanned and thus, these measurements do not provide exact percentage of broadcast-enabled clients in the wild. Thus, while attack surface for the proposed DoS attack is small, it is still non-negligible.

### 7.4.2 Preventing Time Synchronization in Unauthenticated Broadcast Mode

Consider a network topology having five entities as shown in Figure 7.3 - one $i^{th}$ stratum NTP broadcast server ($B_s$), two $(i+1)^{th}$ stratum victim clients ($V_c^1$ and $V_c^2$) configured to accept mode 5 NTP packets, one malicious client ($M_c$) and one genuine client ($G_c$)- such that $B_s$, $V_c^1$ and $V_c^2$ are part of the broadcast network $N_1$ while $M_c$

and $G_c$ are part of another network $N_2$ on the Internet. Here, $G_c$ is controlled by

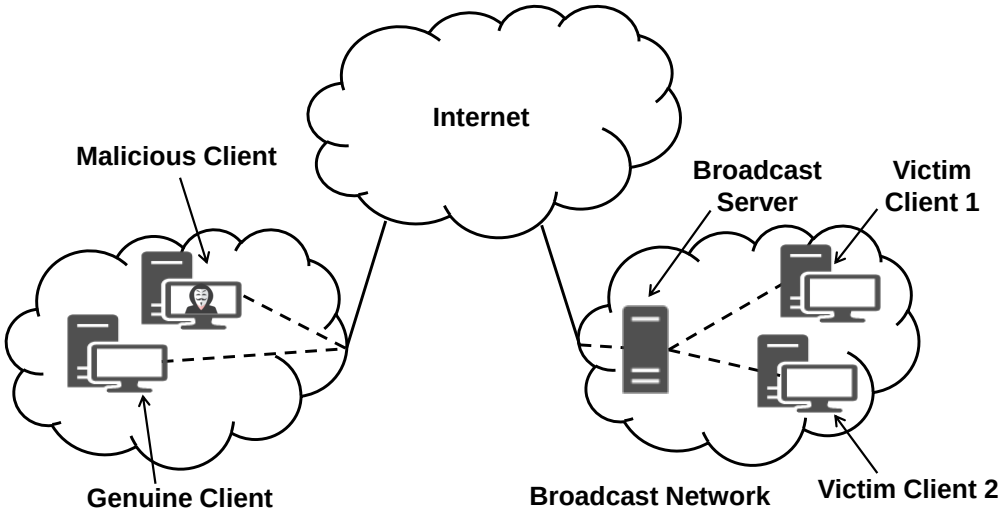



Figure 7.3: Topology for attack description

malicious client and its only purpose is to confirm that $V_c^1$ is not able to synchronize itself with $B_s$ while the attack is going on.

**Launching the Attack:** The steps involved while launching the attack are as follows:

**1.** In the first step, $V_c^1$ is let to successfully synchronize its clock with $B_s$. $V_c^1$ continuously listens to the mode 5 packets sent by $B_s$ at regular intervals. As soon as it receives the first mode 5 packet, it calculates the path propagation delay by exchanging mdoe 3 and mode 4 packets with $B_s$. After calculating the path propagation delay, $V_c^1$ reverts back to the broadcast mode and synchronizes its clock successfully.

**2.** $M_c$ sends mode 5 packets to $V_c^1$ at regular intervals which are spoofed to look like they are coming from $B_s$. $T_1$ and $T_2$ in these mode 5 packets are set to zero while $T_3$ is set to a timestamp such that $\theta$ calculated by client is greater than PANICT. Few moments later, $M_c$ starts sending mode 3 packets also to $B_s$ at regular intervals which are spoofed to look like they are coming from $V_c^1$.

**3.** On receiving spoofed mode 5 packets sent by $M_c$ in previous step, NTP daemon of

$V_c^1$ calculates $\theta$ using Equation 7.1. Since $\theta >$ PANICT, the daemon starts sending mode 3 NTP queries to $B_s$ to recalculate the path propagation delay.

**4.** Since $B_s$ continuously receives spoofed mode 3 NTP queries sent by $M_c$ in Step 2, $B_s$ sends KoD packets to $V_c^1$ in response to its mode 3 queries sent in Step 3 instead of valid mode 4 responses due to which $V_c^1$ is not able to calculate the path propagation delay. As a result, $V_c^1$ is not able to synchronize itself with $B_s$.

The exchange of various messages and their respective time of transmission for the successful execution of proposed attack are shown in Figure 7.4.

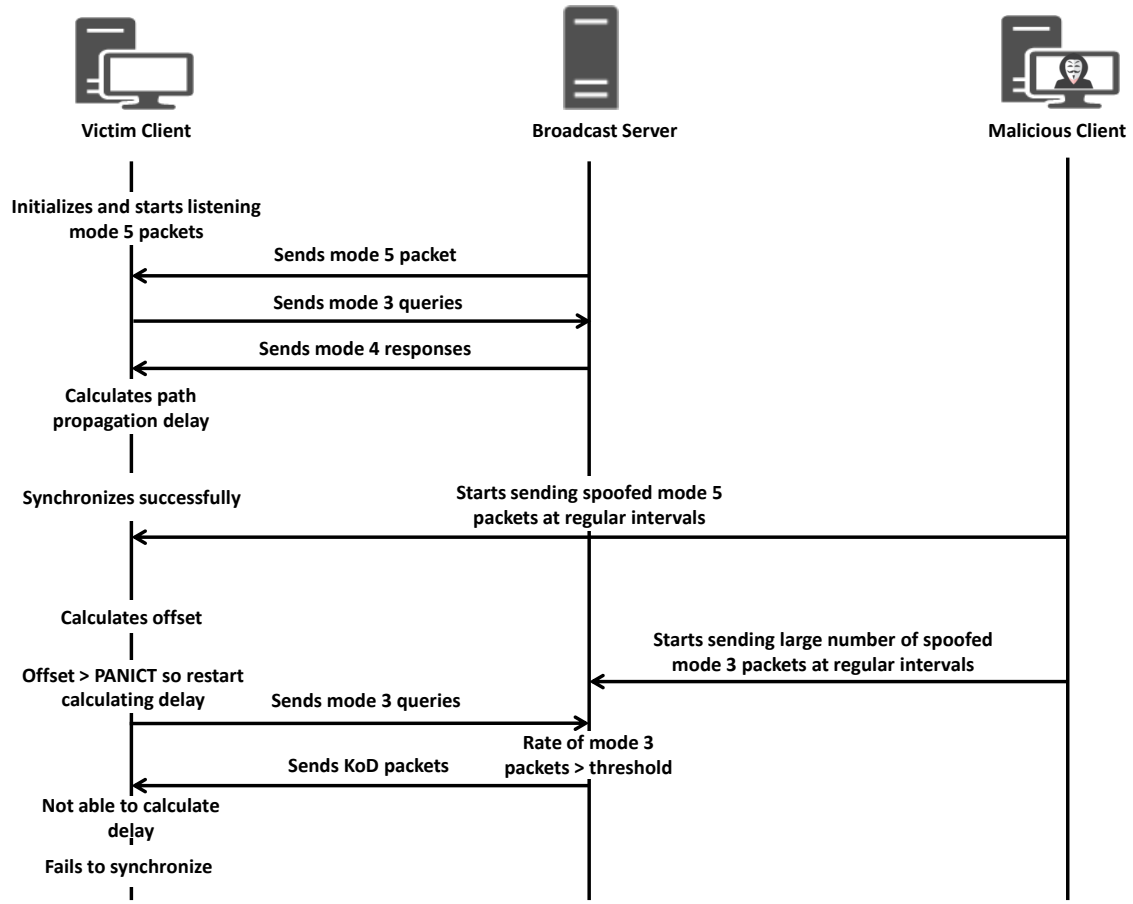



Figure 7.4: Exchange of messages while launching the attack

**Confirming Attack Success:** In order to confirm that $V_c^1$ is no longer able to synchronize itself with $B_s$, $G_c$ sends genuine mode 3 NTP queries to $V_c^1$ at regular

intervals and in return, $V_c^1$ sends back mode 4 NTP responses. In all these responses, if reference timestamp remains same, it means that $V_c^1$ is not able to synchronize itself with $B_s$.

**Targeting other clients:** Using the above steps to generate the attack, an adversary can target any client in that broadcast domain. This is done by placing appropriate source and destination IP addresses in mode 3 and mode 5 packets.

### 7.4.3 Preventing Clock Synchronization in Authenticated Broadcast Mode

To describe the attack procedure in NTP's authenticated broadcast mode, we consider two different scenarios - 1) $M_c$ is part of the broadcast network $N_1$ as shown in Figure 7.5 or 2) $M_c$ controls a slave $S$ in the broadcast network $N_1$ as shown in Figure 7.6. We now discuss how the proposed attack is effective in both these scenarios.

**1. When $M_c$ is part of the broadcast network $N_1$:** In this scenario, as mentioned

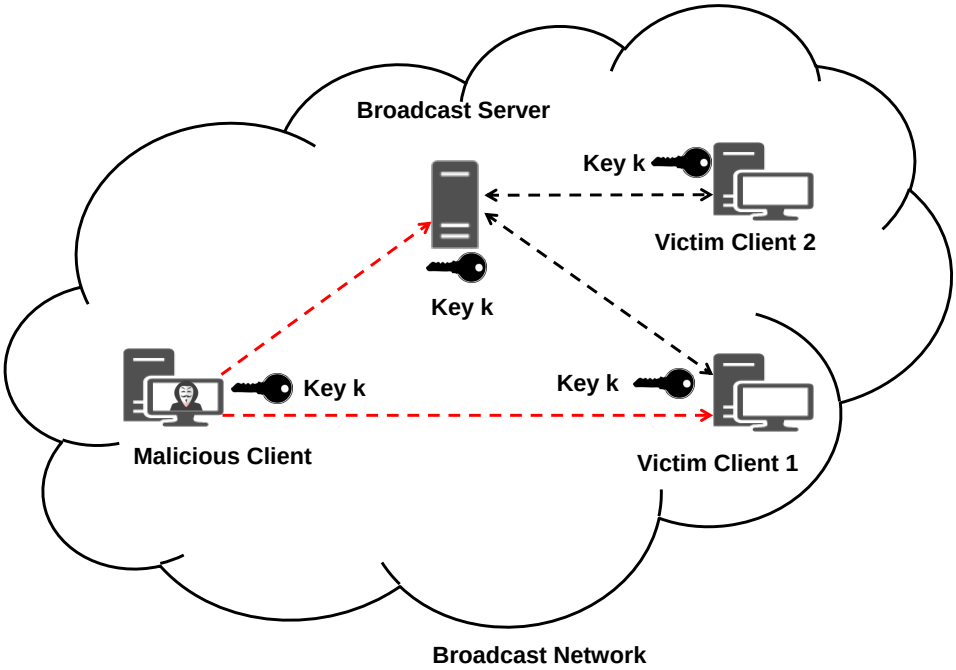



Figure 7.5: Malicious client is part of the broadcast network

in [139], [140], $M_c$ also possesses the same key $k$ as $B_s$, $V_c^1$ and $V_c^2$. Thus, $M_c$ can easily spoof identity of $B_s$ and send broadcast mode 5 packets containing bogus time to $V_c^1$ and $V_c^2$. Similarly, $M_c$ can also spoof identities of $V_c^1$ and $V_c^2$ to send large number of mode 3 packets to $B_s$. These packets are accepted by the destination entities as the MD5 hash appended to the packets are generated using $k$. Thus, by following the attack procedure discussed in Section 7.4.2, $M_c$ can prevent both $V_c^1$ and $V_c^2$ from synchronizing their clock with $B_s$.

**2. When $M_c$ controls a slave in $N_1$:** In this scenario, we assume that $M_c$ controls a slave in $N_1$ that can capture one copy each of type mode 5 packet sent by $B_s$ and mode 3 query sent by either $V_c^1$ or $V_c^2$ to $B_s$[2]. Once slave captures the copies of required packets, it forwards these copies to $M_c$. After receiving copies of mode 3 and

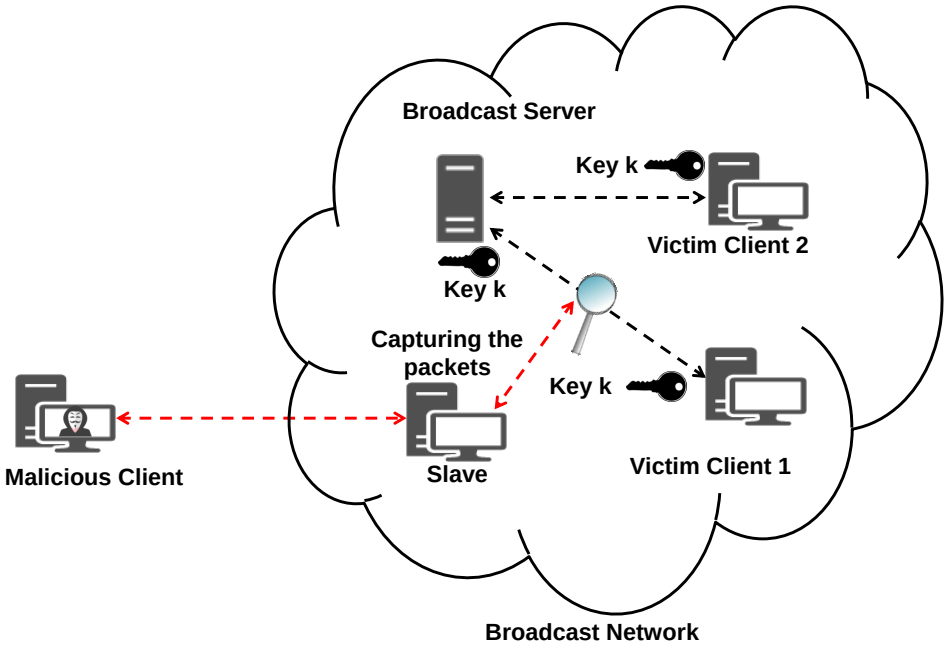



Figure 7.6: Malicious client controls a slave in the broadcast network

mode 5 packets, $M_c$ can launch the attack from anywhere in the wild but at a time $T_f$ such that $\theta$ calculated by $V_c^1$ at $T_f$ is greater than PANICT. To target $V_c^1$, $M_c$ continuously replays copies of mode 5 and mode 3 queries at regular intervals. The

[2]Unlike unauthenticated mode, adversary needs a copy of these packets to meet authentication check with $k$.

copies of mode 5 NTP packets are sent to IP address of $V_c^1$ with source IP address of $B_s$ while copies of mode 3 query are sent to $B_s$ with source IP address of $V_c^1$. Since $M_c$ does not change any field in the NTP header of captured mode 5 and mode 3 packet, the destination entities of these packets accept it as the MD5 hash appended to the packets are correct. Thus, the attack procedure discussed in Section 7.4.2 prevents $V_c^1$ from synchronizing itself with $B_s$ in NTP's authenticated broadcast mode also. Likewise, $M_c$ can prevent $V_c^2$ from synchronizing itslf with $B_s$ by placing appropriate source and destination IP address in IP header of mode 5 and mode 3 NTP packets. The exchange of various messages and their respective time of transmission for the successful execution of proposed attack against $V_c^1$ are shown in Figure 7.7.

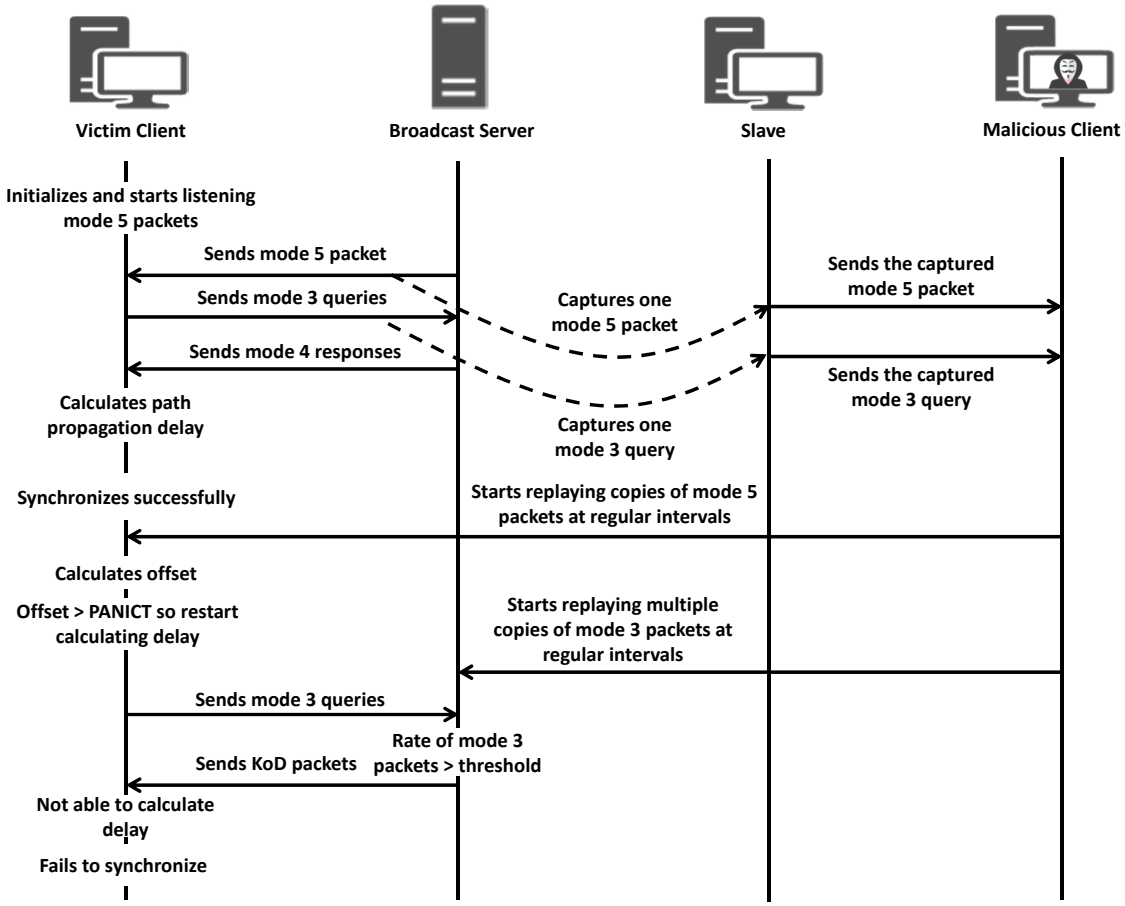



Figure 7.7: Exchange of messages while launching the attack in second scenario

In the previous three subsections, we described a vulnerability present in NTP

broadcast mode and an attack to exploit this vulnerability in order to prevent a client from synchronizing its clock with a NTP broadcast server in unauthenticated and authenticated mode. This attack requires repeatedly sending spoofed mode 3 and mode 5 packets to broadcast server and victim client respectively. Since preventing clock synchronization can have grave security implications on Internet's crucial infrastructure such as Domain Name System (DNS) resolution, interdomain routing using Resource Public Key Infrastructure (RPKI) and Transport Layer Security (TLS) [131, 130, 132, 14], we argue that the proposed NTP attack should be detected well before it affects the other core services of the Internet. Thus, we also propose a technique that can detect spoofed NTP packets injected during execution of proposed attack. In the next section, we describe the working of our proposed detection technique.

## 7.5 Proposed Detection Technique

Our proposed detection technique is an active verification technique where the recipient of a NTP packet initiates a verification process to verify the authenticity of the source IP address of an NTP packet. The source IP address of the NTP packet is verified by sending a TCP SYN packet as a probe request. This TCP SYN packet has a TTL value which is same as the calculated number of hops the NTP packet being verified has traversed to reach the destination. Based on the received probe response, a decision is made about the authenticity of the original packet. Our proposed active verification technique is based on the following assumptions.

**1.** A client attempting to verify itself does not spoof its IP address.

**2.** Majority of communication routes on the Internet are stable [141, 142] and symmetric [143] in nature.

**3.** Hop-count stability is dictated by the end-to-end routing behaviours [144] in the Internet.

**Previously known Methods which check TTL Inconsistencies:** There are few approaches [50, 51] in the literature that also detect spoofed packets on

the premise of predicting the initial TTL value based on received TTL value of packets. We take motivation from these techniques and thus, our proposed approach also checks for inconsistencies in the TTL value of packets coming from a source to detect spoofed NTP packets. Our approach is different from these techniques in the following way:

**1.** Our proposed active verification method can verify all unknown/new packets. However, previously proposed technique straightaway drops all unknown packets in case of traffic overload, thereby create more collateral damage.

**2.** Earlier approaches create table having entries of the seen IP addresses mapped to the corresponding hop count. This table is used to verify the packets received during the attack time. However, our proposed scheme verifies the packets by actively probing the source from which packets are received.

**3.** Previous approaches create table once and use it subsequently for TTL comparison. This table is updated only by packets from established TCP connections. Thus, if a client sends UDP packets (which carry NTP datagrams), the change in hop count is not updated in the table and thus, the packets are likely to be detected as spoofed ones. However, our approach does not differentiate the packets based on the transport layer protocol.

In the next subsection, we describe different scenarios based on the probe reply received in response to the TCP SYN probe sent to the source IP address of a NTP packet being verified. In each scenario, we discuss how a decision can be taken about the authenticity of NTP packet.

### 7.5.1 Different Scenarios

As discussed earlier in last subsection, majority of the communication routes on the Internet are stable. Thus, two packets sent from the same source over a short span of time should traverse same route and thus, same number of hops to reach a destination. As a result, the TTL values present in the received two packets should be equal. If there is inconsistency between the received TTL values, either of the two packets is sent from a different source which is spoofing the IP address of the genuine

source. Based on this, our detection approach verifies spooofed NTP packets. In particular, on receiving a NTP packet, we first extract the received TTL value from the IP header of the packet carrying NTP datagram. After this, the initial TTL value set by the sender's operating system is predicted. This prediction is possible due to two reasons. First, many operating systems use only a few selected initial TTL values such as 32, 64 and 128 and second, any two clients connected to Internet are not more than 30 hops away [52]. Thus, for example, if received TTL value is 2, the predicted initial TTL value should be 32. This is because if predicted TTL value would be 64 or 128, the number of hops taken by the NTP packet would be either 64-2=62 or 128-2=126 which is not possible (as maximum hops are not likely be more than 30). In this way, the initial TTL value is predicted. We then calculate the number of hops taken by the NTP packet to reach the destination by subtracting received TTL value from the predicted initial TTL value. After this, a TCP SYN probe is crafted such that its destination IP address is set to the IP address of the sender of the NTP packet and its TTL field is set to the calculated number of hops taken by the NTP packet received earlier. Once the probe is sent, we wait for a particular time to receive the corresponding probe response. Depending on what response is received, there are four possible scenarios which are discussed in subsequent paragraphs. The notations used to describe these scenarios are shown in Table 7.2.

Table 7.2: Notations

| **Notation** | **Description** |
|---|---|
| $Spoof\_IP$ | Spoofed IP address |
| $VC$ | Victim client whose IP address is spoofed |
| $MC$ | Malicious client which is spoofing the IP address |
| $RC$ | Recipient client which receives the packet to be verified and initiates probe |
| $GC$ | Genuine client is some other client on the Internet. |

**Normal Scenario:** Figure 7.8 shows the normal scenario in which a NTP

packet sent by $GC$ is being verified. The different events observed while verifying this IP packet are numbered and shown with different colors in Figure 7.8. As shown in

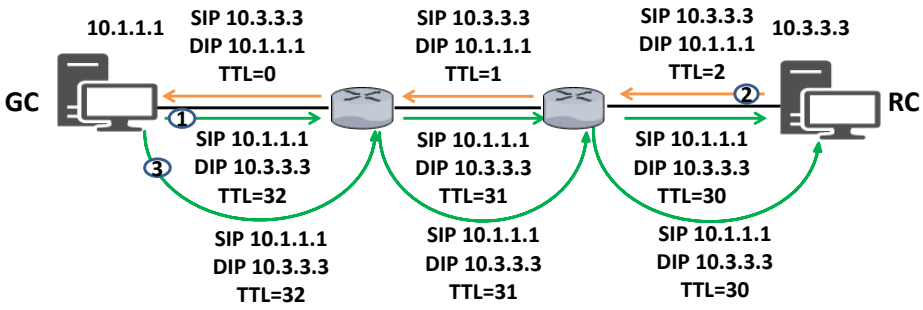



Figure 7.8: Occurrence of different events in case of normal scenario

the figure, the first event (packet) is the NTP packet sent by $GC$ to $RC$ with initial TTL value of 32 in its IP header. The received TTL value at $RC$ is 30. Thus $RC$ calculates the number of hops between $RC$ and $GC$. $RC$ then sends a probe request (TCP SYN packet) with TTL value equal to the calculated number of hops, i.e. 2 in this case. This is the second event (packet). The third event is the probe reply from the correct source $GC$ in this case with received TTL value of 30 at $RC$. Since the TTL value in IP header of the packet carrying NTP datagram and probe reply is same, the NTP packet received earlier is a normal one.

**Spoofing Scenario-1:** Figure 7.9 shows the first spoofing scenario where $VC$ is actually using the IP address $Spoof\_IP$ (10.1.1.1) spoofed by $MC$ and $VC$ is just two hops away from $RC$ while $MC$ is four hops away from $RC$. $MC$ sends a spoofed NTP packet to $RC$ with initial TTL value of 32. At $RC$, the received TTL value in this packet is 28, thus $RC$ calculates number of hops between it and $MC$ is 4. $RC$ sends a probe request with TTL value 4 and destination IP address as $Spoof\_IP$ (10.1.1.1). We can notice that probe packet is received by $VC$ rather than $MC$. $VC$ replies with a probe reply having initial TTL value of 32. When this reply reaches $RC$, the received TTL value is 30. Thus, the current TTL value (30) does not match with the TTL value (28) present in the previously received NTP packet and hence, difference in number of hops traversed to reach $RC$ can be identified.

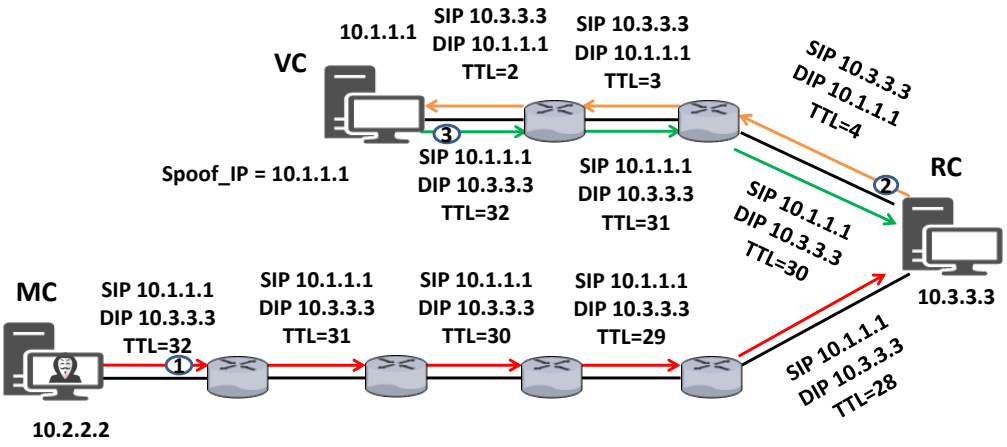



Figure 7.9: Occurrence of different events in case of spoofing scenario-1

**Spoofing Scenario-2:** Figure 7.10 shows the second spoofing scenario where $VC$ is actually using the IP address $Spoof\_IP$ (10.1.1.1) spoofed by $MC$ and $VC$ is four hops away from $RC$ while $MC$ is just two hops away from $RC$. $MC$ sends a

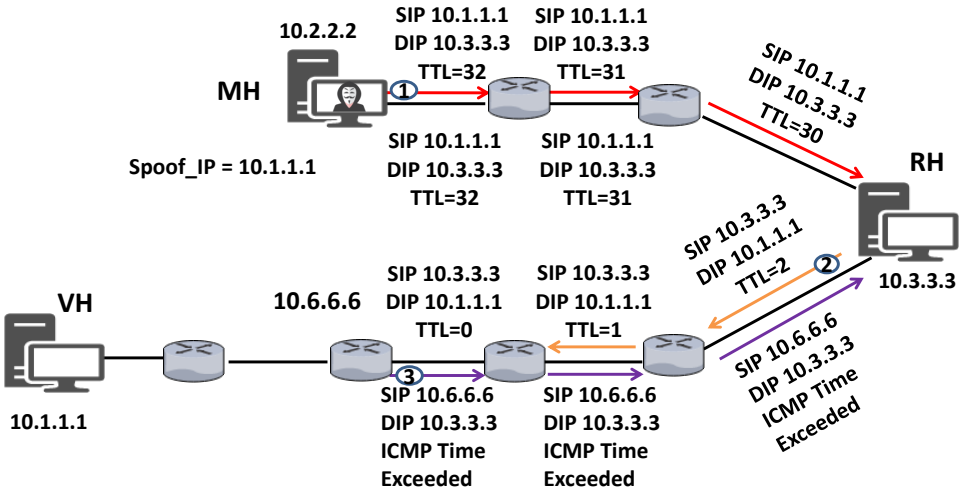



Figure 7.10: Occurrence of different events in case of spoofing scenario-2

spoofed NTP packet to $RC$ with initial TTL value of 32. At $RC$, the received TTL value in this packet is 30, thus $RC$ calculates number of hops between it and $MC$ is 2. $RC$ sends a probe request with TTL value 2 and destination IP address as $Spoof\_IP$ (10.1.1.1). In this case, probe packet does not reach the client $VC$. An intermediate

router (10.6.6.6) drops the probe packet and an ICMP packet (TTL expired) is sent to $RC$ indicating that the probe packet is dropped because its TTL reaches 0.

In the next subsection, we model these different scenarios using Discrete Event Systems (DES).

### 7.5.2 Discrete Event System Modeling

Discrete Event System (DES) requires creation of state diagrams for sequence of events under normal and spoofing scenario conditions. Subsequently, DES detector which is a combined state model with suitable labels is constructed. In this subsection, we present a DES formal model used to describe various normal and anomalous events of IP spoofing cases in NTP operation. We use the notations shown in Table 7.3 to describe our DES model. The DES model used for representing the communication behavior under normal and spoofing scenarios is a seven tuple $\langle \Sigma, S, S_0, V, C, t, F \rangle$, where

$\Sigma$ is the set of events,

$S$ is the set of states,

$S_0 \subseteq S$ is the set of initial states,

$V$ is the set of model variables,

$C$ is the set of clock variables

$t$ is the set of transitions and

$F \subseteq S$ is a set of final states.

Each variable $v_i \in V$ $(1 \leq i \leq n)$ takes values from a corresponding well defined domain $D_i$. The clock variables take values in non-negative reals $R$. An invariant condition on a clock variable $C$, denoted as $\Phi(C)$, is a linear inequality or equality of the variable with a non-negative real. Each transition $t_i \in t$ of the DES model is a seven tuple $\langle s_i, s_j, \sigma, \phi(V), \Phi(C), Assign(V), Reset(C) \rangle$, where $s_i$ is the source state, $s_j$ is the destination state, $\sigma$ is an event (on which the transition is fired), $\phi(V)$ is boolean conjunction of equalities of a subset of variables in $V$, $\Phi(C)$ is the invariant condition on a subset of clock variables, $Assign(V)$ is a subset of model variables and assignments with values from their corresponding domains and $Reset(C)$ is a

Table 7.3: Notations used in DES model

| Notation | Description |
|---|---|
| $NTP$ | NTP packet |
| $IP\_SRC$ | Source IP address of a NTP packet in question |
| $IP\_DST$ | Destination IP address of a NTP packet in question |
| $IP\_TTL$ | Residual TTL or hop count in the NTP packet |
| $PRQ$ | Probe ReQuest (TCP) packet |
| $PRQ\_IPS$ | Source IP address of a Probe ReQuest packet |
| $PRQ\_IPD$ | Destination IP address of a Probe ReQuest packet |
| $PRQ\_TTL$ | TTL or hop count set in a Probe ReQuest packet |
| $PRSP$ | Probe ReSponse (TCP) packet |
| $PRSP\_IPS$ | Source IP address in a Probe Response Packet |
| $PRSP\_IPD$ | Destination IP address in a Probe ReSponse Packet |
| $PRSP\_TTL$ | Residual TTL or hop count in a Probe ReSponse Packet |
| $ICMP$ | ICMP packet carrying information about dropped probe packet |
| $ICMP \rightarrow IPD$ | Destination IP address of a dropped probe packet whose details are carried in the ICMP packet |
| $ICMP \rightarrow IPS$ | Source IP address of dropped probe packet whose details are carried in the ICMP packet |

subset of clock variables to be reset. In the model, we define a set of events as $\Sigma = \{NTP, PRQ, PRSP, ICMP\}$ which signify arrival of NTP packet at $RC$, subsequently probing the source IP address of the packet, receiving probe response packet and arrival of an ICMP packet at the client. All these events are noticed on $RC$. The model variable set $V = \{SrcIP, RCVD\_TTL\}$ has respective well defined domains as $D_{SrcIP} = \{X.X.X.X | X \in \{0, 1, 2, \cdots 255\}\}$ and $D_{RVCD\_TTL} = \{N : 0 \leq N \leq 256\}$. There is a clock variable $y$ which keeps track of event timings and it is used to match and restrict events happening within the time threshold $\eta$. The event diagrams with DES model corresponding to event diagrams of Figures 7.8, 7.9 and 7.10 are shown in Figures 7.11, 7.12 and 7.13 respectively. We describe DES models under normal

and different spoofing scenarios in next three paragraphs.

**DES Model under Normal Scenario:** DES event model under normal operation is shown in Figure 7.11. Initially, the system is in start state $S0$ waiting for

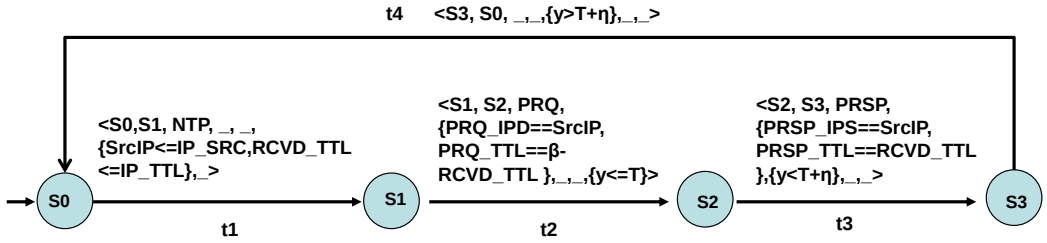



Figure 7.11: DES model under normal scenario

the event to occur. In this state, the first event occurs which corresponds to receiving an NTP datagram within an IP packet. The transition from $S0$ to $S1$ shows this event. In the process of transition, the source IP address of received NTP packet is assigned to the model variable $SrcIP$ and similarly $RCVD\_TTL$ is assigned the value of TTL value seen in the IP header of the packet. In the state $S1$, the system sends a probe request $PRQ$ (a TCP SYN packet) within an IP packet and it corresponds to the second event. The transition from $S1$ to $S2$ shows this event. This transition also has couple of invariant conditions to be met. First condition is that the destination IP of probe packet is $SrcIP$ and second condition is that the TTL value of the probe packet is set to the calculated number of hops the packet has traversed to reach receiver from the sender. This is calculated as guessed initial TTL value ($\beta$) minus the $RCVD\_TTL$ value in the previous event. There is no constraint on the time value in this case as this is the first event where active verification begins. On this transition to keep track of timing, a timer is initialized by assigning current clock value $T$ to the timer variable $y$. The third event is the receiving of probe response which has the invariant constraint of having the source IP address and received TTL value as $SrcIP$ and $RCVD\_TTL$ respectively. There is a time constraint on this event that it should happen within $T + \eta$ time which is greater than round trip time

value between any two clients. These three events correspond to the three events shown in Figure 7.8. It may be noted that there is a transition from final state to initial state after $T + \eta$, this is denoted as renewal process where the transition graph created is terminated.

**DES Model under Spoofing Scenario-1:** The DES model for spoofing scenario-1 is shown in Figure 7.12. In this scenario also, the first two events, i.e.,

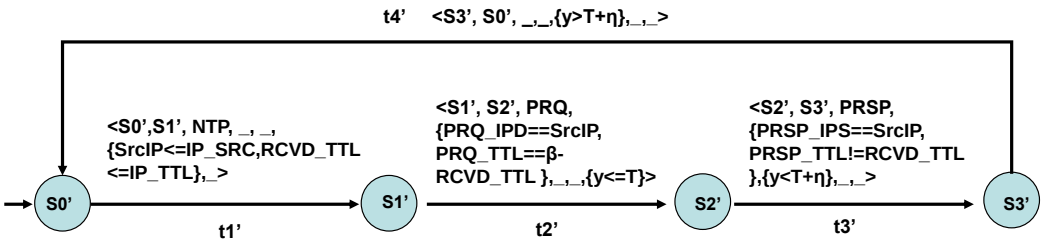



Figure 7.12: DES model under spoofing scenario-1

receiving NTP packet and sending a probe request are same as in previous case (normal scenario). However, the difference lies in the third event which corresponds to receiving a probe response which has the invariant constraint of having the source IP address as $SrcIP$ and received TTL value greater than $RCVD\_TTL$. There is a time constraint on this event which should happen within time of $T + \eta$ where $\eta$ is a round trip delay between any two clients. Thus, the probe response event in this case signifies an anomalous event.

**DES Model under Spoofing Scenario-2:** The DES model for spoofing scenario-2 is shown in Figure 7.13. From spoofing scenario-1, the only difference in this scenario lies in the third event which corresponds to receiving an ICMP packet with the message "TTL value exceeded" which has the invariant constraint of having the source IP address as $SrcIP$. There is a time constraint on this event which should happen within time of $T + \eta$ where $\eta$ is a round trip delay between any two clients.

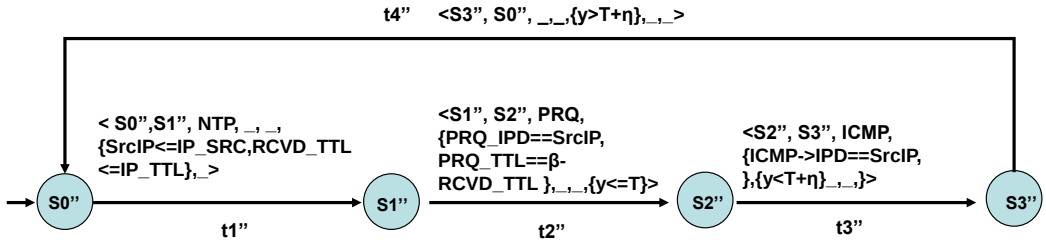



Figure 7.13: DES model under spoofing scenario-2

**Detector:** Once DES models for all possible scenarios of anomalous and normal cases are designed, a detector is constructed as shown in Figure 7.14. This DES detector is

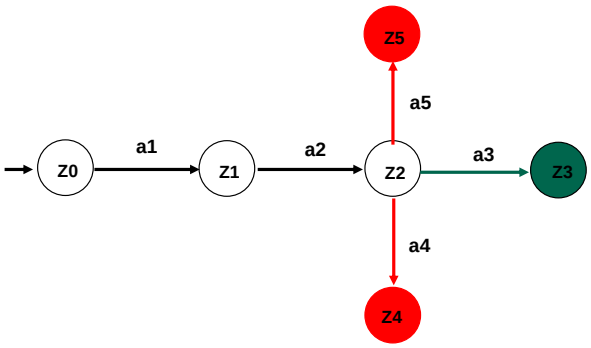



Figure 7.14: Detector

a state estimator and is created by identifying common transitions from multiple DES models. The common transitions are called as measurement equivalent transitions. Formally, two transitions $t_1 = s_i, s_j, \sigma_1, \phi_1(V), \Phi_1(C), Reset_1(C), Assign_1(V)$ and $t_2 = s_i^{'}, s_j^{'}, \sigma_1^{'}, \phi_1^{'}(V), \phi_1^{'}(C), Reset_1^{'}(C), Assign_1^{'}(V)$ are equivalent if $\sigma_1 \equiv \sigma_1^{'}$, $\phi_1(V) \equiv \phi_1^{'}(V)$,$\Phi_1(C) \equiv \Phi_1^{'}(C)$, $Reset_1(C) \equiv Reset_1^{'}(C)$, and $Assign_1(V) \equiv Assign_1^{'}(V)$. The list of the measurement equivalent transitions from various event diagrams and their correspondence to the transitions in the detector (Figure 7.14) is as follows.

**1.** $a_1 = t_1 = t_1^{'} = t_1^{''}$. This transition corresponds to the event of receiving a NTP packet.

**2.** $a_2 = t_2 = t_2^{'} = t_2^{''}$. This transition corresponds to the event of sending a probe

request to the source IP address of the NTP packet.

**3.** $a_3 = t_3$. This transition corresponds to the event of receiving a probe reply with TTL value same as that of the NTP packet whose authenticity is being verified.

**4.** $a_4 = t_3^{'}$. This transition corresponds to the event of receiving a probe reply with TTL value different from that of the NTP packet.

**5.** $a_5 = t_3^{''}$. This transition corresponds to the event of receipt of an ICMP error message in response to the probe request sent to check the authenticity of the NTP packet received earlier.

The detector is denoted as a directed graph $O =< Z, A >$, where Z is the set of detector states, referred as O-states, and A is the set of detector transitions, called O-transitions. Each O-state $z \in Z$ comprises of a subset of equivalent model states representing the uncertainty about the actual state and each O-transition $a \in A$ is a set of equivalent model transitions denoting the uncertainty of the actual transition that occurs. In the detector shown in Figure 7.14, an O-state which contains only model states corresponding to the normal scenario is called Normal certain O-state. Similarly an O-state corresponding to a spoofing scenario is called attack-certain O-state. The remaining states are called uncertain states. In this case, $z_3$ (green colour) is the only normal certain state, $z_4$ and $z_5$ (red colour) are attack certain states while remaining states are uncertain states.

## 7.6 Experiments and Discussion

We tested the proposed attack and the detection approach to detect spoofed NTP packets in a real network setup. In this section, we first describe the experiments with the proposed attack and then we present the experiments performed to test the performance of our detection approach.

### 7.6.1 Experiments with the Proposed Attack

The testbed setup in which we performed the experiments to test our proposed attack is shown in Figure 7.15. Further details about the setup is described in the

next paragraph.

**Testbed Setup:** The testbed setup had three entities - a malicious client,

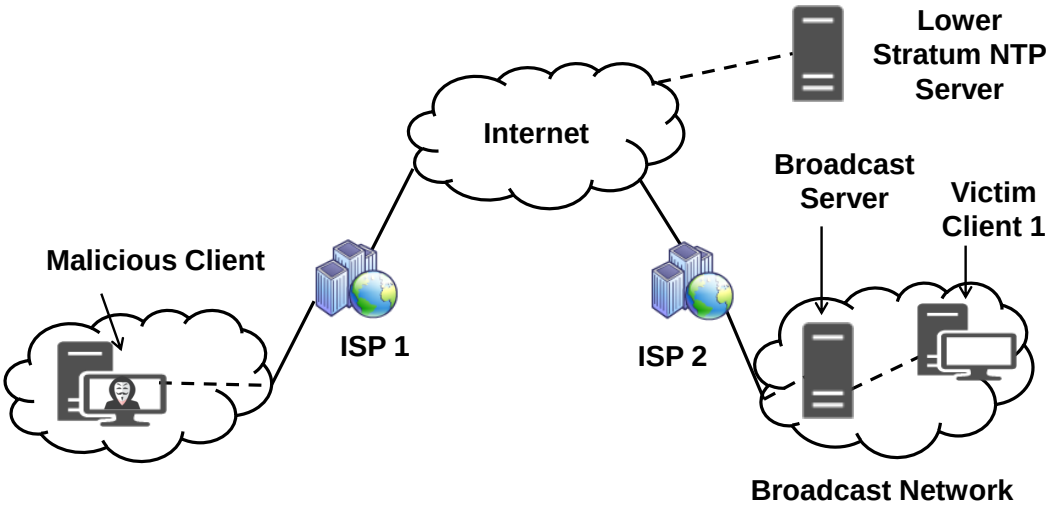



Figure 7.15: Testbed setup

victim client and broadcast server. The broadcast server and victim client were part of same broadcast network while the malicious client was connected to another network. All the entities in the testbed were configured on computers running Kali Linux 2.0 operating system and having Core i7 processor with 4 GB of physical memory. Here are the details about various entities and how they were configured:

**1. Broadcast Server:** We installed *ntpd* v4.2.8p10 on the broadcast server and configured it by adding *broadcast* keyword and network broadcast address in NTP configuration file. A symmetric key was also configured in the configuration file for authentication purpose. This server was synchronized to an external stratum 2 NTP server on the Internet using one of its network interface while another interface was used to send mode 5 NTP packets on the internal network.

**2. Victim Client:** We also installed *ntpd* v4.2.8p10 on the victim client and configured it by adding *broadcastclient* keyword and the required symmetric key in its NTP configuration file.

**3. Malicious Client:** The malicious client was equipped with two python scripts to

replay mode 5 and mode 3 NTP packets at regular intervals. In particular, we used scapy python library to replay these packets. The first python script was configured to replay copies of the mode 5 packet at a rate of 1 packet per second while the second python script was configured to first replay two copies of mode 3 packet to broadcast server and then at a rate of only 1 packet per 10 seconds.

Before the attack execution started, we let the broadcast server and the victim client to synchronize to external stratum 2 server on the Internet and the broadcast server respectively. In the meantime, from the communication between broadcast server and victim client, we captured one packet each of type mode 5 and mode 3. These packets were then provided to the designated malicious client so that the packets could be replayed by the malicious client in the future.

**Attack Execution:** We executed the attack for $\approx$ 2.5 hours on the testbed setup. Figure 7.16 shows the relative timing of various events as seen in the

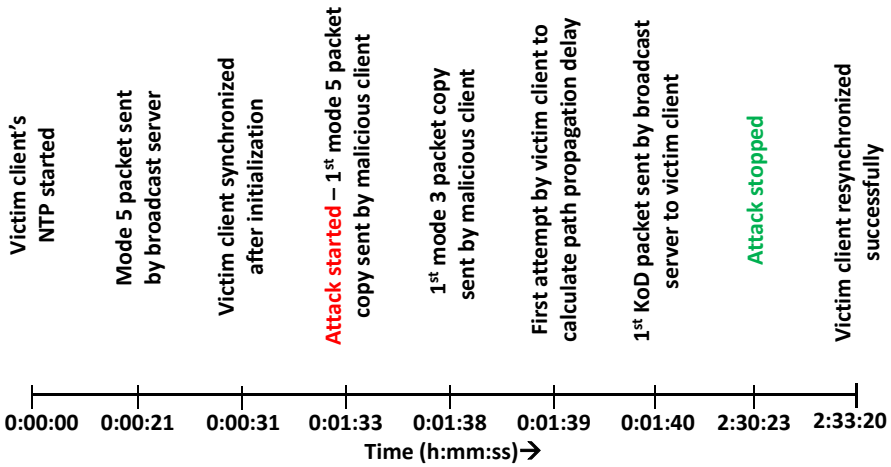



Figure 7.16: Timing of various events

experiment. As it can be seen from the figure, we started the attack only after the victim client was synchronized with the broadcast server. We can also notice that 177 seconds after the attack was stopped, victim client could successfully resynchronize itself with the broadcast server. The total number of mode 5 and mode 3 NTP packets replayed by malicious client while executing the attack were

8388 and 887 respectively. The total number of mode 5 and KoD packets sent by broadcast server were 138 and 886 respectively while the total number of mode 3 queries sent by victim client was 125. Meanwhile, victim client made 30 attempts to calculate the path propagation delay in order to synchronize itself with the broadcast server, however, it failed each time. Eventually, when the attack stopped, victim client could successfully synchronize itself with the broadcast server in its 31st attempt.

**Attack's Effectiveness in NTP's Authenticated Multicast Mode:** RFC 5905 [10] allows NTP clients and server to operate in multicast mode as well. Our proposed attack can also prevent a client configured in multicast mode from synchronizing itself with a multicast NTP server. To test the attack in multicast NTP mode, we configured victim client to listen mode 5 packets by adding *multicastclient* keyword and IPv4 multicast group address 224.0.1.1 in its NTP's configuration file. We also modified NTP broadcast server's configuration file by replacing the network broadcast address with 224.0.1.1 to send mode 5 NTP packets on this address. NTP daemon of broadcast server and victim client were then restarted to apply the changes. We then let the broadcast server and victim client synchronize to external stratum 2 server on the Internet and broadcast server respectively. To launch the attack, malicious client simply replayed mode 5 and mode 3 NTP packets as discussed earlier. On receiving mode 5 packets, victim client made several attempts to recalculate the path propagation delay, however, mode 3 packets replayed by malicious client to broadcast server prevented victim client to calculate the delay each time. Thus, the proposed attack was found to be effective in NTP's authenticated multicast mode also.

### 7.6.2 Experiments with the Proposed Detection Approach

We used the same testbed setup shown in Figure 7.15 to test the proposed detection approach. The broadcast domain and the network to which malicious client belonged were provided Internet connectivity using different Internet Service Providers (ISPs). The choice of selecting different ISPs was deliberate as we use TTL value to differ-

entiate spoofed and genuine addresses and it is likely that IP addresses from same ISPs may not traverse many hops. Since the broadcast server and victim client were part of the same network, there were no hops between them. However, our designated malicious client was part of different network such that the number of hops between malicious client and either of the broadcast server or victim client was 3. For our experiments, we implemented active verification and probing method on the broadcast server only. This was because detecting spoofed mode 3 packets sent to the broadcast server suffices for detecting the attack. Moreover, deploying the proposed detection technique only at broadcast server is easier as compared to its deployment on large number of potential NTP clients in the broadcast domain which can be very cumbersome. Apart from this, deploying the detection technique on clients puts significant traffic overhead on the network due to generation of large number of probes. Thus, we implemented the detection technique on the broadcast server only using a python program having 2 threads. First thread acted as packet sniffer and collected NTP packets and extracted the IP address and the TTL value from each of these packets. The thread further passed these two values to the second thread that acted as a probing module. The second thread first calculated the number of hops the sender of an NTP packet was away from the receiver and then crafted a TCP SYN probe within an IP packet having source IP address as the IP address of the sender of the NTP packet and TTL value as the calculated number of hops between the receiver and the sender. Subsequently, the thread sent the probe and waited for $\eta = 3$ seconds[3] to receive the corresponding probe reply. On receiving the probe reply, the thread compared the TTL value present in the probe reply and the NTP packet received earlier to make a final decision (normal or spoofed) about the packet.

Using this setup, we tested all spoofing scenarios described in Section 7.5.1. The experiments performed and the results obtained in each of these scenarios is as follows:

**Normal Scenario:** In case of normal scenario, we ran the simulation for 10

[3]Its value is several fold higher than the average round trip delay(0.3537867) observed by probing 100 different IP addresses present across the world

minutes by first starting the sniffing module on the broadcast server and then restarting the *ntpd* daemon of the victim client. Figure 7.17 shows the timing of occurrence of various events on victim client and broadcast server. We can notice from the figure that first normal mode 3 packet was correctly detected as normal after 18 seconds since the simulation was started. Moreover, the last mode 3 packet was sent by the victim client after 21 seconds of starting the simulation and it was correctly detected as normal by detection approach at the broadcast server after 23 seconds of starting the simulation. After this time, there were no mode 3 packets sent by the victim client as it could successfully synchronize its clock with the broadcast server's clock.

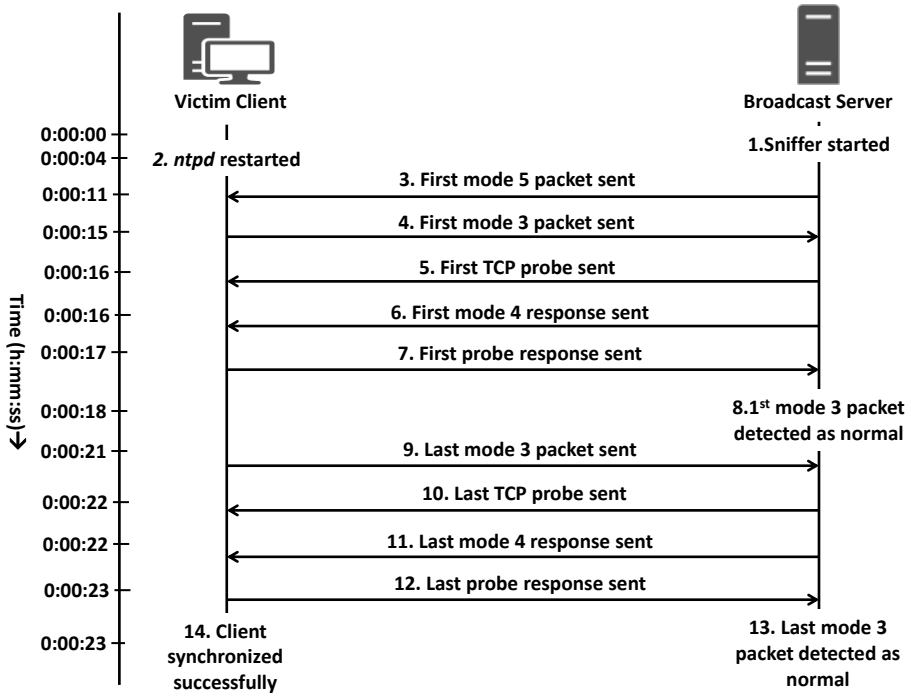



Figure 7.17: Events and their timings in normal scenario

**Spoofing Scenario-1:** In this case also, we ran the simulation for 10 minutes by first starting the sniffing module on the broadcast server and then restarting the *ntpd* daemon of the victim client. Once victim client was successfully synchronized to the broadcast server, malicious client started sending mode 3 and mode 5 packets to

broadcast server and victim client which were spoofed to look like they were coming from victim client and broadcast server respectively. Figure 7.18 shows the timing of occurrence of various events on victim client and broadcast server. For the sake of brevity, we do not show the timing of messages exchanged between victim client and broadcast server in this case. We can notice from the figure that first spoofed mode 3 packet was correctly detected as spoofed after 31 seconds since the simulation was started. Moreover, the last spoofed mode 3 packet was sent by the malicious client after 9 minutes and 57 seconds of starting the simulation to maintain the attack. This last spoofed mode 3 packet was correctly detected as spoofed by detection approach at the broadcast server after 9 minutes and 59 seconds of starting the simulation.

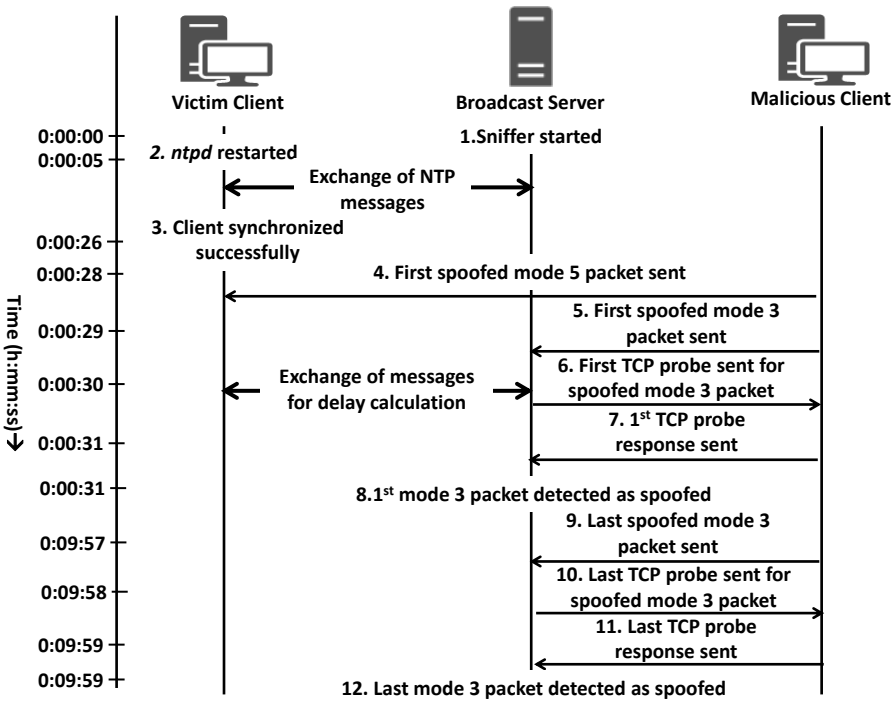



Figure 7.18: Events and their timings in spoofing scenario-1

**Spoofing Scenario-2:** In order to create second spoofing scenario, we deliberately modified */etc/sysctl.conf* of victim client to send all the packets to the broadcast server with an initial TTL value of 25. This caused server to believe that the victim client is 7 hops away (32 (predicted by server) − 25) from it. After this,

we ran the simulation for 10 minutes by first starting the sniffing module on the broadcast server and then restarting the *ntpd* daemon of the victim client. Once victim client was successfully synchronized to the broadcast server, malicious client started the attack execution. Figure 7.19 shows the timing of occurrence of various events on victim client and broadcast server. We can notice from the figure that the broadcast server received first ICMP error message (as a response to the first probe sent) after 30 seconds of starting the simulation. Moreover, the broadcast server received the last ICMP error packet after 9 minutes and 58 seconds. Thus, the last spoofed mode 3 packet was correctly detected as spoofed by detection approach after 9 minutes and 58 seconds of starting the simulation.

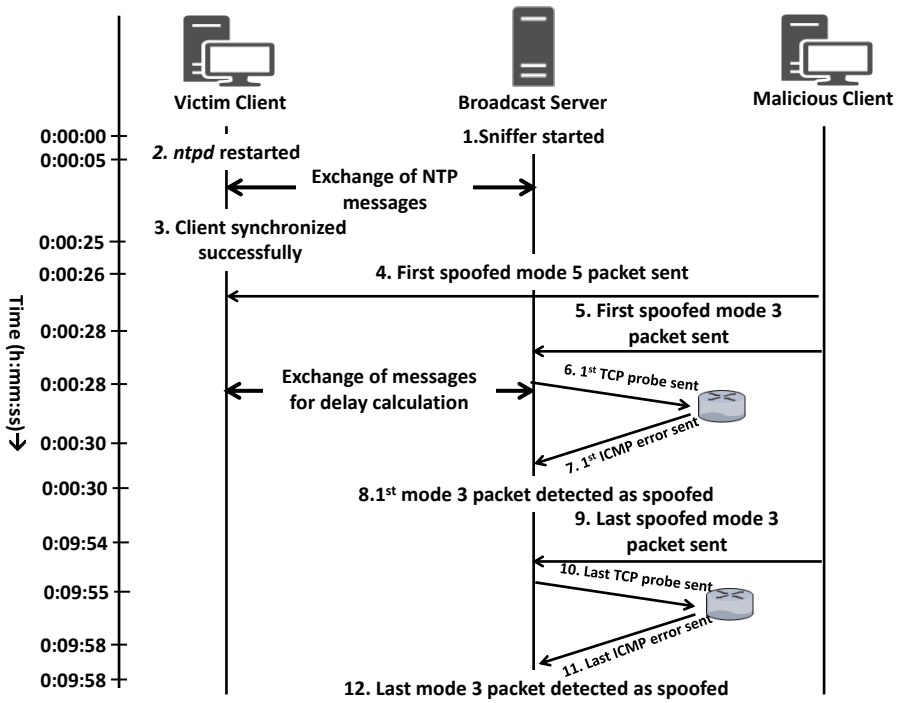



Figure 7.19: Events and their timings in spoofing scenario-2

Table 7.4 shows experiment details such as simulation duration, number of normal mode 3 packets sent by victim client and normal of spoofed mode 3 packets sent by the malicious client to the broadcast server in each of the three scenarios. Table also shows the results obtained in case of different scenarios. We can notice that all the packets were correctly detected as normal or spoofed ones and hence the proposed

attack in all the scenarios .

Table 7.4: Experiment results

| **Scenario** | **Duration of Simulation** | **Normal Packets** | **Spoofed Packets** | **Result** |
|---|---|---|---|---|
| Normal | 10 | 4 | 0 | All 4 normal mode 3 packets were detected as normal ones |
| Spoofing Scenario-1 | 10 | 17 | 53 | All 17 normal and 53 spoofed mode 3 packets were detected as normal and spoofed ones respectively |
| Spoofing Scenario-2 | 10 | 15 | 53 | All 15 normal and 53 spoofed mode 3 packets were detected as normal and spoofed ones respectively |

## 7.7 Conclusion

NTP is used to synchronize clocks of computer systems on the Internet. Inconsistencies in clocks of computer systems of even fractions of a second can cause failure of various core services of the Internet. In this chapter, we proposed a new attack that can prevent a client from synchronizing to a NTP broadcast server. We tested the proposed attack in real network and reported the results. The proposed attack was found to be effective in both authenticated and unauthenticated broadcast and multicast modes. We also discussed how the proposed attack can render all the broadcast clients in the network unsynchronized.

Since the proposed attack against NTP's broadcast mode involves sending few spoofed NTP packets to both broadcast server and victim client, we also proposed an active verification technique to detect spoofed NTP packets. The detection of spoofed NTP packets was done by verifying the source IP address of the packet using a probe

request and then comparing the TTL value of the probe response and the NTP packet whose authenticity was being verified. We tested the performance of this detection approach in real network and showed that it could detect the spoofed packets without any false positives.

# Chapter 8

# Conclusion

In this chapter, we summarize the various aspects of our work towards different application layer DoS attacks and their detection. We chose DHCP, HTTP/1.1, HTTP/2 and NTP to perform their vulnerability assessment as these protocols are quite popular and are widely used in today's networks. The goal of our research was to find potential vulnerabilities in these four application layer protocols which can be exploited to launch DoS attacks and propose appropriate detection mechanism to detect the resulting DoS attacks.

We first motivated our thesis by stating the importance of countering DoS attacks to prevent the disruption of network services. We provided background on DoS attacks and discussed some of the popular DoS/DDoS attacks. We pointed out that the application layer DoS attacks are stealthier and easier to launch as compared to other types of DoS/DDoS attacks. We also presented some recent trends which show that the number of incidences where application layer DoS attacks are encountered is continuously increasing. The goal of this thesis research was therefore to find novel application layer DoS attacks against different protocols and propose appropriate detection techniques that can be deployed as as a first line of defense to detect these attacks. We first proposed Induced DHCP starvation attack that can prevent clients from acquiring an IP address from DHCP server. We also proposed three detection approaches to detect the proposed Induced DHCP starvation attack. Subsequently, we presented an empirical study of Slow HTTP DoS attacks against different web

servers and real web sites and also proposed a statistical abnormality measurement technique to detect Slow HTTP DoS attacks. Thereafter, we proposed new Slow Rate DoS attacks against HTTP/2 and a chi-square test based detection technique to detect the attacks. As our last contribution, we proposed an attack against NTP's broadcast mode that can prevent clients from synchronizing their clocks with a broadcast server and a detection technique that can detect spoofed NTP packets.

## 8.1 Thesis Contributions

We recap the thesis contributions in this subsection.

### 8.1.1 Induced DHCP Starvation Attack

We proposed a new attack against DHCP called as Induced DHCP starvation attack which can prevent a client from ever obtaining an IP address. We tested the proposed attack in both wired and wireless networks and showed that it is effective in both the network topologies. We also compared the proposed attack with the classical DHCP starvation attack and showed that the proposed attack is easier to launch and requires fewer number of packets to be sent from malicious client. We also described how Induced DHCP starvation attack can be used to launch targeted DNS spoofing attack within a local network wherein an adversary can send forged DNS replies in response to a particular DNS query sent by a particular client within the network.

### 8.1.2 Detecting Induced DHCP Starvation Attack

In this contribution, we first describe how previously known defense mechanisms and security features in network switches fail to detect different variants of Induced DHCP starvation attack. Thereafter, we proposed three different approaches to detect different variants of Induced DHCP starvation attack. The first detection method was a statistical abnormality measurement technique that compares normal DHCP traffic profile generated during training phase with the profiles generated in different time

intervals of the testing phase using Hellinger Distance. The second detection approach simply compares the number of DHCP messages of a particular type received in a particular time interval of a day of training phase with the number of DHCP messages of the same type received in the same interval of the testing phase. If the difference in number of received messages exceeds a predefined threshold, Induced DHCP starvation attack is detected in that time interval. In our last detection approach, we compared the detection performance of different one-class classification algorithms to detect different variants of the Induced DHCP starvation attack. We used DHCP traffic collected from one of our institutional network to test the detection performance of these approaches and furnished the obtained results. We also performed several experiments to test the effect of varying attack rate, time duration of each interval and predefined threshold on the detection performance of proposed approaches and presented the results in the form of sensitivity analysis.

### 8.1.3 Slow HTTP DoS Attacks against HTTP/1.1 and Detection

We considered two popular Slow HTTP DoS attacks- *Slow Header* and *Slow Message Body* attack and first performed an empirical study of these attacks against different web servers and live websites. Subsequently, we also proposed an anomaly detection scheme to detect these attacks. Similar to our first approach to detect Induced DHCP starvation attack, this detection scheme also uses Hellinger Distance to compare normal HTTP traffic profile generated during training phase with the profiles generated during different time intervals of the testing phase. We used both simulated HTTP traffic collected from the web server configured in our testbed and real HTTP traffic collected from our institutional web server to test the detection performance of proposed scheme. We showed that the proposed scheme could detect the Slow HTTP DoS attacks with good detection rates. We also described the experiments performed to test the effect of varying attack rate, time duration of each interval and predefined threshold on the detection performance of proposed approach and presented the

results in the form of sensitivity analysis.

### 8.1.4 Slow Rate DoS Attacks against HTTP/2 and Detection

We proposed five new Slow Rate DoS attacks against HTTP/2 protocol. These attacks require injecting specially crafted HTTP/2 requests in order to deplete all the free connection slots available in web server's connection pool. We tested the proposed attacks against four popular web servers and showed that all of them are vulnerable to at least one of the attacks. We also compared both HTTP/1.1 and HTTP/2 protocols on the basis of similarity of threat vectors and showed that HTTP/2 has relatively more number of threat vectors as compared to its predecessor. To detect the proposed attacks, we also proposed an anomaly detection scheme that uses chi-square test to identify any deviation in HTTP/2 traffic patterns generated during testing phase from the patterns generated during training phase.

### 8.1.5 Attack against NTP's Broadcast Mode and Detection

As our last contribution, we proposed a new attack that exploits a vulnerability present in NTP broadcast mode to prevent a client from synchronizing its clock with a server. We tested the attack in a real network and showed that the two most recent versions of NTP's reference implementation *ntpd* is vulnerable to the attack. We also described how the attack can prevent any number of clients in a broadcast domain from synchronizing their clock with the broadcast server. We tested the attack against NTP's multicast mode also and showed that it is effective in multicast mode also. Subsequently, we also proposed a technique to detect spoofed NTP packets and hence the proposed attack. This technique verifies the authenticity of an NTP packet by proactively probing the source of the packet and then matching the Time-To-Live (TTL) value present in IP header of received probe reply and the NTP packet under consideration. If inconsistency in TTL value is found, the packet is considered as spoofed, otherwise it is considered as coming from genuine client. We tested the detection technique in a real network and showed that it could detect all the spoofed

NTP packets sent by malicious client while attack execution.

## 8.2 Future Work

There are several interesting and important future directions:

**Slow Rate DoS Attacks against Other Protocols:** We performed an empirical study of Slow Rate DoS attacks against HTTP/1.1 and proposed new Slow Rate DoS attacks against its successor HTTP/2. These attacks exploit the fact that these protocols require requests to be completely received before they are processed by the server. Unless these requests are completely received, they are kept in a connection queue maintained at the server. If a large number of such incomplete requests are sent, the connection queue space gets exhausted due to which further requests, potentially legitimate, are not processed by the server which leads to DoS. There are other protocols such as File Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP) which also require receiving complete requests before processing them. These protocols are very popular and there are plethora of implementations of these protocols in the wild. Depending on the protocol, there can be several possible ways in which incomplete requests can be crafted. Thus, as a future work, different application layer protocols should be explored to find out the effect of Slow Rate DoS attacks on them. Moreover, different ways in which incomplete requests can be crafted and sent to different implementations of these protocols should also be explored. Moreover, an empirical study of thus newly investigated attacks should be performed by considering different implementations of these application layer protocols.

**Vulnerability Assessment of NTP:** As discussed in Chapter 7, the working of the reference implementation *ntpd* of NTP differs vastly from its reference specification (RFC 5905). This creates surface for a large number of vulnerabilities which are present due to either flaws in NTP specification guidelines or implementation

bugs in NTP software implementations. Moreover, we described in Chapter 7 that different modes of NTP operation follow the same code path. Due to this, patching a vulnerability in one mode of NTP operation lead to creation of vulnerability in another mode. For example, our attack (CVE-2018-8956) proposed in Chapter 6 exploits a vulnerability that is created due to patching the Small-step-big-step attack (CVE-2015-5300) [14]. Thus, we believe that a detailed vulnerability assessment of *ntpd* and other implementations of NTP should be done so as to find more potential vulnerabilities in the protocol that can be exploited to prevent clock synchronization. Moreover, since changing specification requires deliberations and discussions between various stakeholders which takes a lot of time, necessary modifications should be done in *ntpd* implementation to follow the RFC 5905 as close as possible in order to minimize the chance of existence of new vulnerabilities. In Chapter 7, we also described that unsynchronized clocks can affect various core Internet services such as DNS resolution, interdomain routing using RPKI and Transport Layer Security (TLS). We believe that there should be works that can be carried on to find which and how other services can be affected due to various NTP attacks that prevent clock synchronization.

**Exploring Other Application Layer Protocols:** We took into account four application layer protocols and found potential vulnerabilities in them which can be exploited to launch DoS attacks. However, there are several application layer protocols with each protocol having several implementations developed by different vendors. Though these vendors closely follow the protocol specifications while developing the respective implementations, they use their own methods to implement some of the functionalities or modules of the protocol. Thus, it is possible that different set of vulnerabilities exist in each of the implementations. Thus, we suggest that a careful and comprehensive vulnerability assessment of application layer protocols and their implementations should be performed so as to find vulnerabilities in them and timely provide appropriate defense mechanisms before the vulnerabilities are known to various underground communities.

# Bibliography


[1] Global DDoS Threat Landscape Quarter 4. https://www.incapsula.com/ddos-report/ddos-report-q4-2017.html. Last Accessed: December 2018.

[2] E. Adi, Z. Baig, C. P. Lam, and P. Hingston. Low-Rate Denial-of-Service Attacks against HTTP/2 Services. In *Proceedings of 5th International Conference on IT Convergence and Security (ICITCS'15)*, pages 1–5, 2015.

[3] E. Adi, Z. A. Baig, P. Hingston, and C. Lam. Distributed Denial-of-Service Attacks against HTTP/2 Services. *Cluster Computing*, 19(1):79–86, 2016.

[4] M. S. Olivier. Database Privacy: Balancing Confidentiality, Integrity and Availability. *ACM SIGKDD Explorations Newsletter*, 4(2):20–27, 2002.

[5] S. T. Zargar, J. Joshi, and D. Tipper. A Survey of Defense Mechanisms against Distributed Denial of Service (DDoS) Flooding Attacks. *IEEE Communications Surveys & Tutorials*, 15(4):2046–2069, 2013.

[6] J. Mirkovic and P. Reiher. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[7] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of Network-based Defense Mechanisms Countering the DoS and DDoS Problems. *ACM Computing Surveys*, 39(1):1–42, 2007.

[8] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir. Taming the 800 Pound Gorilla: The Rise and Decline of NTP DDoS Attacks. In

*Proceedings of the Internet Measurement Conference (IMC'14)*, pages 435–448, 2014.

[9] P. Mockapetris. (RFC 1035) Domain Names - Implementations and Specifications, 1987.

[10] D. Mills, J. Martin, J. Burbank, and W.Kasch. (RFC 5905) Network Time Protocol Version 4: Protocol and Algorithms Specification, 2010.

[11] S. Ranjan, R. Swaminathan, M. Uysal, A. Nucci, and E. Knightly. DDoS-shield: DDoS-resilient Scheduling to Counter Application Layer Attacks. *IEEE/ACM Transaction on Networking*, 17(1):26–39, 2009.

[12] S. Ranjan, R. Swaminathan, M. Uysal, and E. Knightly. DDoS-resilient Scheduling to Counter Application Layer Attacks under Imperfect Detection. In *Proceedings of the 25th International Conference on Computer Communications (INFOCOM'06)*, pages 1–13, 2006.

[13] A. Malhotra and S. Goldberg. Attacking NTP's Authenticated Broadcast Mode. *ACM SIGCOMM Computer Communication Review*, 46(2):12–17, 2016.

[14] A. Malhotra, I. E. Cohen, E. Brakke, and S. Goldberg. Attacking the Network Time Protocol. In *Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS'16)*, pages 1–15, 2016.

[15] R. Droms. (RFC 2131) Dynamic Host Configuration Protocol, 1997.

[16] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. (RFC 2616) Hypertext Transfer Protocol – HTTP/1.1, 1999.

[17] M. Belshe, R. Peon, and M. Thomson. (RFC 7540) Hypertext Transfer Protocol Version 2 (HTTP/2), 2015.

[18] X. Cao. *Model Selection Based on Expected Squared Hellinger Distance*. PhD thesis, Colorado State University, Fort Collins, USA, 2007.

[19] M. Mongelli, M. Aiello, E. Cambiaso, and G. Papaleo. Detection of DoS Attacks through Fourier Transform and Mutual Information. In *Proceedings of 21st International Conference on Communications (ICC'15)*, pages 7204–7209, 2015.

[20] Is It Statistically Significant? The Chi-square Test. http://www.ox.ac.uk/media/global/wwwoxacuk/localsites/uasconference/presentations/P8_Is_it_statistically_significant.pdf. Last Accessed: December 2018.

[21] H. Mukhtar, K. Salah, and Y. Iraqi. Mitigation of DHCP Starvation Attack. *Computers and Electrical Engineering*, 38(5):1115–1128, 2012.

[22] M. Patrick. (RFC 3046) DHCP Relay Agent Information Option, 2001.

[23] Port Security. http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/port_sec.html. Last Accessed: December 2018.

[24] DHCP Snooping. http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/snoodhcp.html. Last Accessed: December 2018.

[25] Dynamic ARP Inspection. http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst6500/ios/12-2SX/configuration/guide/book/dynarp.html. Last Accessed: December 2018.

[26] D. C. Plummer. (RFC 826) An Ethernet Address Resolution Protocol, 1982.

[27] B. Issac. Secure ARP and Secure DHCP Protocols to Mitigate Security Attacks. *International Journal of Network Security*, 8(2):107–118, 2009.

[28] R. Droms and W. Arbaugh. (RFC3118) Authentication for DHCP messages, 2001.

[29] Y. I. Jerschow, C. Lochert, B. Scheuermann, and M. Mauve. CLL: A Cryptographic Link Layer for Local Area Networks. In *Proceedings of 6th International*

*Conference on Security and Cryptography for Networks (SCN'08)*, pages 21–38, 2008.

[30] J. Demerjian and A. Serhrouchni. DHCP Authentication using Certificates. In *Proceedings of 18th Security and Protection in Information Processing Systems (IFIP'04)*, pages 456–472. 2004.

[31] K. de Graaf, J. Liddy, P. Raison, J.C. Scano, and S. Wadhwa. Dynamic Host Configuration Protocol (DHCP) Authentication using Challenge Handshake Authentication Protocol (CHAP) Challenge, 2013. US Patent 8,555,347.

[32] H. Ju and J. Han. DHCP Message Authentication with an Effective Key Management. *World Academy of Science, Engineering and Technology*, 8(1):570–574, 2007.

[33] S. Shen, X. Lee, Z. Sun, and S. Jiang. Enhance IPv6 Dynamic Host Configuration with Cryptographically Generated Addresses. In *Proceedings of 5th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS'11)*, pages 487–490, 2011.

[34] Z. Su, H. Ma, X. Zhang, and B. Zhang. Secure DHCPv6 that uses RSA authentication integrated with Self-Certified Address. In *Proceedings of 3rd International Workshop on Cyberspace Safety and Security (CSS'11)*, pages 39–44, 2011.

[35] S. Duangphasuk, S. Kungpisdan, and S. Hankla. Design and Implementation of Improved Security Protocols for DHCP using Digital Certificates. In *Proceedings of 10th International Conference on Networks (ICN'11)*, pages 287–292, 2011.

[36] S. Majumdar, D. Kulkarni, and C. V. Ravishankar. DHCP Origin Traceback. In *Proceedings of 12th International Conference on Distributed Computing and Networking (ICDCN'11)*, pages 394–406, 2011.

[37] T. Aura, M. Roe, and S.J. Murdoch. Securing Network Location Awareness with Authenticated DHCP. In *Proceedings of 3rd International Conference*

*on Security and Privacy in Communications Networks and the Workshops (SecureComm'07)*, pages 391–402, 2007.

[38] A. Malhotra, M. Van Gundy, M. Varia, H. Kennedy, J. Gardner, and S. Goldberg. The Security of NTP's Datagram Protocol. In *Proceedings of 21st International Conference on Financial Cryptography and Data Security (FC'17)*, pages 405–423, 2017.

[39] Mohammad Karami, Youngsam Park, and Damon McCoy. Stress Testing the Booters: Understanding and Undermining the Business of DDoS Services. In *Proceedings of 25th International Conference on World Wide Web (WWW'16)*, pages 1033–1043, 2016.

[40] A. Bremler-Barr and H. Levy. Spoofing Prevention Method. In *Proceedings of 24th IEEE International Conference on Computer Communications Workshops (INFOCOM'05)*, pages 536–547, 2005.

[41] Y. Shen, J. Bi, J. Wu, and Q. Liu. A Two-Level Source Address Spoofing Prevention based on Automatic Signature and Verification Mechanism. In *Proceedings of 13th Symposium on Computers and Communications (ISCC'08)*, pages 392–397, 2008.

[42] A. A. Zuquete. Improving the Functionality of SYN Cookies. In *Proceedings of 6th IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS'02)*, pages 57–77, 2002.

[43] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. *ACM SIGCOMM Computer Communication Review*, 30(4):295–306, 2000.

[44] T. Peng, C. Leckie, and K. Ramamohanarao. Adjusted Probabilistic Packet Marking for IP Traceback. In *Proceedings of 2nd International Conference on Research in Networking (Networking '02)*, pages 697–708, 2002.

[45] A. Belenky and N. Ansari. IP Traceback With Deterministic Packet Marking. *IEEE Communication Letters*, 7(4):162–165, 2003.

[46] V. Paruchuri, A Durresi, and S. Chellappan. TTL Based Packet Marking for IP Traceback. In *Proceedings of 9th Global Telecommunications Conference (GLOBECOM'08)*, pages 1–5, 2008.

[47] W. T. Strayer, C. E. Jones, F. Tchakountio, and R. R. Hain. SPIE-IPv6: Single IPv6 Packet Traceback. In *Proceedings of 29th IEEE Conference on Local Computer Networks (LCN'04)*, pages 118–125, 2004.

[48] Y. Kim, W. C. Lau, M. C. Chuah, and H. J. Chao. PacketScore: A Statistics-Based Packet Filtering Scheme against Distributed Denial-of-Service Attacks. *IEEE Transactions on Dependable and Secure Computing*, 3(1):141–155, 2006.

[49] R. Vaidyanathan, A. Ghosh, Y.H. Cheng, A. Yamada, and Y. Miyake. Method and Apparatus for Detecting Spoofed Network Traffic, 2012. US Patent 8,281,397.

[50] C. Jin, H. Wang, and K. G. Shin. Hop-Count Filtering: An Effective Defense Against Spoofed DDoS Traffic. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03)*, pages 30–41, 2003.

[51] H. Wang, C. Jin, and K. G. Shin. Defense Against Spoofed IP Traffic Using Hop-Count Filtering. *IEEE/ACM Transactions on Networking*, 15(1):40–53, 2007.

[52] U. Weinsberg, Y. Shavitt, and Y. Schwartz. Stability and Symmetry of Internet Routing. In *Proceedings of 29th IEEE International Conference on Computer Communications Workshops (INFOCOM'10)*, pages 407–408, 2010.

[53] H. Beitollahi and G. Deconinck. Analyzing well-known countermeasures against distributed denial of service attacks. *Computer Communications*, 35(11):1312 – 1332, 2012.

[54] Aleksandar Kuzmanovic and Edward W Knightly. Low-rate TCP-targeted Denial of Service Attacks and Counter Strategies. *IEEE/ACM Transactions on Networking (TON)*, 14(4):683–696, 2006.

[55] G. Macia-Fernandez, J. E. Diaz-Verdejo, and P. Garcia-Teodoro. Mathematical Model for Low-Rate DoS Attacks Against Application Servers. *IEEE Transactions on Information Forensics and Security*, 4(3):519–529, 2009.

[56] G. Maciá-Fernández, J. E. Díaz-Verdejo, P. García-Teodoro, and F. de Toro-Negro. LoRDAS: A Low-Rate DoS Attack against Application Servers. In *Proceedings of 2nd International Conference on Critical Information Infrastructures Security (CRITIS'07)*, pages 197–209. 2007.

[57] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. An Overview of IP Flow-based Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 12(3):343–356, 2010.

[58] M. Aiello, E. Cambiaso, S. Scaglione, and G. Papaleo. A Similarity based Approach for Application DoS Attacks Detection. In *Proceedings of 18th Symposium on Computers and Communications (ISCC'13)*, pages 430–435, 2013.

[59] L. C. Giralte, C. Conde, I. M. De Diego, and E. Cabello. Detecting Denial of Service by Modelling Web-Server Behaviour. *Computers & Electrical Engineering*, 39(7):2252–2262, 2013.

[60] P. E. Román, J. D. Velásquez, V. Palade, and L. C. Jain. *New Trends in Web User Behaviour Analysis*, pages 1–10. Springer Berlin Heidelberg, 2013.

[61] M. Aiello, E. Cambiaso, M. Mongelli, and G. Papaleo. An On-line Intrusion Detection Approach to Identify Low-rate DoS Attacks. In *Proceedings of 3rd International Carnahan Conference on Security Technology (ICCST'14)*, pages 1–6, 2014.

[62] S. Bhatia, G. Mohay, A. Tickle, and E. Ahmed. Parametric Differences Between a Real-world Distributed Denial-of-Service Attack and a Flash Event. In *Pro-*

*ceedings of 6th International Conference on Availability, Reliability and Security (ARES'11)*, pages 210–217, 2011.

[63] P. Farina, E. Cambiaso, G. Papaleo, and M. Aiello. Are Mobile Botnets a Possible Threat? The Case of SlowBot Net. *Computers & Security*, 58(7):268–283, 2016.

[64] Core. https://httpd.apache.org/docs/2.4/mod/core.html. Last Accessed: December 2018.

[65] mod_antiloris. https://sourceforge.net/projects/mod-antiloris/.

[66] mod_limitipconn. http://dominia.org/djao/limitipconn.html.

[67] mod_reqtimeout. https://httpd.apache.org/docs/trunk/mod/mod_reqtimeout.html.

[68] D. Moustis and P. Kotzanikolaou. Evaluating Security Controls Against Http-based DDoS Attacks. In *Proceedings of 4th International Conference on Information, Intelligence, Systems and Applications (IISA'13)*, pages 1–6, 2013.

[69] G. Mantas, N. Stakhanova, H. Gonzalez, H. H. Jazi, and A. A. Ghorbani. Application-layer Denial of Service Attacks: Taxonomy and Survey. *International Journal of Information and Computer Security*, 7(2):216–239, 2015.

[70] Arbor Networks. http://whitepapers.datacenterknowledge.com/content12127.

[71] Gobbler. http://gobbler.sourceforge.net/. Last Accessed: December 2018.

[72] DHCPIG. https://github.com/kamorin/DHCPig. Last Accessed: December 2018.

[73] X. Xing, E. Shakshuki, D. Benoit, and T. Sheltami. Security Analysis and Authentication Improvement for IEEE 802.11-i Specification. In *Proceedings of 9th Global Telecommunications Conference (GLOBECOM'08)*, pages 1–5, 2008.

[74] S. Cheshire. (RFC5227) IPv4 Address Conflict Detection, 2008.

[75] ISC DHCP. https://www.isc.org/downloads/dhcp/. Last Accessed: December 2018.

[76] H. Sengar, H. Wang, D. Wijesekera, and S. Jajodia. Detecting VoIP Floods using the Hellinger Distance. *IEEE Transactions on Parallel and Distributed Systems*, 19(6):794–805, 2008.

[77] C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.

[78] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian Network Classifiers. *Machine Learning*, 29, 1997.

[79] Y. Chien. Pattern Classification and Scene Analysis. *IEEE Transactions on Automatic Control*, 19(4):462–463, 1974.

[80] D. M. J. Tax. *One-class Classification: Concept-learning in the Absence of Counter-examples*. PhD thesis, Delft University of Technology, Stevinweg, Netherlands, 2001.

[81] G. Maciá-Fernández, J. E. Díaz-Verdejo, and P. García-Teodoro. Evaluation of A Low-Rate DoS Attack against Application Servers. *Computers & Security*, 27(7):335–354, 2008.

[82] G. Maciá-Fernández, J. E. Díaz-Verdejo, and P. García-Teodoro. Evaluation of A Low-Rate DoS Attack against Iterative Servers. *Computer Networks*, 51(4):1013–1030, 2007.

[83] G. Maciá-Fernández, R. A. Rodríguez-Gómez, and J. E. Díaz-Verdejo. Defense Techniques for Low-Rate DoS Attacks against Application Servers. *Computer Networks*, 54(15):2711–2727, 2010.

[84] A. Shevtekar and N. Ansari. A Proactive Test based Differentiation Technique to Mitigate Low Rate DoS Attacks. In *Proceedings of 16th International Conference on Computer Communications and Networks (ICCCN'07)*, pages 639–644, 2007.

[85] G. Maciá-Fernández, J. E. Díaz-Verdejo, and P. García-Teodoro. Mathematical Model for Low-Rate DoS Attacks against Application Servers. *IEEE Transactions on Information Forensics and Security*, 4(3):519–529, 2009.

[86] Y. Xiang, K. Li, and W. Zhou. Low-Rate DDoS Attacks Detection and Traceback by Using New Information Metrics. *IEEE Transactions on Information Forensics and Security*, 6(2):426–437, 2011.

[87] E. Cambiaso, G. Papaleo, G. Chiola, and M. Aiello. Slow DoS Attacks: Definition and Categorisation. *International Journal of Trust Management in Computing and Communications*, 1(3):300–319, 2013.

[88] Slowloris. https://github.com/llaera/slowloris.pl.

[89] SlowHTTPtest. http://cyborg.ztrela.com/slowhttptest.php/. Last Accessed: December 2018.

[90] Rudy. https://packetstormsecurity.com/files/97738/r-u-dead-yet-denial-of-service-tool-2.2.html.

[91] Y. Xie and S. Z. Yu. Monitoring the Application-Layer DDoS Attacks for Popular Websites. *IEEE/ACM Transactions on Networking*, 17(1):15–25, 2009.

[92] D. Atienza, Álvaro Herrero, and Emilio Corchado. Neural Analysis of HTTP Traffic for Web Attack Detection. In *Proceedings of Computational Intelligence in Security for Information Systems Conference (CISIS'15), year=*.

[93] R. Braga, E. Mota, and A. Passito. Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow. In *Proceedings of 35th IEEE Conference on Local Computer Networks (LCN'10)*, pages 408–415, 2010.

[94] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[95] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards An Operating System for Networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[96] Teuvo Kohonen. The self-organizing map. *Neurocomputing*, 21(1):1–6, 1998.

[97] A. Bhattacharyya. On A Measure of Divergence between Two Statistical Populations Defined by Their Probability Distribution. *Bulletin of the Calcutta Mathematical Society*, 35(1):99–110, 1943.

[98] Distances and Affinities Between Measures. www.stat.yale.edu/∼pollard/Books/Asymptopia/Metrics.pdf. Last Accessed: December 2018.

[99] Mahalanobis Metric. https://www.cs.princeton.edu/courses/archive/fall08/cos436/Duda/PR_Mahal/M_metric.htm. Last Accessed: December 2018.

[100] Kullback Leibler Distance (KL). http://www.csse.monash.edu.au/∼lloyd/tildeMML/KL/. Last Accessed: December 2018.

[101] Netcraft Web Survey 2015. http://news.netcraft.com/archives/2015/06/25/june-2015-web-server-survey.html. Last Accessed: December 2018.

[102] Nginx Technical Specifications. https://www.nginx.com/products/technical-specs/. Last Accessed: December 2018.

[103] Jmeter. http://jmeter.apache.org/. Last Accessed: December 2018.

[104] Web Application Firewall (WAF). https://technet.microsoft.com/en-us/library/cc940466.aspx. Last Accessed: December 2018.

[105] T. Yatagai, T. Isohara, and I. Sasase. Detection of HTTP-GET Flood Attack Based on Analysis of Page Access Behavior. In *Proceedings of 1st Pacific Rim Conference on Communications, Computers and Signal Processing (PacRim'07)*, pages 232–235, 2007.

[106] T. Berners-Lee, R. Fielding, and H. Frystyk. (RFC 1945) Hypertext Transfer Protocol – HTTP/1.0), 1996.

[107] 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. Last Accessed: December 2018.

[108] Attacking HTTP/2 Implementations. https://yahoo-security.tumblr.com. Last Accessed: December 2018.

[109] CVE-2016-1546. Denial-of-Service by Thread Starvation.

[110] R. Peon and H. Ruellan. (RFC 7541) HPACK: Header Compression for HTTP/2), 2015.

[111] Y. Dodge. *The Concise Encyclopedia of Statistics*. Springer Science & Business Media, 2008.

[112] N. Ye and Q. Chen. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International*, 17(1):105–112, 2001.

[113] N. Ye, S. M. Emran, X. Li, and Q. Chen. Statistical Process Control for Computer Intrusion Detection. In *Proceedings of 1st International Conference on DARPA Information Survivability Conference & Exposition II (DISCEX'01)*, pages 3–14, 2001.

[114] N. Ye, X. Li, Q. Chen, S. M. Emran, and M. Xu. Probabilistic Techniques for Intrusion Detection based on Computer Audit Data. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 31(4):266–274, 2001.

[115] N. Ye, S. M. Emran, Q. Chen, and S. Vilbert. Multivariate Statistical Analysis of Audit Trails for Host-based Intrusion Detection. *IEEE Transactions on Computers*, 51(7):810–820, 2002.

[116] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical Approaches to DDoS Attack Detection and Response. In *Proceedings of 3rd DARPA Information Survivability Conference & Exposition, DISCEX'03.*, pages 303–314, 2003.

[117] Netcraft: September 2016 Web Server Survey. https://news.netcraft.com/archives/2016/09/19/september-2016-web-server-survey.html. Last Accessed: December 2018.

[118] h2load: Benchmarking Tool for HTTP/2 and SPDY Server. https://nghttp2.org/documentation/h2load.1.html. Last Accessed: December 2018.

[119] T. Dierks and E. Rescorla. (RFC 5246) The Transport Layer Security (TLS) Protocol Version 1.2, 2008.

[120] OpenSSL: Cryptography and SSL/TLS Toolkit. https://www.openssl.org/source/. Last Accessed: December 2018.

[121] scapy-ssl_tls 1.2.2. https://pypi.python.org/pypi/scapy-ssl_tls/1.2.2. Last Accessed: December 2018.

[122] jNetPcap. http://jnetpcap.com/docs/javadocs/jnetpcap-1.4/index.html. Last Accessed: December 2018.

[123] C. O'Reilly, A. Gluhak, M. A. Imran, and S. Rajasegarar. Anomaly Detection in Wireless Sensor Networks in a Non-stationary Environment. *IEEE Communications Surveys & Tutorials*, 16(3):1413–1432, 2014.

[124] OWASP Zed Attack Proxy Project. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project. Last Accessed: December 2018.

[125] Burp Proxy. https://portswigger.net/burp/. Last Accessed: December 2018.

[126] mitmproxy. https://mitmproxy.org/. Last Accessed: December 2018.

[127] J. Park, K. Iwai, H. Tanaka, and T. Kurokawa. Analysis of Slow Read DoS Attack. In *Proceedings of 3rd International Symposium on Information Theory and its Applications (ISITA'14)*, pages 60–64, 2014.

[128] M. Aiello, G. Papaleo, and E. Cambiaso. SlowReq: A Weapon for Cyberwarfare Operations. Characteristics, Limits, Performance, Remediations. In *Proceedings of International Joint Conference on Advances in Intelligent Systems and Computing (AISC'14)*, pages 537–546, 2014.

[129] E. Cambiaso, G. Papaleo, and M. Aiello. Slowcomm: Design, Development and Performance Evaluation of a New Slow DoS Attack. *Journal of Information Security and Applications*, 35(1).

[130] Becoming a Time Lord - Implications of Attacking Time Sources. Shmoocon Firetalks 2013. https://www.youtube.com/watch?v=XogpQ-iA6Lw. Last Accessed: December 2018.

[131] David L. Mills. *Computer Network Time Synchronization: The Network Time Protocol on Earth and in Space, Second Edition*. CRC Press, Inc., 2nd edition, 2010.

[132] Breaking SSL Using Time Synchronisation Attacks. DEFCON'23. https://www.youtube.com/watch?v=hkw9tFnJk8k. Last Accessed: December 2018.

[133] NTP Issues Today. Outages Mailing List. https://mailman.nanog.org/pipermail/nanog/2012-November/053449.html. Last Accessed: December 2018.

[134] Did Your Active Directory Domain Time Just Jump to the Year 2000? Microsoft Server & Tools Blogs. https://blogs.technet.microsoft.com/askpfeplat/2012/11/19/did-your-active-directory-domain-time-just-jump-to-the-year-2000/. Last Accessed: December 2018.

[135] ntpd. http://www.ntp.org/. Last Accessed: December 2018.

[136] D. Mills. (RFC 1059) Network Time Protocol (Version 1) Specfication and Implementation, 1988.

[137] D. L. Mills. (RFC 1119) Network Time Protocol (Version 2) Specfication and Implementation, 1989.

[138] D. L. Mills. (RFC 1305) Network Time Protocol (Version 3) Specfication, Implementation and Analysis, 1992.

[139] T.Mizrahi. (RFC 7384) Security Requirements of Time Protocols in Packet Switched Networks, 2014.

[140] D. Sibold, S. Roettger, and K. Teichel. draft-ietf-ntp-network-time-security-10: Network Time Security, 2014.

[141] C. Labovitz, A. Ahuja, and F. Jahanian. Experimental study of Internet stability and backbone failures. In *Proceedings of 29th International Symposium on Fault-Tolerant Computing (FTCS'99)*, pages 278–285, 1999.

[142] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang. BGP Routing Stability of Popular Destinations. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment (IMC'02)*, pages 197–202, 2002.

[143] M. Kovik, M. Kajan, and M. dnk. Detecting IP Spoofing by Modelling History of IP Address Entry Points. In *Proceedings of the 7th International Conference on Autonomous Infrastructure, Management and Security (AIMS'13)*, pages 73–83, 2013.

[144] V. Paxson. End-to-End Routing Behavior in the Internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, 1997.

[145] P. Pazzani. Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier. *Machine Learning*, 29:103–130, 1997.

[146] P. Datta. *Characteristic Concept Representations*. PhD thesis, University of California at Irvine, Irvine, CA, USA, 1997.

# Appendix A

# Hellinger Distance

The Hellinger Distance is used to quantify the similarity between two probability distributions $P$ and $Q$. These probability distributions are $N$ dimensional vectors with each component $P_i \in P$ of the vector representing the probability of an attribute of the vector $P$. The Hellinger Distance $d_H$ between $P$ and $Q$ is given by Equation A.1

$$d_H = (\frac{1}{2}\Sigma_{i=1}^{N}(\sqrt{P_i} - \sqrt{Q_i})^2)^{\frac{1}{2}} \tag{A.1}$$

where $N$ represents the number of attributes present in the probability distributions and $P_i$ and $Q_i$ denote the $i^{th}$ components of vectors $P$ and $Q$.

Similar to Hellinger Distance, there are some other popular distance metrics such as Bhattacharyya Distance [97], Total Variation Distance [98], Mahalanobis Distance [99] and Kullback-Leibler Divergence [100]. Bhattacharyya Distance between two distributions is calculated by extracting the mean and variances of two probability distributions as in Equation A.2

$$d_B(P,Q) = \frac{1}{4}ln(\frac{1}{4}(\frac{\sigma_P^2}{\sigma_Q^2} + \frac{\sigma_Q^2}{\sigma_P^2})) + \frac{1}{4}(\frac{(\mu_P - \mu_Q)^2}{\sigma_P^2 + \sigma_Q^2}) \tag{A.2}$$

where $\mu_*$ and $\sigma_*$ are the mean and standard deviation of the corresponding distributions respectively.

The Total Variation Distance between two distributions is calculated by taking the half of the difference of corresponding vectors in $P$ and $Q$ as in A.3

$$d_T(P,Q) = \frac{1}{2} \parallel P - Q \parallel \tag{A.3}$$

Mahalanobis Distance between two distributions $P$ and $Q$ is calculated by measuring the distance between a distribution $Q$ and the mean $\mu_P$ of another distribution $P$ as shown in Equation A.4.

$$d_M(P,Q) = \sqrt{(Q-\mu_P)^t C^{-1} (Q-\mu_P)} \tag{A.4}$$

where $C^{-1}$ is the co-variance matrix generated from the probability distributions.

Kullback-Leibler Divergence is the measure of logarithmic difference between two probability distributions $P$ and $Q$ where the expectation is taken using the probability distribution $P$. Kullback-Leibler Divergence is given by the Equation A.5

$$d_K(P,Q) = \sum_i (P_i \; log \; \frac{P_i}{Q_i}) \tag{A.5}$$

Some of the reasons due to which Hellinger Distance is usually preferred over other distance metrics are as follows.

- Lightweight Computation: Unlike Mahalnobis Distance measure, measuring Hellinger Distance between two probability distributions does not involve computationally expensive calculations like matrix inverse or covariance. Thus, Hellinger Distance is a lightweight alternative for IDS/IPSs to detect anomalies.

- Natural Lower and Upper Bounds: It is worth noting that the calculated Hellinger Distance always ranges between 0 and 1 with 0 representing perfect similarity and 1 representing maximum dissimilarity between $P$ and $Q$. Thus, Hellinger Distance has natural lower and upper bounds of 0 and 1 which other distance metrics do not have.

- Yielding Finite Distance Value: Unlike Kullback-Leibler Divergence which is defined only if $Q_i = 0 \Rightarrow P_i = 0$, $\forall i$, Hellinger Distance does not require any

such dependency between two probability distributions. Thus, in cases where probability of an attribute of $Q$ is zero, Kullback-Leibler Divergence gives infinite value while on the other hand, Hellinger Distance yields appropriate finite distance value ranging from 0 to 1.

# Appendix B

# One-class Classification Algorithms

One-class classification algorithms, also called as unary classification algorithms, are used to identify objects of a specific class amongst several other objects by learning from a training set containing only the objects of that class. A feature of one-class classification algorithms is that they use only sample points from the assigned class, so that a representative sampling is not strictly required for non-target classes. The task of classification involves defining such boundary around target class so as to cover as much of the target objects as possible and at the same time to minimize the chance of accepting non-target objects. Various one-class classification algorithms used in this thesis are described in next few subsections.

### B.0.1 Gaussian Density based One-Class Classifier

The most straightforward method to obtain a one-class classifier is to estimate the density of the training data and to set a threshold on this density. Out of different density based models, Gaussian density [77] is the most simple model. According to the Central Limit Theorem, when it is assumed that objects from one class originate from one prototype and are additively disturbed by a large number of small independent disturbances, then this model is correct. The probability distribution (Gaussian Distribution in this case) for a $d$-dimensional object $x$ is given by:

$$p_N(z;\mu,\Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} exp\left\{ -\frac{1}{2}(z-\mu)^T\Sigma^{-1}(z-\mu) \right\} \tag{B.1}$$

where $\mu$ is the mean and $\Sigma$ is the covariance matrix. The method is very simple and it imposes a strict unimodal and convex density model on the data. Thus, the number of free parameters in the normal model is

$$n_{free\mathcal{N}} = d + \frac{1}{2}d(d-1) \tag{B.2}$$

with $d$ parameters for the mean $\mu$ and $d(d-1)/2$ for the covariance matrix $\Sigma$. The main computational effort in this method is the inversion of the covariance matrix. In case of badly scaled data or data with singular directions, the inverse of the covariance matrix $\Sigma$ cannot be calculated and should be approximated by the pseudo-inverse $\Sigma^+ = \Sigma^T((\Sigma\Sigma^T))^{-1}$ or by applying some regularization (adding a small constant $\lambda$ to the diagonal, i.e. $\Sigma' = \Sigma + \lambda\mathcal{I}$). In the last case, the user has to supply a regularization parameter $\lambda$. This is then the only magic parameter present in the normal model. Because this model utilizes the complete covariance structure of the data, this method is insensitive to scaling of the data (assuming that a reasonable regularization parameter is chosen).

### B.0.2 Naive Bayes One-Class Classifier

Traditional Naive Bayes calculates the probability of being in a class given an example, consisting of specific values of different attributes. One calculates this by assuming the different attributes are independent, applying Bayes theorem and using the a priori probability of the different classes. According to [145] this is surprisingly accurate even when the independence assumption is broken. However, when only the positive information is available, the usual calculations cannot be performed. Datta [146] showed how the algorithm can be modified for positive data only and we follow his presentation. Naive Bayes one-class classifier [78] performs the classification based

on Baye's theorem given by Equation B.3

$$P(C_t|X) = \frac{P(X|C_t)P(C_t)}{P(X)} \tag{B.3}$$

where $C_t$ denotes the target class and a $d$-dimensional vector $(X = X_1, X_2, \ldots, X_d) \in C_t$. $P(C_t)$ is the prior probability of occurrence of class $C_t$ and $P(X|C_t)$ is the conditional probability of $X$ given class $C_t$. These feature vectors are used to estimate prior probability of $C_t$ and the conditional prior probability $P(X|C_t)$. During testing phase, for a given test vector $X$, posterior probability $P(C_t|X)$ of class $C_t$ is estimated. A threshold $\delta$ is set based on the probabilities $P(C_t|X)$ obtained during training phase. If probability of a testing sample is greater than $\delta$, the sample is considered as outlier, otherwise, the sample belongs to target class.

### B.0.3 $K$-Nearest Neighbour

In K-nearest neighbour algorithm,the probability density is calculated on the basis of number of observation vectors in a region of a certain volume. Moreover, the number of observations $K$ is fixed in advance, and the volume of the area is allowed to grow so that K nearest observation vectors would be included in it. Taking the number $K$ as an input smoothing parameter, the method estimates the density as:

$$p(x) = \frac{K}{NV_k} \tag{B.4}$$

where $V_K$ is the volume of the smallest area (hypersphere) with the centre in $x$ surrounding $K$ observation vectors nearest to $x$.

The advantage of K-nearest-neighbours is its ability to estimate arbitrary distributions. Furthermore, by using varying volume size, K-nearest-neighbours overcomes the shortcoming of the other methods like kernel-based estimation, which tends to over-smooth the estimate in the areas of high density, and tends to produce too noisy estimate in the areas of low density. The drawback of this method is the need to keep the observation vectors. Furthermore, the produced estimate is not true prob-

ability density as its integral over the $x$ space diverges. This limitation is however compensated by the ability to adjust the area volume to the density of the data.

No parameters need to be learnt by this method. The number of neighbours $K$, however, needs to be provided, and too great or too low $K$ values result in over-smoothed or noisy estimated density, respectively. One approach to the problem of selecting the optimal value of $K$ is to assign the value of K that minimizes error on a separate dataset.

### B.0.4 $K$-means Clustering

K-means clustering is the method which assumes that the data is clustered and can be described by a set of vectors $\mu_k$ where $k = 1...K$. The number $K$ of prototype vectors should be selected beforehand. During classification, the reconstruction error is calculated as:

$$\varepsilon_{reconstr} = min_k||x - \mu_k||^2 \tag{B.5}$$

and compared against a threshold.

The placement of prototype vectors is derived from the training dataset. In K-means clustering, the prototype vectors are selected to minimize error using following Equation:

$$\varepsilon_{KM} = \sum_{i=1}^{N}(min_k||x_i - \mu_k||^2) \tag{B.6}$$

where $N$ is the size of training data set.

In this algorithm, at each step, the observation vectors are grouped into $k$ disjoint sets $S_k$ according to the nearest prototype vectors. Then, the prototype vectors are recalculated as:

$$\mu_k = \frac{1}{N}\sum_{i \in S_k}(x_i) \tag{B.7}$$

where $N_k = |S_k|$. The procedure is repeated until convergence.

### B.0.5 Principal Component Analysis

Principal Component Analysis (PCA) [77] is used if data is distributed in a linear subspace. PCA mapping is aimed to find out the orthogonal subspace which captures the data variance as best as possible. There are various neural network approaches which exist for optimization of PCA. The simplest procedure is to use eigenvalue decomposition for the estimation of eigenvectors of the target covariance matrix. The eigenvectors with the largest eigenvalues are the principal axis of the $d$-dimensional data and point in the direction of the largest variance. These vectors are used as basis vectors for the mapped data. The number of basis vectors $M$ is optimized to explain a certain, user defined, fraction of the variance in the data. The basis vectors $\boldsymbol{W}$ become a $d \times M$ matrix. Since they form an orthogonal basis, number of free parameters in PCA is given by Equation B.8

$$n_{free_{PCA}} = \begin{pmatrix} d-1 \\ M \end{pmatrix} \tag{B.8}$$

In PCA it is often assumed that the data has zero mean. When the mean of the data has to be estimated also, this will add another $d$ free parameters to $n_{freePCA}$. The reconstruction error of an vector $z$ is now defined as the squared distance from the training samples and its mapped version:

$$d_{PCA}(z) = \left\| z - (W(W^T W)^{-1} W^T) z \right\|^2 = \left\| z - (W W^T) z \right\|^2 \tag{B.9}$$

where the second step is possible because the basis $W$ is orthonormal. The PCA performs well when a clear linear subspace is present. Also for very low sample sizes the data is automatically located in a subspace. When the intrinsic dimensionality of the data is smaller than the feature size, the PCA can still generalize from the low sample size data. When the data has variance in all feature directions, it might sometimes be impossible to reduce the dimensionality without reducing the fraction of the explained variance too much. For instance, when the user requests that 90% of the variance of some 2-dimensional data should be explained, it might happen that each of the two PCA features explain about 50% of the variance. In this case,

no feature reduction can be applied and the complete feature space is described by the two features. Therefore, all data is accepted. Also when data is distributed in separate subspaces, the PCA produces an average subspace which may represent the data in each subspace very poorly. The PCA is relatively sensitive to the scaling of the features, it directly influences the feature variances. Scaling changes the order of the large variance directions and thus the PCA basis. When data directions are enhanced, this improves the PCA description, but when noise is amplified, it harms the characterization. Finally, because the PCA only focuses on the variance of the target set, the PCA is incapable of including negative examples in the training.

### B.0.6 Support Vector Data Description

Support Vector Machine (SVM) based one-class classifier, called as Support Vector Data Description (SVDD) [80], aims to obtain a spherical boundary instead of a planar one around the data in feature space. The goal is to minimize the volume of hypersphere so as not to incorporate outliers in the solution. By introducing kernels, this inflexible model becomes much more powerful, and can give excellent results when a suitable kernel is used. It is possible to optimize the method to reject a pre-defined fraction of the target data. This means that for different rejection rates, the shape of the boundary changes. Furthermore it is possible to use example outliers to improve the classification results.

Let $c$ and $R > 0$ be the center and radius of resulting hypersphere where $a$ is the linear combination of the support vectors and $R$ is the distance from center to the boundary. Just as the traditional formulation, it could be required that $d(x_i, c) < R$ where $d$ is the distance between data points $x_i$ and $c$ but to create a soft margin, again slack variables $\xi_i$ with penalty parameter $C$ are used. The minimization problem in

this case is given by Equation B.10

$$\begin{aligned} &\min_{R,\ \mathbf{a}} R^2 + C \sum_{i=1}^{n} \xi_i \\ &\text{subject to:} \\ &\|x_i - \mathbf{a}\|^2 \leq R^2 + \xi_i \text{ for all } i = 1, \ldots, n \\ &\xi_i \geq 0 \text{ for all } i = 1, \ldots, n \end{aligned} \tag{B.10}$$

Once we solve this by introducing Lagrange multipliers $\alpha_i$, we can test if a new data point $z$ belongs to the class if distance from $z$ to the center is smaller than or equal to $R$. Using the Gaussian kernel as a distance function over two data points, we get the distance as given by Equation B.11

$$\|z - \mathbf{x}\|^2 = \sum_{i=1}^{n} \alpha_i \exp\left(\frac{-\|z - x_i\|^2}{\sigma^2}\right) \geq -R^2/2 + C_R \tag{B.11}$$

# Appendix C

# Chi-Square Test

Chi-square test is a statistical hypothesis test used to determine if there is a significant deviation in the observed and expected frequencies of one or more categories. This test requires proposing two hypotheses- *null* and *alternate*. Null hypothesis states that there is no significant difference between the expected and observed frequency while alternate hypothesis states that observed frequencies are significantly deviated from the expected ones. The confidence with which we can conclude whether this difference is not created only due to chance is known as *significance level*, denoted by $\alpha$. Usually a significance level of 0.05 is considered for most of the scientific experiments. The chi-square test statistic is defined as shown in Equation C.1

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{C.1}$$

where $O_i$ and $E_i$ are the observed and expected number of cases in $i^{th}$ category respectively and $n$ is the number of categories. $\chi^2$ value is very small in case $O_i$ and $E_i$ for each category are closer to each other. Higher the difference between $O_i$ and $E_i$, higher will be the $\chi^2$ value.

**Null and Alternate Hypotheses**: The null and alternate hypotheses for a chi-square test can be stated as shown in Equation C.2

$$H_N : O_i = E_i; H_A : O_i \neq E_i \tag{C.2}$$

Table C.1: Chi-square distribution table

| Degree of Freedom \ Significance Level | 0.1 | 0.05 | 0.025 | 0.01 |
|---|---|---|---|---|
| 1 | 2.71 | 3.84 | 5.02 | 6.63 |
| 2 | 4.61 | 5.99 | 7.38 | 9.21 |
| 3 | 6.25 | 7.81 | 9.35 | 11.34 |
| 4 | 7.78 | 9.49 | 11.14 | 13.28 |
| 5 | 9.24 | 11.07 | 12.83 | 15.09 |

In case the null hypothesis is found to be true, $O_i$ and $E_i$ for each category are closer to each other. This results into smaller value of numerator in Equation C.1, thereby, causing smaller $\chi^2$ value. However, if $O_i$ is dissimilar to what has been expected in null hypothesis, $O_i - E_i$ value in Equation C.1 becomes large. In this case, we obtain a larger $\chi^2$ value and thus, alternate hypothesis is accepted. The significance level ($\alpha$) and chi-square distribution table (example shown in Table C.1), can be used to determine the threshold $\chi^2$ value. If the obtained $\chi^2$ value exceeds this threshold, we can reject null hypothesis $H_N$. However, if $\chi^2$ value is below the predefined threshold, we do not have enough evidence to reject $H_N$ and accept alternate hypothesis $H_A$.