Mitigating Network-Based Denial-of-Service Attacks with Client Puzzles

Timothy John McNevin

Thesis submitted to the faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
In
Computer Engineering

Dr. Jung-Min Park, Committee Chair
Dr. Scott F. Midkiff
Mr. Randolph C. Marchany

April 15, 2005
Blacksburg, VA

Keywords: Denial-of-Service Attacks, Distributed Denial-of-Service Attacks, Denial-of-Service countermeasures, Client puzzles.

Mitigating Network-Based Denial-of-Service Attacks with Client Puzzles

Timothy John McNevin

Dr. Jung-Min Park, Committee Chair
Bradley Department of Electrical and Computer Engineering

ABSTRACT

Over the past few years, denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks have become more of a threat than ever. These attacks are aimed at denying or degrading service for a legitimate user by any means necessary. The need to propose and research novel methods to mitigate them has become a critical research issue in network security. Recently, client puzzle protocols have received attention as a method for combating DoS and DDoS attacks. In a client puzzle protocol, the client is forced to solve a cryptographic puzzle before it can request any operation from a remote server or host. This thesis presents the framework and design of two different client puzzle protocols: Puzzle TCP and Chained Puzzles.

Puzzle TCP, or pTCP, is a modification to the Transmission Control Protocol (TCP) that supports the use of client puzzles at the transport layer and is designed to help combat various DoS attacks that target TCP. In this protocol, when a server is under attack, each client is required to solve a cryptographic puzzle before the connection can be established. This thesis presents the design and implementation of pTCP, which was embedded into the Linux kernel, and demonstrates how effective it can be at defending against specific attacks on the transport layer.

Chained Puzzles is an extension to the Internet Protocol (IP) that utilizes client puzzles to mitigate the crippling effects of a large-scale DDoS flooding attack by forcing each client to solve a cryptographic problem before allowing them to send packets into the network.

This thesis also presents the design of Chained Puzzles and verifies its effectiveness with simulation results during large-scale DDoS flooding attacks.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Park, for his guidance, support, and patience for allowing me to pursue and research such challenging topics in network security.

I would also like to thank Mr. Randy Marchany for allowing me to use his computer lab, reviewing some of my work, the letters of recommendation, and for serving on my committee.

I would also like to thank Dr. Midkiff for his support, the letters of recommendation, and for also serving on my committee.

I'd like to thank the members of the Advanced Research in Information Assurance and Security (ARIAS) Lab for their support and for their patience for working with me during the last two years. I'd also like to thank Vinit Joshi for working with OPNET to help improve some of the simulations for this thesis. Also, I'd like to thank many of you that I have shared an office with and worked with in Torgersen Hall for the last year and a half. I would also like to thank the CEL TAs that I have worked with over the past two years as well.

I also want to thank all of the people that have lived in, stopped by, or hung out in 303G during the last five years.

Last but not least, I'd like to thank my Mom, Dad, Brother Tony, all of my aunts, uncles, cousins, and to both of my grandmothers for all of their love, support, and encouragement. Thank you all.

CONTENTS

CONTENTS v
LIST OF FIGURES vii
LIST OF TABLES viii
CHAPTER 1 1
1. Introduction 1
1.1. Denial-of-Service Attacks 2
1.2. An Introduction to DoS Countermeasures 5
1.2.1 Prevention 6
1.2.2 Detection 6
1.2.3 Mitigation 7
1.2.4 Traceback 7
1.3. Motivation for Researching Mitigation Techniques 9
1.4. Contributions 9
1.5. Organization of Thesis 10
CHAPTER 2 12
2. Related Work 12
2.1. Transport Layer Attacks and Defenses 12
2.2. Network Layer Attacks and Defenses 14
2.2.1 End-Host-Based Protection Mechanisms 14
2.2.2 Network-Based Protection Mechanisms 16
2.3. Client Puzzles as a Mitigation Technique 18
2.3.1 Client Puzzles at the Transport Layer 19
2.3.2 Client Puzzles at the Application Layer 24
2.3.3 Client Puzzles at the Network Layer 25
CHAPTER 3 28
3. TCP Client Puzzles 28
3.1. Overview of pTCP 28

3.2. The Client Puzzle for pTCP 31
3.3. Implementation of pTCP 33
3.3.1 Overview of the Implementation 33
3.3.2 pTCP Implementation Details 38
3.4. Experiments with pTCP 41
3.4.1 The Puzzle Algorithm 41
3.4.2 Modulation of the Puzzle Difficulty Level 43
3.4.3 Performance of pTCP during a synflood Attack 44
3.4.4 Performance of pTCP in CPU-Exhaustion Attacks 46
3.5. Deployment Scheme for pTCP 48
3.6. Shortcomings of pTCP 49
CHAPTER 4 50
4. IP Client Puzzles 50
4.1. Technical Challenges in an IP Puzzle Scheme 50
4.2. The Client Puzzle for Chained Puzzles 51
4.3. Chained Puzzles 52
4.3.1 Overview of Chained Puzzles 52
4.3.2 The Details of Chained Puzzles 55
4.4. The Effectiveness of Chained Puzzles 59
4.5. Security Concerns of Chained Puzzles 61
4.6. Implementation Details of Chained Puzzles 64
4.7. Simulation Results 66
4.8. Deployment Scheme for Chained Puzzles 71
CHAPTER 5 72
5. Conclusions and Future Work 72
5.1. Future Work with pTCP 73
5.2. Future Work with Chained Puzzles 74
5.3. Future Work with Client Puzzles 75
5.4. Conclusions 76
REFERENCES 77
VITA…… 82

LIST OF FIGURES

Figure 1-1: A DDoS attack scenario 4
Figure 1-2: Components of a DoS Countermeasure 5
Figure 2-1: A server-generated puzzle 21
Figure 2-2: A client-generated puzzle 22
Figure 3-1: Client-Server interaction in pTCP 30
Figure 3-2: XTEA6 client puzzle 34
Figure 3-3: MD5 client puzzle 35
Figure 3-4: Puzzle request packet 36
Figure 3-5: Puzzle challenge packet 37
Figure 3-6: Pseudocode for puzzle solving algorithm 37
Figure 3-7: Puzzle solution packet 38
Figure 3-8: pTCP state transition diagram 42
Figure 3-9: Puzzle verification times for pTCP 43
Figure 3-10: pTCP connection time versus puzzle difficulty 44
Figure 3-11: Percentage of connections completed versus connection times for pTCP, syncookies, and TCP 46
Figure 3-12: Percentage of connections completed versus connection times during a CPU Exhaustion attack 48
Figure 4-1: Client puzzle for Chained Puzzles 52
Figure 4-2: Initial client-router interaction 56
Figure 4-3: The forwarding process for incoming packets at a Puzzle Router 58
Figure 4-4: NS-2 node 65
Figure 4-5: Normal Packet Survival Ratio for legitimate clients 68
Figure 4-6: Total number of legitimate packets generated 68
Figure 4-7: Normal Packet Survival Ratio for legitimate clients (Larger Network) 70
Figure 4-8: Total number of legitimate packets generated (Larger Network) 70

LIST OF TABLES

Table 3-1: Assumptions in pTCP…….28
Table 4-1: Assumptions in Chained Puzzles…..50
Table 4-2: Overview of Chained Puzzles…….54
Table 4-3: Classes of users in Chained Puzzles…… …...59

CHAPTER 1

1. Introduction

As the Internet becomes an integral part in many people's lives, the need to keep servers protected, online, and available has become increasingly important. In recent years, denial-of-service (DoS) attacks and distributed DoS (DDoS) attacks have become more sophisticated and effective at obstructing this availability. In 2000, several online companies such as eBay, Amazon.com, CNN.com, and Yahoo were all affected by a large scale DDoS attack [1]. During this attack, their websites were rendered virtually unreachable to many Internet users, resulting in severe financial losses, in addition to the many unsatisfied customers. In 2002, several root Domain Name System (DNS) servers were brought down by yet another DDoS attack [2]. This attack demonstrated that attackers were becoming more intelligent because critical systems were now being attacked. The general trend in DoS attacks implies that future attacks are likely to become much worse and more disruptive, affecting a larger number of Internet users. In addition to these highly publicized attacks, there have been countless other smaller scale DoS attacks that have targeted various companies or corporations. Regardless of their scale, DoS attacks have become a serious threat and nuisance throughout the Internet because they can directly be used to destabilize the Internet. Despite the knowledge of their existence, an effective defense solution that is both practical to implement and easy to deploy has yet to be developed. This thesis presents a discussion about the current research in this area and presents two different novel mitigation techniques that minimize the crippling effects of a network-based DoS attack.

1.1. Denial-of-Service Attacks

There are many different types of DoS attacks and the number of them only increases with the release of newer protocols and network applications. In order to gain a better understanding of the most common DoS attacks, it is best to separate these attacks into two different categories: local and remote (or network-based). These attacks can be further separated into two more subcategories that describe the overall goal of the attack: stopping critical services and exhausting system resources [3].

A local DoS attack is typically a form of malicious software that resides on the local machine that intends to disrupt the normal operation of the computer's programs, processes, or services. These attacks have the ability to stop these processes from executing and cause problems for the current user and possibly other remote users that are depending upon that computer's service. Besides stopping critical processes on a local computer, local DoS attacks can also exhaust system resources such as memory, clock cycles, disk space, and even network resources. Exhausting resources on a local system is an effective means to conduct a DoS attack because when the required system resources are not available on the local machine, new applications or data can neither be executed nor processed. In this case the attack causes more damage because it does not target a specific application or weakness; it prohibits or limits the capabilities of that system and prevents users from further and continued use of that machine. Examples of these resource exhaustion attacks are a fork bomb or an application that intentionally causes errors to fill up an error log, thus exhausting disk space [3].

Remote DoS attacks, or network-based DoS attacks, is an attack where the attacker attempts to deny, disrupt, or degrade a client's access to a network service by any means necessary via a remote computer. Network-based DoS attacks can both stop current processes and exhaust system resources. Thus, when an attack is underway, the end host or server will be virtually unavailable to other clients. Examples of a remote attack that stops services are the Land attack and the Teardrop attack [3]. Examples of network-based resource-exhaustion attacks are synflood attacks, Smurf attacks, and Distributed DoS flooding attacks [3]. With the increased usage of the Internet and broadband

Internet access, network-based resource-exhaustion attacks are becoming more common and popular because they are among the easiest to launch and one of the most effective forms of a DoS attack. Unfortunately, these attacks are becoming the most difficult to defend against.

A DDoS is similar to a DoS attack, except that it involves the use of several attacking computers. In other words, in a DoS attack there is a single stream of attack traffic, but in a DDoS attack there are multiple streams of attack traffic [4]. Sometimes a single attacking is computer (i.e. synflood, Land attack). These attacks are commonly referred to as DoS attacks. Although, when the goal of the attack is primarily to consume resources of a victim, the attack will always be more effective with the use of multiple attacking computers. In this case, these attacks are referred to as DDoS attacks. The primary goal of a DDoS attack is to overwhelm the victim server and its secondary goal is to consume bandwidth of the network surrounding the victim. A DDoS attack typically consists of an attacker (or multiple attackers), several handler computers, and many attack zombies. During the DDoS attack setup phase, the attacker(s) will relentlessly probe many computers looking for various weaknesses in their systems and then make attempts to infiltrate them and convert them to handlers or zombies. If users do not properly defend and patch their systems, an attacker has the ability gain access to the computer with full privileges, to install tools or programs on their systems so they can later be used in an attack. When an attacker gains control of the computer, it has the ability to communicate back with itself and other zombies. Recent attacks have shown that attackers are becoming more sophisticated recruiting zombies and how they are commanded following their conversion. It is conceivable that an attacker could recruit zombies over a period of several years, and then when they have accumulated a large number, begin flooding packets in the direction of the victim.

When the attack begins, the attackers notify the handlers to invoke the zombies to begin flooding useless data towards the victim. It should be noted that the traffic is not always useless data; an attacker may disguise their traffic to resemble legitimate traffic to avoid any filtering mechanism that is designed to detect attack packets. For example, for an

attack on a popular website, it may be best for an attacker to flood the server with false HTTP requests. These requests are disguised and are practically indistinguishable from the normal HTTP requests from legitimate clients. Thus, the attacker can manipulate the attack in many ways to bypass any known security mechanism that is already in place. In Figure 1-1, there is an image of a typical DDoS attack scenario. The victim is the end host or the routers that are adjacent to the end host. Depending on the intensity of the attack, the end host may become overwhelmed with incoming traffic, or the adjacent routers will become overwhelmed with traffic. Nonetheless, the victim in the attack is the server in addition to the countless number of users that are no longer able to communicate with that server due to the severe congestion. Due to the nature of the attack, this attack is commonly referred to as a *flooding attack*.

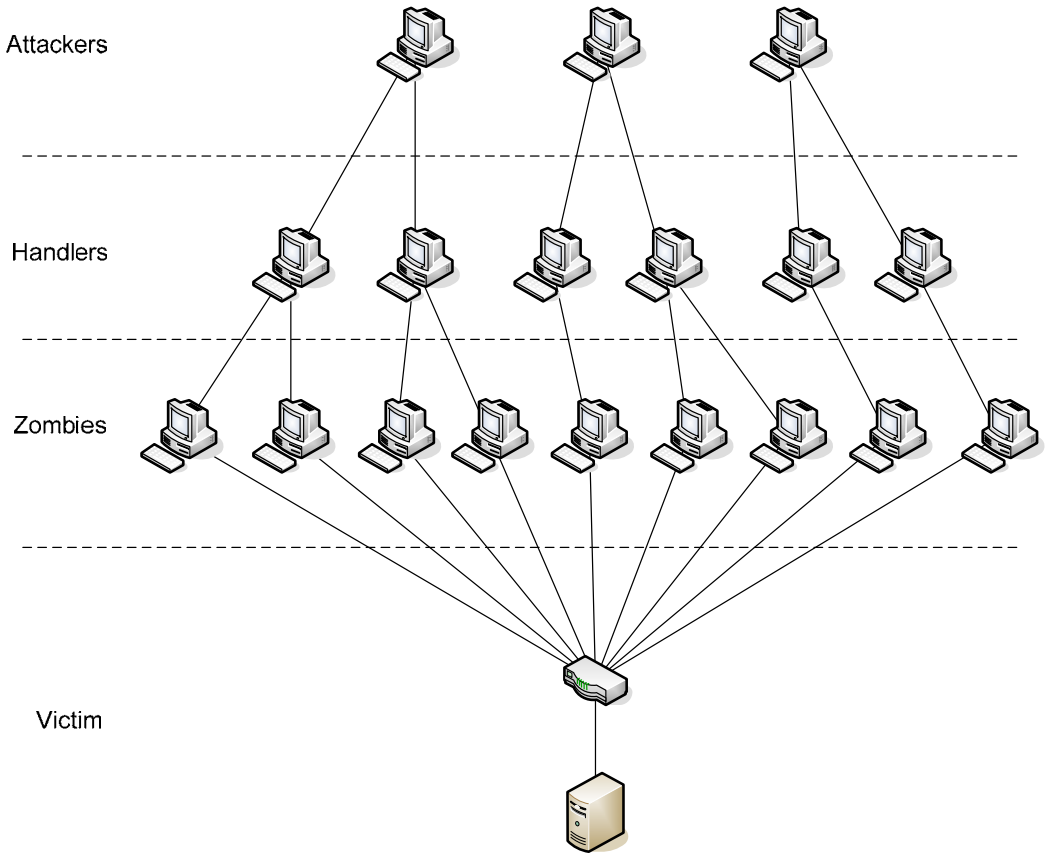

Figure 1-1: A DDoS attack scenario

1.2. An Introduction to DoS Countermeasures

A comprehensive countermeasure for DoS attacks has four distinct elements: prevention, detection, mitigation, and traceback, shown together in Figure 1-2. Before an attack occurs, there should be existing prevention mechanisms that are capable of eliminating the threat of the attack. When the attack does occur, only a successful and timely detection of the attack will allow the appropriate mitigation mechanism to be deployed. During or following the attack, a method called traceback can be used to determine the source of the attack. In addition, traceback can also help improve future methods for detecting and preventing a DoS attack. The dependence upon all four of these items is crucial for a successful DoS countermeasure.

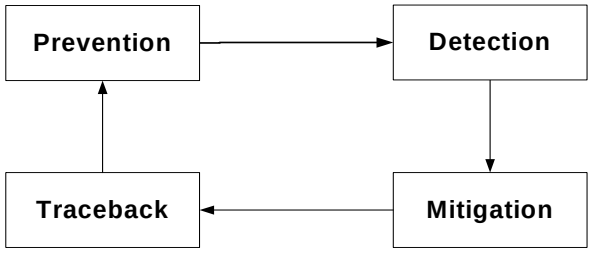

Figure 1-2: Components of a DoS Countermeasure

In addition to these four items, a DoS countermeasure should be designed to defend against attacks on various layers of the Internet stack. Among the five layers of the Internet protocol stack [5], a network-based DoS attack is associated with three of the five layers: the application, transport, and network layer. In order to design the most effective countermeasure against DoS attacks, we must address the security vulnerabilities at each layer and introduce defense mechanisms that are capable of mitigating specific threats at the appropriate layer. In computer security, there is no "magical panacea" for all potential threats. The only way to defend a computer from attacks is to design and employ a number of protection mechanisms that are designed to combat a specific threat. The combined use of all of these mechanisms will provide the greatest protection against a wide range of attacks. The most important aspect to the design of a DoS countermeasure is to prevent the countermeasure from becoming the

target of a new and unique DoS attack. All DoS attacks target some kind of vulnerability and if the vulnerability exists within the DoS countermeasure, then it needs to be redesigned.

1.2.1. Prevention

Prevention is the first step towards defending a machine from a DoS/DDoS attack. The key to prevention is awareness and vigilance. System patches and updates are very important to protection but they only fix known problems. Thus, systems are always vulnerable to unknown attacks and undiscovered weaknesses. Therefore, it is necessary for many to understand how attacks occur and to always remain aware and watching for new potential threats.

Prevention is especially important with DDoS attacks. Referring back to Figure 1-1, many of the machines used in the attack do not physically belong to the attacker. They belong to anyone who has not taken the proper steps to secure and protect their computer. Thus, informing users about the security risks and the precautions they should take with their computer is one of the key elements to prevention.

1.2.2. Detection

One of the most important components in designing a DoS countermeasure is to determine and establish the optimal methodology to detect an ongoing attack. Most detection mechanisms rely on some form of an application or software that resides on a host system or within the network that can observe traffic patterns or resource usage. These programs typically are configured to detect anomalies or deviations from normal behavior. When anomalies are detected, alerts are created so that either a system administrator or an automated program can quickly determine the type of the attack and decide which actions to take to safely minimize the effects of the attack and return the system back to its original state.

Most attacks are detected by an end host or server. However, in many cases a DDoS can affect the routers within the network. In the case of a large-scale DDoS flooding attack, detecting the attack should be done throughout the network. To combat a DDoS attack, a distributed defense is essential. In general, the further downstream the detection process is implemented, the easier it will be to determine if there is an ongoing attack [6]. The study of detecting a DoS attack is a very important area for network security research. However, it is beyond the scope of the work for this thesis. Despite its importance, we assume that throughout the remainder of this thesis that there exists a simple and basic detection mechanism that will activate the appropriate mitigation mechanism.

1.2.3. Mitigation

Mitigation is the process of minimizing the effects of an ongoing attack. The simplest and easiest form of mitigation is to simply drop packets that belong to an attacker and allow packets that belong to a client to reach their destination. This method is also by far the most effective way to mitigate the effects of a DoS attack. However, the main obstacle with this method is how to accurately determine if a packet belongs to an attacker or to a legitimate client. Attackers have shown great sophistication in their ability to disguise themselves as legitimate clients, making it virtually impossible to determine if a packet belongs to a legitimate client or an attacker. In this thesis, the mitigation techniques that are researched do not necessarily deal with making this distinction. Instead, the techniques presented in this thesis allow each client to prove itself as being legitimate before being granted access.

1.2.4. Traceback

The process of traceback involves the methods used to determine the source of the attack. This technique is commonly referred to as IP traceback. In many previous DoS/DDoS attacks, the attackers have demonstrated the ability to spoof the identity of the attack packets by selecting a different source IP address than the IP address of the computer that

is actually sending the packet. This technique is referred to as *IP spoofing*. In the past few years, there have been several attempts to prevent IP spoofing have been considered, such as ingress or egress filtering, which is performed by an Internet Service Provider (ISP) [7]. Before a router forwards a packet, it verifies that the source IP address of the packet is valid. A router inside an ISP will know the IP addresses or the IP address range of its clients and will be able to filter packets that do not have valid IP addresses. However, attackers have also been able to subvert this method by spoofing the IP address with an IP address that is valid. In other words, an attacker uses an IP address that belongs to another client on the local subnet. Thus, despite filtering at the routers, this issue still remains a difficult problem to solve.

Common traceback methods involve packet marking, a technique where routers place a unique mark within the header of each packet that it forwards. When a packet traverses all the way to the server, each router will have already marked the packet in such a way that the combination of all of the marks creates a unique signature that is used to identify a client. Thus, when the end host receives a packet, the total mark will be used to differentiate between a client and an attacker. Therefore, with an effective packet marking scheme and the assumption that packets can not be modified between the client and the server, a server can identify a client correctly without relying on the correct source IP address.

A serious challenge that is facing many researchers in IP traceback is how to determine the real attacker, rather than the attack handlers or zombies. From Figure 1-1, one can see that the real attacker hides its identity behind multiple levels of handlers and zombies. Current IP traceback methods can determine the point of origin for the attack traffic at the zombies, but they do not accurately reveal the identity of the real attacker behind the zombie. Thus, developing methods to trace the attack to the real attacker is a challenging topic in network security.

1.3. Motivation for Researching Mitigation Techniques

As recent events have indicated, DoS and DDoS attacks remain to be a severe threat to the stability of the Internet. This research topic has received much attention in the last several years because many people believe that these attacks will be a persistent threat in the Internet and could undermine the stability and usability of the Internet. Despite methods that are in existence today, the threat of an attack still lingers and future attacks will likely be more powerful and the aftermath could be considerably worse.

Successfully mitigating a DoS attack is a very challenging problem that has not yet been completely solved. The Internet itself is such a complex world and designing a perfect countermeasure to thwart these attacks has become difficult. The need to develop an effective, scalable, and resilient countermeasure has become one of the biggest challenges facing current researchers in network security.

1.4. Contributions

The main focus and contribution of this thesis is the design and analysis of a specific mitigation technique called client puzzles. Since distinguishing between an attacker and a client has been proven to be difficult, client puzzles is a technique that is used to help determine this characteristic for us. In essence, a client puzzle is a method for a client, whether it is a legitimate client or an attacker, to prove its legitimacy by solving a moderately hard computational problem that involves the utilization of both the client's system resources and time. Generally, when referring to client puzzles, the client is considered either an attacker (a zombie) or a legitimate client. The server is considered the victim that is being attacked. Client puzzles do not make a distinction between clients and attackers and consider both of them to be the same. However, all client puzzle schemes must make the same important assumption: that attackers exhibit their attacker-like behavior by making more requests or sending greater amounts of data than any legitimate client.

In a client puzzle scheme, the difficult problem presented to the client is solved by that client before it has access to a remote service or access to a remote server. Thus, before an attacker can send a large number of requests to a server, it has to solve a corresponding large number of puzzles. The exchange of puzzle information (the puzzle challenge and the puzzle solution) that is completed before a request can be made is commonly referred to as a client puzzle protocol because it describes the actions taken by both sides to complete a request. One of the main contributions in this thesis is the design and implementation of *Puzzle TCP* (pTCP): a client puzzle protocol that is embedded into the Transmission Control Protocol (TCP). As this thesis shows in later chapters, pTCP is capable of safeguarding the available resources of a server in the event of a DoS attack.

When the attack becomes larger, such as a DDoS flooding attack, the transport layer is not the appropriate location to deploy a mitigation scheme. In this attack the network layer is responsible for forwarding packets towards the destination. Thus, in order to combat such an attack, client puzzles must be deployed on the network layer. This thesis also presents a novel framework for a client puzzle protocol embedded into the network layer called *Chained Puzzles* (CP). In Chained Puzzles, any client, whether it is a zombie or legitimate client, is forced to solve a puzzle before sending a packet into the network. By doing so, this throttles every user and degrades performance for those wishing to send a large amount of data, behavior that is consistent with that of an attacker or a zombie involved in a DDoS flooding attack. A legitimate client will typically not have a relatively large number of puzzles to solve and will have slightly limited access to a server that would otherwise be unavailable.

1.5. Organization of Thesis

The remainder of this thesis is organized in the following way: Chapter 2 presents a discussion about the related work in the mitigation of DoS/DDoS attacks. Specific attacks and their respective mitigation techniques are discussed. In Chapter 3 there is a

discussion on the design and implementation details of pTCP. A performance analysis and a summary of the experimental results for pTCP are also presented in Chapter 3. In Chapter 4, the framework for a client puzzle protocol that is embedded into the network layer, called Chained Puzzles, is presented. Chapter 4 presents the design of Chained Puzzles, as well as a performance analysis and a discussion of simulation results. In Chapter 5, the work and contributions of this thesis are summarized; the future work for both pTCP and CP is presented along with improvements that could be made to each protocol.

CHAPTER 2

2. Related Work

During the last few years, there has been a sharp increase in the number of network-based computer attacks. This has lead many researchers to study this field in great depth in order to develop novel methods that are capable of eliminating this threat from today's computer networks. This chapter presents a summary of some of the most recent work on the mitigation techniques of common DoS and DDoS attacks. The work that is summarized in this chapter deals primarily with attacks on the transport layer, attacks on the network layer, and a thorough introduction to the concept of the mitigation technique known as client puzzles.

2.1. Transport Layer Attacks and Defenses

Several approaches have been proposed to prevent transport layer resource-exhaustion attacks on systems. Occasionally, these attacks are referred to as connection-depletion attacks because they consume resources on the server and prevent future clients from establishing connections or communicating further with a server. In these attacks, the victim has enough processing resources to handle each incoming packet; the attack typically targets a vulnerability that is unique to the transport layer protocol. There have been several attempts to counter these attacks such as syncache, syncookies, and client puzzles.

TCP is an end-to-end transport layer protocol that provides reliable data transmission in a connection-oriented fashion [8]. One of the most common attacks on the transport layer is the synflood attack [9, 10]. A synflood is accomplished when an attacker sends a large number of SYN packets to the victim, thus creating a large number of half-open connections that are stored on the server. The server has limited storage in terms of the number of half-open connections it can store. An attacker can effectively exhaust the server's resources by filling the queue of half-open connections, which denies service to future clients wishing to make a connection. After a timeout period, the server will remove the half-open connection from its queue, but as long as the attacker can send SYN packets at a high enough rate, the queue for half-open connections will always be full.

Syncache [11] was designed to replace the linear chain of pending and incomplete connections. Syncache implements a global hash table that protects a server from resource depletion by limiting the size of the table and the amount of time spent searching for a pending connection. Despite this modification, syncache can still suffer from connection depletion or synflood attacks because a syncache bucket, or a hash chain in the hash table, can still overflow. Lemon [11] suggests that the syncookies mechanism should be used when this occurs.

Syncookies [11, 12] focuses on defending a server solely against a synflood attack. It accomplishes this by removing state information after sending the SYN-ACK packet to the potential client. The sequence number in the SYN-ACK packet is created by applying a hash algorithm on the client's information (destination port, source port, IP address, etc.) and a secret key maintained by the server that changes every minute. The client increments this sequence number and sends an acknowledgement back to the server. When the server receives this, it decrements the number (which should yield the output of the hash function) and then reapplies the same hash function and compares the two values. If they are the same, the connection is established. The advantage to this scheme is that it requires no modification to the client's operating system. The

syncookies scheme has a potential vulnerability if an attacker actually completes the TCP handshaking procedure. Such an attack can be carried out by zombies controlled by an attacker in a DDoS attack. The zombies would execute an extremely large number of TCP handshaking procedures with the server or maintain a large number of connections to exhaust resources of the server or of a particular application.

Client puzzles have also been proposed to combat the synflood attack and other resource-exhaustion attacks that may exist on the transport and application layers. Due to client puzzles being a major theme throughout the work of this thesis, a lengthy discussion on client puzzles can be seen in Section 2.3.

2.2. Network Layer Attacks and Defenses

The resources at the server are not always the only target of an attack. In most DDoS attacks, the victim of the attack can exist in possibly two places: the server and the network. In a DDoS flooding attack, the attackers relentlessly flood the network with packets which can overwhelm either the routers in the network near the victim or the actual processing resources of the victim itself. Most current mitigation techniques involve filtering the attacker's packets or rate-limiting the traffic that is suspected to belong to an attacker. Because the effects of the attack may exist in several places, researchers have developed several methods to diminish the effects of these attacks that reside both at the server and in the network.

2.2.1. End-Host-Based Protection Mechanisms

Many approaches to defend against DoS attacks have included a filtering mechanism at the end host or server. This location is usually suitable for detecting the attack, but in the event of a flooding attack, it is not the ideal place to begin defending against the attack. However, when it comes to deployment, these schemes are among the easiest to deploy because only the end-host needs to be modified and not the rest of the network. Recently,

a scheme was introduced to filter packets based on the hop-count or the Time to Live (TTL) value located within the IP header. This scheme is called Hop-Count Filtering (HCF) [13]. Essentially, in this scheme a server first undergoes a learning phase, where it stores a database of mapping of IP addresses to TTL values. This process creates a baseline for this particular system. During an attack, if the system determines or detects anomalies by examining the IP address and the TTL value, it can drop the packet. The idea behind this scheme is that during an attack, packets will often contain spoofed IP addresses and when the server receives them it can quickly determine if the IP address does not have the corresponding observed TTL value. Since the database was created earlier, the server can drop incoming packets that do not coincide with entries in the database. This scheme suffers when zombies use their correct source IP addresses. In a large-scale DDoS attack with zombies, an attacker does not always need to spoof the IP address of the zombie. As long as the attacker can separate itself from the zombie system and cover its tracks so that they are not revealed afterwards, the zombie would be able to use its real source IP address.

Many of the current filtering mechanisms rely on the use of IP traceback, which was briefly discussed in the first chapter. Since IP traceback creates a unique signature or path identification mark, each packet can be identified and determined if it belongs to the source of an attack [14]. In these schemes, each router marks a small unique stamp into the packet. After all routers forward the packet to its destination, the packet has accumulated multiple marks from the different routers, and it forms a path identification mark or signature. During a learning phase, and when the system is not under attack, these marks are stored in a database. Following the learning phase, and during an attack, the server can identify packet anomalies by examining the mark inside each packet and decide if it needs to be discarded. This scheme, along with other packet marking schemes, requires significant changes to routers throughout the Internet for its implementation and successful deployment.

2.2.2. Network-Based Protection Mechanisms

As discussed in first chapter, the most effective way to mitigate a DoS attack is to filter packets that belong to an attacker. In recent attacks, the attack programs running on the zombies have employed IP spoofing to avoid revealing their identity. Ingress and egress filtering is a technique used to combat IP spoofing [7]. Both of the filtering mechanisms rely on routers filtering packets by examining the source IP address of each packet and then determine if the IP address was spoofed. For instance, if a packet that was leaving the subnet and had a source IP address that is not in the correct range, the router can detect this and drop the packet. Likewise, a router can drop incoming packets if the source IP address matches a client within that subnet.

The disadvantage to both ingress and egress filtering is that they rely on widespread deployment. If certain ISPs do not implement it, attackers are more likely to recruit zombies from within that network. In addition, this technique only filters zombies that spoof their IP address with one from another subnet. It does not prevent a zombie from spoofing its IP address with that of one belonging to a client within the same subnet. Also, since the attacker hides its identity through several levels of handlers and zombies, it may be feasible for the zombies not to employ IP spoofing, since it does not reveal the identity of the real attacker. Despite these issues, ingress and egress filtering can still help mitigate the attack because it limits the options of the attacker.

One of the most prominent methods to prevent a DDoS attack was proposed recently called Aggregate Congestion Control (ACC) with Pushback [15, 16]. In this scheme, an attack is detected by observing the number of dropped packets at a given router. If the number of dropped packets exceeds a given threshold, the router will identify (typically by the destination address) the packets that are consuming a large amount of the bandwidth, which are referred to as high-bandwidth aggregates. The router will perform rate-limiting on these packets. The rate-limiting feature is called the ACC mechanism. If the rate-limiting at the current router does not solve the congestion problem, a pushback message can be sent to a router upstream. The message is sent to the routers that are

forwarding the traffic that is supposedly from the attacker. The purpose of this message is to invoke the ACC mechanism at the router upstream. Ideally, these pushback messages should propagate as far upstream as possible to the point where the traffic enters the network. Thus, rate-limiting can be performed without affecting the rest of the traffic in the network.

The authors of these papers admit that the combination of ACC and Pushback can mistakenly diagnose or identify a normal client as an attacker. Since the decision to rate-limit a flow of traffic is typically based on the destination IP address. Thus, traffic from a legitimate client destined for that same address will also be rate-limited. The authors identified this as poor traffic; they identify the attack traffic as bad traffic, and the traffic belonging to legitimate clients not affected by ACC mechanism as good traffic. The combination of ACC and Pushback is a very promising scheme that has been one of more effective methods to defend against DDoS attacks. However, it requires changes to many routers throughout the Internet and the scheme itself is not always perfect in correctly identifying attackers.

Overlay networks have recently been proposed as a proactive approach to defend against DoS attacks [17, 18]. Overlay networks introduce a system with a protected internal network that only allows approved traffic to enter. To protect the resources of the victim, it is placed inside this protected internal network. Filtering is performed at the edge of the protected network so malicious users cannot enter. When a packet reaches the protected network, it is routed through a series of routers until it reaches its final destination. This process is referred to as overlay routing. The identities of some of the routers within the network are hidden so they cannot be targeted unless they enter at the network edge. To enter the network edge, decisions are made based on the credibility of the client. Lakshminarayan et al. identified weaknesses of overlay networks by stating that they assume that the list of clients are known in advance and that it does not scale very well to the current Internet setting [19]. This certainly holds true, because it is very difficult to determine if the client in question is malicious.

2.3. Client Puzzles as a Mitigation Technique

Before discussing client puzzles, it is necessary to define a cryptographic puzzle. The concept of a cryptographic puzzle was first proposed by Merkle in [20]. In this paper, a public key cryptosystem is introduced that uses the difficult problem of cryptographic puzzles rather than the discrete logarithmic problem used in the Diffie-Hellman cryptosystem [21].

In essence, a cryptographic puzzle is defined by Merkle as a cryptogram that is meant to be broken. In Merkle's scheme, the cryptogram is encrypted with a strong encryption function and a part of the solution is revealed to the solver. Thus, in this scheme a puzzle would consist of a plaintext, a ciphertext, and a part of the key. The remaining unknown bits of the key would be the solution to the puzzle. In order for the solver to find the solution to the puzzle, a brute-force approach must be applied that tries random values for the remaining bits of the key and then checks the value of the ciphertext to determine if the correct key has been found.

To apply this strategy to a public-key cryptosystem as Merkle did, one person (Bob) would create a large number of puzzles. Each puzzle solution would consist of an ID and a key, both of which are random variables. When a particular person (Alice) wishes to communicate with Bob, then Bob would send Alice a large number of previously created puzzles. Alice would solve only one puzzle at random to retrieve the ID and the key. This key would be the private key, or shared key, that is stored by both Alice and Bob. After having solved one puzzle, Alice sends the ID back to the server. Since Bob created the puzzles beforehand, it has a table of ID values and private key values. Thus, a key agreement can be established without sending the actual value of the private key. An attacker in between these two would only know the ID value and the N puzzles. Thus, in order to find the private key shared between Alice and Bob; the attacker would have to solve N puzzles. Thus, solving this many puzzles is the difficult problem used for this

cryptosystem. Although this scheme is not widely used in practice, it was one of the first attempts to introduce a novel method for public-key cryptography.

2.3.1. Client Puzzles at the Transport Layer

Juels and Brainard first introduced client puzzles to prevent connection-depletion attacks at the transport layer [22]. In addition, client puzzles have been devised to help prevent DoS attacks on authentication protocols [23]. Over the past few years, there have been a number of variations of client puzzle schemes that have also been proposed [24, 25, 26, 27, 28, 29, 30]. The basic idea of a client puzzle is that when a server is under attack, it sends out a cryptographic puzzle for the client to solve before allocating resources or performing an operation for that client. A cryptographic puzzle is created by taking a difficult problem from an appropriate cryptosystem and making it "feasible" by providing helpful information that will aid in finding the solution. This information reduces the solution search space so that the puzzle solver can simply apply brute-force techniques to find the correct solution. Therefore, as the name implies, puzzles are solvable problems that require both the solvers time and effort.

Since an attacker typically generates a large number of requests, it will have to solve a correspondingly large number of puzzles. In contrast, the legitimate client typically has only a small number of puzzles to solve. This method is effective in separating the attackers from the legitimate clients, and also gives the legitimate clients a better chance to have its requests completed. In order for a client to prove that it is legitimate, it must use its own resources and time to show that it is not an attacker (by solving a puzzle). The ideal characteristics of a client puzzle protocol are summarized below [14, 15, 16]:

- First, a puzzle should be easy for the server to create and verify, and should be much more difficult for the client to solve. The level of difficulty can be parameterized, and can be changed if needed. However, if the server is not under an attack, it should be possible that a puzzle would not be generated at all, allowing the client access without solving a puzzle.

- Second, it should not be possible for an attacker to keep a table of known puzzles and solutions.
- Third, the client should know that it has the correct answer before submitting it to the server. The puzzle solving process involves a repetitive brute-force task. The client should know when to terminate this process when it has the correct solution.
- Fourth, the server should know what puzzles it has generated and which ones to verify. There must be some type of mechanism in place that prevents an attacker from fabricating its own puzzle and sending its own solution to the server. The server needs to store a small amount of information so that it can determine which responses from the clients are solutions to valid puzzles.

The common problem among all of the client puzzle schemes that have been proposed is the puzzle verification, which involves the execution of a hash function or an encryption function to verify the client's answer. As mentioned in the first chapter, when creating a DoS resilient protocol, it is imperative that the defense mechanism itself does not become the source for another DoS attack. An attacker can easily attempt to exhaust the computing power of a server by forcing it to verify a large number of incorrectly solved puzzles. In this scenario, the attacker does not bother solving the puzzles; it simply intends to force the server to waste its resources. Optimizing the puzzle verification mechanism is critical and doing so will undoubtedly improve the server's performance.

In the vast majority of client puzzles that have been proposed, solving a puzzle involves reversing a one-way hash function by brute force. Depending on how the difficulty of the puzzle is set, the puzzle can be trivial or impossible to solve[1]. The puzzles presented in the related literature use different algorithms but they all require the client to solve the puzzle by brute-force [14, 15]. The entire reversal of a one-way hash function is considered computationally infeasible, but by revealing a portion of the answer to the client, the server creates a puzzle that is solvable. In hash-based puzzle algorithms, the

[1] In general, adjusting the difficulty involves revealing varying degrees of a puzzle's solution.

client has knowledge of an output value and a part of the corresponding input value to a hash function. Instead of attempting to reverse the hash function, the client solves the puzzle by using a brute-force method to find the rest of the input value. In a *server-generated puzzle*, the server generates the partial hash input and hash output, and transmits both pieces of information to the client. An example of a server-generated puzzle can be seen below in Figure 2-1. The solution to the puzzle is the value *S*. The server provides *X, Y*, and the hash function *h()* to the client.

$$h(X \,||\, S) = Y$$
$$X - \textit{puzzle parameter provided by server}$$
$$Y - \textit{puzzle parameter provided by server}$$
$$S - \textit{puzzle solution}$$

Figure 2-1: A server-generated puzzle

The difficulty of this puzzle is defined by the size of *S* in bits. The greater *S* is in size, the more difficult the puzzle is to solve. Also, the larger *S* is the smaller *X* will be and when the puzzle is at the highest difficulty level, there will be no value for *X*. The client will be responsible for fully reversing the hash function *h()*, which is considered computationally infeasible.

There is another method for creating hash-based puzzles. In this method, the client is only given the partial hash input, and it needs to solve for both the rest of the hash input and the hash output [15]. We refer to this type of a puzzle as a *client-generated puzzle*. The difficulty level is set by forcing the first *d* bits of the output to be zero. Thus, the server only needs to distribute a random number and the client will be responsible for creating the puzzle. In a client-generated puzzle, the server needs to keep only two local variables (a nonce and the difficulty level), and respond with these values when a client makes a connection request. An example of a client-generated puzzle can be seen below in Figure 2-2. The server provides the value *X* and the hash function *h()* to the client. When the server verifies the puzzle solution, it simply concatenates *X* and *S* to form the input of the

hash function, executes the hash function, and verifies the output satisfies the current difficulty level requirement.

$$h(X \,||\, S) = 0_0 0_1 ... 0_{d-1} 0_d \,||\, Y$$

X – *puzzle parameter provided by server*

Y – *puzzle solution* (*not sent to server*)

S – *puzzle solution* (*sent to server*)

d – *puzzle difficulty*

Figure 2-2: A client-generated puzzle

In [24], Wang and Reiter present a client puzzle protocol called a *puzzle auction* that was implemented and embedded into the TCP stack in Linux. The puzzle auction was designed to allow clients, with a modified kernel, to bid for a connection. The puzzle difficulty is the bid, and a higher bid implies a more difficult puzzle to be solved by the bidder. In their incremental bidding scheme, the client bids for a connection by solving a puzzle at a certain difficulty and can re-bid for a connection by solving another puzzle of a higher difficulty level and retransmit the request. At the end of bidding, the server grants connections to those who have the highest bid, or those who have solved the most difficult puzzle. Allowing a client to set the difficulty level may permit an attacker to control a zombie to purposely raise the difficulty level and to outbid other clients. The authors claim that most DDoS tools that execute on zombies are designed to operate quietly so that users will not notice their existence. They assume that solving puzzles repeatedly and of incremental difficulty will signal a user that their computer has been comprised. This is not always true, especially if the attack is launched during non-active times or using zombies where there is little human interaction with the computer. Another assumption relevant to this issue is that the user of the zombie computer is a knowledgeable computer user and would take the appropriate actions when he/she realizes that his/her computer is being used as a zombie. Again, this is not always true. For the reasons stated above, the difficulty level of a puzzle scheme should always be controlled by the server, or the end host distributing the puzzles.

For backwards compatibility, the puzzle auction scheme has a method for allowing a client with an unmodified kernel to complete a connection during an attack. Unfortunately, this implementation has potential vulnerabilities. In the implementation, a client with an unmodified kernel does not solve a puzzle. Instead, when the server receives a connection request, it is the server who computes the hash of a nonce, the source IP address, the source port, the destination IP address, the destination port, the initial sequence number, and another random value. If the output from the hash function meets the difficulty level, the connection is completed. A client can only establish a connection by repeatedly making attempts, and hope that the server can grant it. They call this scheme "Bid and Query". This scheme clearly violates the first characteristic of an ideal client puzzle protocol mentioned above, because the client and server are both required to perform a similar amount of work to complete the connection. This vulnerability can be taken advantage of by a malicious client.

Waters et al. [27] propose a new method to outsource client puzzles at the transport layer and briefly describe how their implementation could be modified for puzzles at the network layer. In their scheme, they claim that most clients do not wish to wait for a puzzle to be solved in order to complete a connection request. Instead, they introduce a complex method to solve puzzles before connection requests are actually made, thereby eliminating the time spent waiting for a puzzle to be solved. In their scheme, puzzles are solved in a certain time period, and then the puzzle answers are used in the following time period. The problem is that when a client first connects online it has to wait until the next time period to begin in order to use the answers it has computed from the last time period. The authors suggest that this time period be on the order of minutes and even use a time period of 20 minutes in their experiments. This means that a client, who had recently just connected online, would need to wait for 20 minutes before making a connection with a remote server. In their paper, one of their main contributions is decreasing the puzzle verification time. Since puzzles are solved in a specific time period, the puzzle solutions are stored during that time period and the next time period.

During the next time period, verification of a puzzle can simply be done with a table look-up. In terms of verification time, this is the fastest method to verify a puzzle. However, storage and complexity are sacrificed at the expense of the puzzle verification time.

2.3.2. Client Puzzles at the Application Layer

Client puzzles have also been proposed to combat attacks that may occur at the application layer. In [26], the authors have modified a webserver to support client puzzles. In TLS, a DoS attack can occur when the attacker targets the CPU of the server. When a secure connection is established, the server is required to perform CPU intensive RSA decryptions. If malicious users repeatedly force the server to perform these operations, the server's CPU will be exhausted and the server will not be able to service other clients. The authors in this paper propose a client puzzle scheme, where the client solves a puzzle before the server performs any work. Thus, the client puzzle protocol is able to shift the cost of the connection to the client, rather than the server. This client puzzle protocol is application specific and was designed to fix an exploit that allowed an attacker to exhaust the server's resources. By utilizing a client puzzle protocol, it was more difficult for the attacker to successfully exhaust those resources.

One of the more promising uses of client puzzles at the application layer is deploying them to combat junk e-mail or spam. The authors of [31] were the first to propose a scheme where a user solves a difficult problem before an e-mail is sent. Since spammers, on average, send more mail than legitimate mail users, they argue that forcing each e-mail sender to solve a cryptographic puzzle for each piece of e-mail, it would throttle the malicious users. In their future work, they have also proposed using a memory-bound problem rather than a computational-based problem [32]. Memory-bound problems were originally presented in [33]. In their paper, their idea was to develop an improved puzzle that forces the client to devote its memory resources to solve moderately difficult problems. Despite the concept of memory-bound problems, very few client puzzle

protocols have been proposed that use this kind of problem because they are difficult to create.

In addition to researching a proof-of-work protocol like client puzzles within the transport layer, Back has considered the possibilities of utilizing such a protocol within various applications [34]. The author has proposed a scheme called Hashcash, which can be deployed within an application to throttle spam, service requests in a cryptographic file system, or USENET flooding. There are many possibilities for using a proof-of-work protocol to prevent a malicious user from undermining the service and possibly denying it to legitimate users.

2.3.3. Client Puzzles at the Network Layer

Recently, there have been two papers [28, 29] published that address the need for IP-layer client puzzles. The basic idea for IP client puzzles is to have each client solve a puzzle before they can send traffic into the network. Thus, IP client puzzles are designed to rate-limit clients that send large amounts of data to help prevent large-scale DDoS flooding attacks.

In *Congestion Puzzles* by Wang and Reiter [29], each client is responsible for solving a number of puzzles before their packets can be forwarded from a congested router. When the clients begin to send packets towards a particular IP address it continuously sends separate probe packets, along with the data in normal packets it is sending. When a router downstream detects congestion, it relays the probe packets off the destination by changing the ICMP code number to resemble a ping when it reaches the victim. This packet will be modified to contain the puzzle information: a nonce and a difficulty level. When the client receives the challenge it begins to continuously solve puzzles and embed the solutions in separate ICMP packets. It takes the nonce it received from the router, creates its own nonce and uses those two items to create a hash-based puzzle. Therefore, the client does not need to contact the congested router to get a new puzzle. Solutions are

sent in ICMP packets sent to the destination but are intercepted by the router for puzzle verification. After correct verification of the puzzle, tokens are added into a token bucket at the congested router. When a data packet arrives, tokens are removed from the bucket. Therefore, this acts as a rate limiter and only allows the client to send more packets if they in turn solve more puzzles. Periodically, the puzzle information (i.e. a nonce and the difficulty level) is refreshed by sending back a puzzle solution with the updated information. Thus, in this scheme, twice as many packets are being sent to a congested router. This can create a potential problem, because an IP-layer puzzle protocol should limit the amount of traffic sent to the victim. Recall that one of the goals of a DDoS attack is to consume the bandwidth near the victim. Another concern in this protocol is that an attacker can take advantage of the token bucket design by flooding the network with packets (without solving puzzles) and hoping that it removes tokens that were supplied earlier by legitimate clients. The authors are aware of this problem and call it the "free-riding" problem. The authors attempt to fix this problem by introducing an IP-caching algorithm that allocates a separate token bucket for clients that are sending more data than others or for those with a common IP prefix. However, this adds much complexity to their scheme and increases the memory storage overhead at the router.

The protocol presented by Feng et al., called *Network Puzzles* [28], requires the client and congested router to constantly exchange puzzle information for each puzzle. Unlike Congestion Puzzles, each client can only solve a puzzle when it has been presented with the puzzle information from the congested router. During a severe attack, each client needs to be rate-limited by solving more puzzles. If a client needed to solve a puzzle per packet, it would repeatedly need to request the puzzle information from that congested router before it can send its packet. This scheme does not allow a seamless integration of the puzzle protocol into IP because it does not allow the client to create its own puzzles.

In all client puzzle protocols it should be clearly stated that puzzles should only used during an ongoing attack. We realize that there are many applications that rely on the speed of certain protocols for performance. However, the performance of these network applications suffers immensely in the presence of a DoS attack. The additional delays

associated with solving puzzles might hinder some applications, but it will be an improvement over the side effects that are observed during a DoS attack when puzzles are not deployed. The idea of implementing client puzzles into the transport layer and network layer is still relatively new in the research community, as there have been very few conceptual designs of such a protocol.

CHAPTER 3

3. TCP Client Puzzles

In recent years, TCP client puzzles have been proposed to mitigate attacks on the transport and application layers. In this chapter, the design, implementation details, and simulation results of Puzzle TCP, pTCP, are presented. However, before an introduction to pTCP is given, it is necessary to define the assumptions that are made. These assumptions are shown in Table 3-1.

Table 3-1: Assumptions in pTCP

Assumption 1:	Attackers are generally more aggressive and send more requests than the average legitimate client.
Assumption 2:	The server under attack can process every incoming packet and send responses to each client.
Assumption 3:	If a stateful firewall is protecting the victim server, the puzzle mechanism can be embedded into the firewall, or rules can be added to the firewall (to allow ACK packets without state information to pass through) if the puzzle mechanism is chosen to be implemented at the server.

3.1. Overview of pTCP

In order for client puzzles to be truly effective, they must be placed below the application layer. Feng [30] documented the need to place client puzzles in the TCP or IP-layer. In the following section, we show how client puzzles can be integrated within the TCP stack

to prevent resource-exhaustion DoS attacks. This section also gives a detailed description on how the client and server interact in pTCP.

The current three-way handshake implemented in TCP has lead to security problems, mainly because the server can allocate resources before clients are authenticated. To prevent a client from depleting a server's resources, it is necessary to modify TCP. pTCP is a modification to TCP that allows the server to issue a challenge to a potential client. Based on the number of pending active connections, the server should be able to determine if newer clients are required to solve a puzzle before establishing a connection. By placing the client puzzle protocol at the transport layer, we force a client to prove its legitimacy by solving a puzzle before a connection is granted.

pTCP implements a similar three-way handshake to establish a connection. When a server receives a packet with the SYN code bit set, it normally replies with a packet with the SYN and ACK code bits set (SYN-ACK packet). However, if the server is experiencing heavy traffic (e.g., flash crowd or DoS attack), then it replies with a challenge to the client which includes a server nonce and difficulty level embedded into the SYN-ACK packet. A server can determine when this is necessary by examining the SYN queue. When the queue reaches near its maximum capacity, puzzles can be turned on. When it is necessary for a server to issue a challenge to a client, the server removes the state information for that client. Thus, for pending connections there are no resources allocated on the server other than the server nonce and current difficulty level, which is common to all potential clients. Therefore, the server is not susceptible to half-open attacks designed to consume server resources. The server nonce is the same for all clients during a 60 second time period. If a client solves a puzzle in the previous time epoch and submits the answer in the following time epoch, the answer is considered incorrect. The client's connection will be reset and the client will need to make another attempt to complete the connection. Since most puzzles take less than a few seconds to solve, the 60 second server nonce period should be adequate.

Creating puzzles in pTCP is very fast for the server because server only needs to send the server nonce and difficulty level embedded in a SYN-ACK packet. When the server issues this challenge to the client, the client parses the SYN-ACK packet for the server nonce and difficulty level, solves the puzzle, and replies with a solution in an ACK packet. The server then verifies the correct solution and changes the state of the connection to "established". Figure 3-1 illustrates how a client and a server communicate in pTCP.

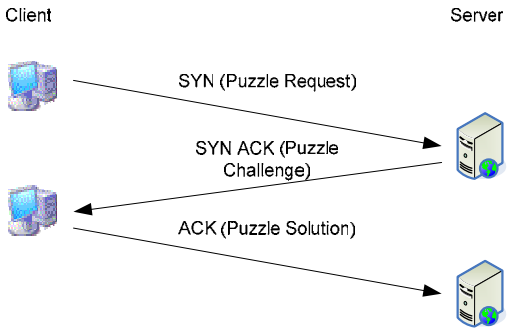

Figure 3-1: Client-Server interaction in pTCP

In pTCP, the server does not issue puzzle challenges during normal traffic conditions (which is indicated by the vacancies in the SYN queue). In this case, the server sends an acknowledgement (i.e., a normal SYN-ACK packet) to the client instead of a puzzle challenge. The client replies with an acknowledgement (i.e., a normal ACK packet). This flexibility of pTCP will allow clients with unmodified versions of TCP (i.e., TCP without puzzle challenge capability) to establish connections with servers using pTCP when the SYN queue level is sufficiently low. This feature facilitates backward compatibility with puzzle incapable clients and also helps to support the gradual deployment of client puzzles in the Internet. In practice, the backward compatibility is very important because the implementation of client puzzle protocols require changes in the clients' software. A smooth migration from standard TCP to puzzle capable TCP will require the protocol's backward compatibility.

A problem common to both pTCP and syncookies is the inability of the server to retransmit the SYN-ACK packet when this packet is lost en route to the client. Since all

state information is removed on the server following the transmission of this packet, retransmissions are not possible in either scheme.

3.2. The Client Puzzle for pTCP

In pTCP, we propose an entirely different puzzle algorithm. Our puzzle algorithm is based on a lightweight block cipher. The solution to a given puzzle is found by finding the appropriate key given the plaintext that produces a particular pattern in the ciphertext; the solution is verified by executing the encryption algorithm using the plaintext and key. Our simulation results have shown that our technique enables faster verification of puzzle solutions compared to conventional hash-based puzzle algorithms. The efficiency of the puzzle verification procedure is vital to the overall effectiveness of the client puzzle protocol. In a conventional puzzle algorithm, a server executes a hash function to verify the client's answer. When creating a DoS resilient protocol, it is imperative that the defense mechanism itself does not become the basis for another DoS attack. An attacker can easily attempt to exhaust the computing power of a server by forcing it to verify a large number of incorrectly solved puzzles. In this scenario, the attacker does not bother solving the puzzles; it simply intends to force the server to waste its resources. The computation load of verifying a puzzle solution must be multiple orders of magnitude smaller than that required to solve the puzzle. The proposed puzzle algorithm is designed to minimize the verification time and enables a server to verify a large volume of puzzles rapidly while still maintaining its intended DoS resiliency.

The *Tiny Encryption algorithm* (*TEA*) is a block-cipher encryption algorithm that was proposed in 1994 by Wheeler and Needham [35, 36]. Both the encryption and decryption algorithms are Feistel routines that encrypt or decrypt data by several rounds of addition, subtraction, bit-shifting, and exclusive-OR operations. The goal of the encryption algorithm is to create as much diffusion[2] as possible by incorporating many rounds or iterations of these operations. After TEA was released, certain minor weaknesses were discovered in the encryption algorithm [37]. In response, Wheeler and Needham

[2] In TEA, complete diffusion means that a single change in the plaintext propagates to 32 changes in the ciphertext.

developed an extension to TEA, called XTEA [38]. For our client puzzle algorithm, we use a variation of XTEA which uses 6 iterations. We call this variation *XTEA6*. In this encryption scheme, the plaintext is 64 bits long, the key is 128 bits long, and the ciphertext is 64 bits long. The encryption routine allows a parameter to be passed that indicates the number of iterations to execute. We have selected six iterations because complete diffusion can be observed with that many rounds. Moreover, the level of security provided by six rounds is more than sufficient for a puzzle [38]. Note that in a puzzle algorithm, speed and efficiency are more important than robust security. It should be easier to solve the puzzle correctly than perform an extensive cryptanalysis of the encryption algorithm to retrieve the puzzle solution.

To the best of our knowledge, TEA was first suggested as a puzzle algorithm by Abadi et al. [33], although an implementation of a client puzzle protocol using TEA was not attempted. As a client puzzle, XTEA6 has several advantages over most hash-based functions. As our simulation results will later show, XTEA6 is a much faster algorithm than most hash-based functions. This will allow a server to handle connections faster and will decrease the amount of waiting time on connections to complete for legitimate clients.

The client puzzle that we have developed is a client-generated puzzle, similar to the client-generated hash-based puzzle mentioned in Chapter 2. When the client requests a connection, the server responds with a server nonce, N_S, and the level of difficulty for the current puzzle. The server nonce is a 64-bit random number, generated with the Linux kernel random number generator. The server nonce is the same for every potential client and is changed for all clients every 60 seconds. The client uses the server nonce as a part of the puzzle that it must create. When the client receives the server nonce, it uses it as the plaintext in the XTEA6 encryption algorithm. The client must then find part of a 128-bit key value that will produce a particular ciphertext. More precisely, to solve a puzzle, the client needs to solve for the least significant 32 bits of the key. The rest of the key is comprised of the server's initial sequence number (ISN), the server port, the client's local port, and the client's IP address. By using the client's port and the server

ISN, we guarantee that each puzzle will remain unique for each client. This concept was originally used in a puzzle by Wang and Reiter in [24]. The difficulty of the puzzle is specified by the number of most significant contiguous bits in the ciphertext that must be zero. For example, if the difficulty level is *k*, then the *k* most significant bits of the ciphertext must be all zeros, the *(k+1)-th* most significant bit must be a one, and the rest of the bits can either be a one or a zero. Therefore, to solve a puzzle, the client needs to find a 32-bit portion of the key that will encrypt the given plaintext into a ciphertext that satisfies the requirement specified by the difficulty level. To save communication overhead, the client only sends back the 32-bit solution portion of the key to the server. Because the plaintext (i.e., server nonce), the difficulty level, and the remaining portion of the key are known to the server, the server only needs to receive the 32-bit value to verify the correctness of the puzzle solution. A graphical representation of the puzzle algorithm is shown in Figure 3-2. To compare our puzzle scheme with others that have been suggested, in addition to the XTEA6-based pTCP, we also implemented a version of pTCP that uses the MD5 hash function [36]. In the MD5-based puzzle, the parameters and the size of the parameters are the same as the XTEA6 puzzle. A graphical representation of this MD5 puzzle scheme can be seen in Figure 3-3.

3.3. Implementation of pTCP

The following sections discuss the implementation details of pTCP. The details and algorithms of the design are presented along with other important details including a description of the code changes within the Linux kernel and some of the specific modifications to the existing TCP stack.

3.3.1. Overview of the Implementation

pTCP was implemented into the TCP stack in Linux [39]. In pTCP, all of the puzzle information (puzzle request, puzzle challenge, and puzzle solution) is passed using the TCP header. Since data is not normally passed in the three-way handshake in TCP, any additional data needs to be placed within the TCP header. The options field in the TCP header was utilized to pass the puzzle requests, puzzle challenges, and puzzle answers.

Since the options field is of variable length, many different options can be placed within this field at the same time as long as the header does not exceed its maximum length. The options field in the TCP header is organized into these sections: the option code number, the option size, and the contents of the option. Normally, many options in the header are passed between client and server during the three-way handshake, so the TCP stack was manipulated to handle and recognize more options.

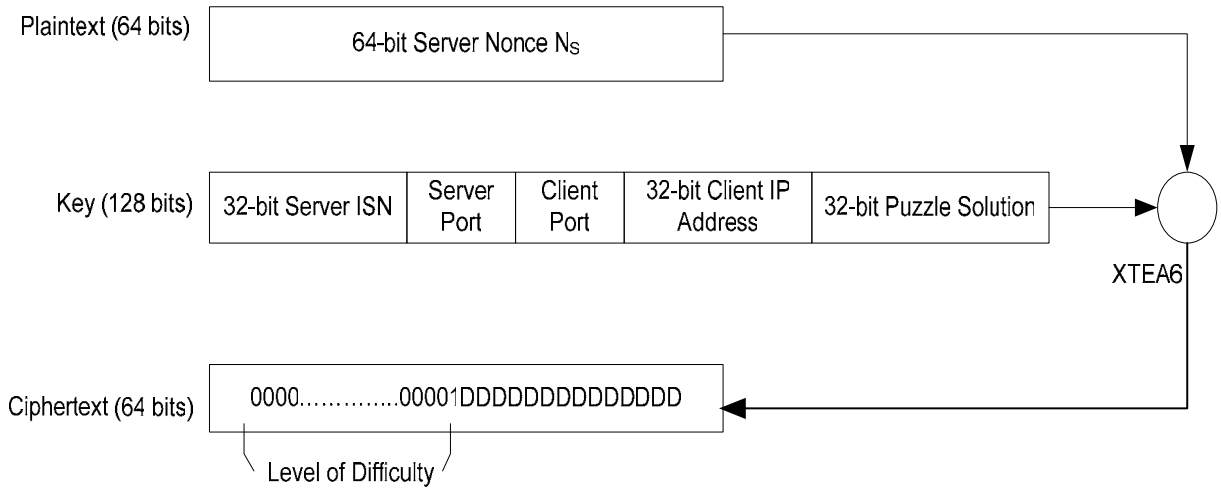

Figure 3-2: XTEA6 client puzzle

The three types of options were given code numbers to avoid conflicting with other known TCP options. The puzzle request, puzzle challenge, and puzzle answer were given codes 99, 100, and 101, respectively. In pTCP, every client sends a puzzle request embedded into the initial SYN packet. If a server supports pTCP, this request will act as a signal that this client is capable of solving puzzles. The contents of the initial SYN packet can be seen in Figure 3-4. When this packet is sent to the server, the server parses the options in the header and is ready to reply with a SYN-ACK packet. If the server determines that it is currently under a heavy amount of traffic, by examining the length of the SYN queue, it replies with a SYN-ACK challenge packet.

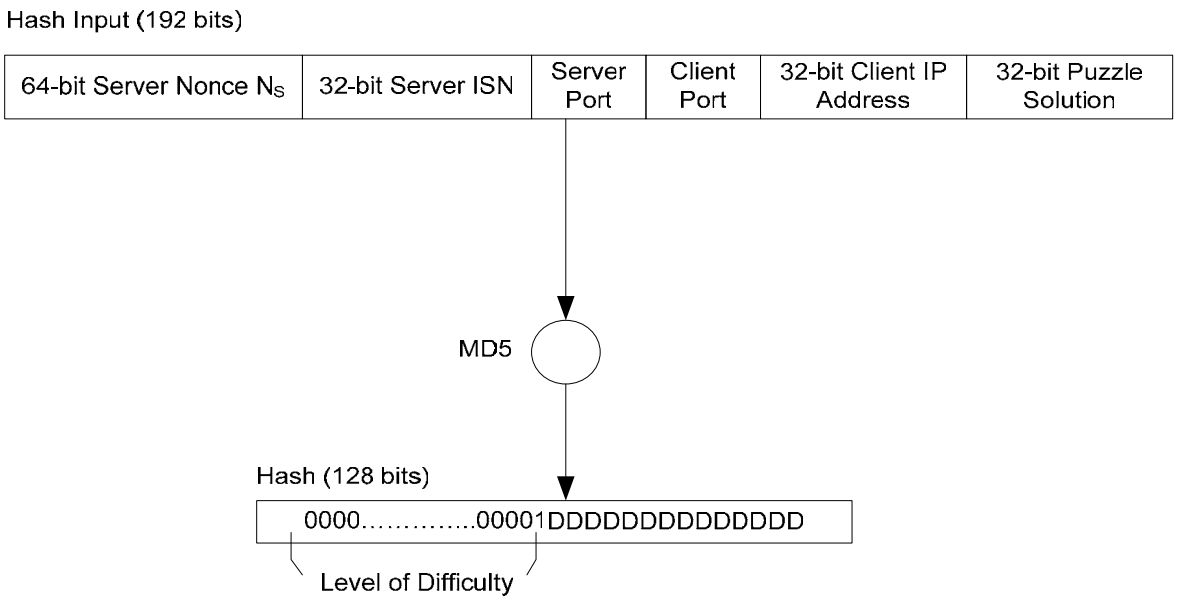

Figure 3-3: MD5 client puzzle

In a SYN-ACK challenge packet, the current puzzle difficulty level and the current server nonce are both embedded into the options field in the header. The contents of a SYN-ACK challenge packet can be seen in Figure 3-5. Following the transmission of a SYN-ACK challenge packet, the server removes the client information from memory, in the exact same way that syncookies removes client information from memory. Therefore, pTCP is resilient to synflood DoS attacks because no state information is stored by the server for pending connections.

The client, after having received the challenge packet, parses the options field in the header for the puzzle information. The client takes the 64-bit server nonce it has just received, and uses that value as the plaintext for the XTEA6 encryption algorithm. The client then examines the TCP and IP headers for the server ISN, the server port and the local port, and the local IP address. It uses these values as part of the key used for the XTEA6 algorithm. Inside the kernel, the client then solves the puzzle and finds an appropriate 32-bit solution. The pseudocode for the puzzle solving function is shown in Figure 3-6.

When the client has successfully solved the puzzle, it creates an ACK packet, as it normally does in the 3rd step of the handshake, but when it has been challenged it embeds the puzzle solution into the options field, as seen in Figure 3-7. When the server receives this packet, it is an ACK packet from an unknown client. The server calls a function to verify the puzzle solution. This function involves calling the XTEA6 encryption function only once. If the puzzle solution is correct, the state of the connection is changed to established, thus bypassing the half-open connection state. If the solution is incorrect, the server does not establish the connection with the client.

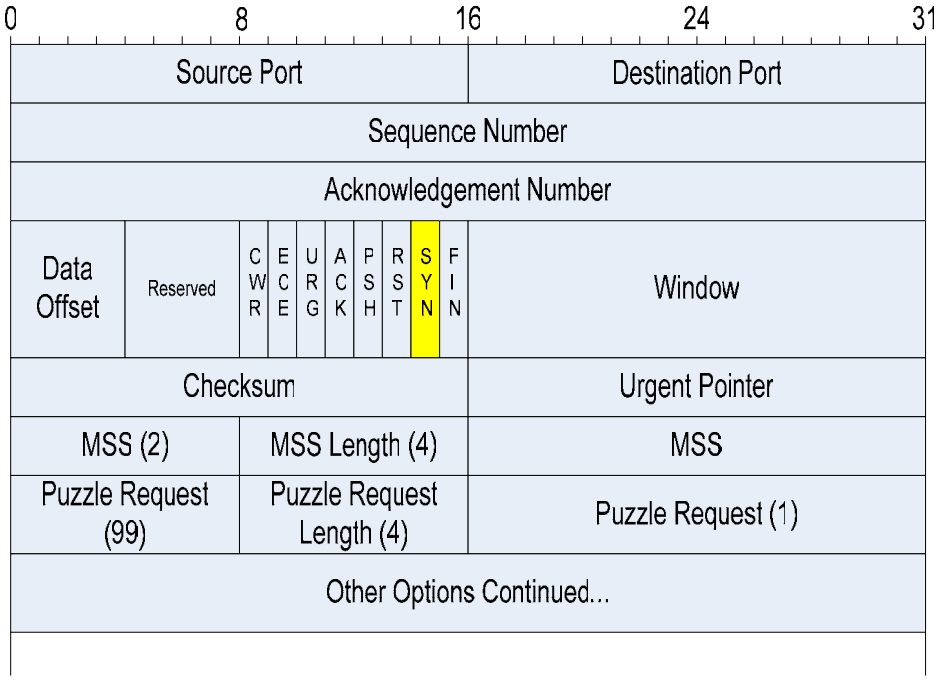

Figure 3-4: Puzzle request packet

Since the server accepts anonymous ACK packets with puzzle solutions, this can result in another type of an attack. An attacker could be sniffing packets and discover a solution to the puzzle that another computer has already solved. The attacker could then send this puzzle solution as its own. To counteract this, our puzzle uses various parameters from the TCP and IP header. By embedding these values into the puzzle, an attacker cannot use previous solutions. A more likely attack is when an attacker could flood the server with false puzzle solutions. This could potentially be another form of a DoS attack that

attempts to exhaust the server's CPU. It is important to always assume that attackers will modify their attack to exploit weaknesses in a modified version of TCP. In this case, an attacker would create an ACKflood tool rather than the traditional synflood tool. Since the client puzzle algorithm is extremely fast, pTCP would be able to discard these false answers quickly and efficiently.

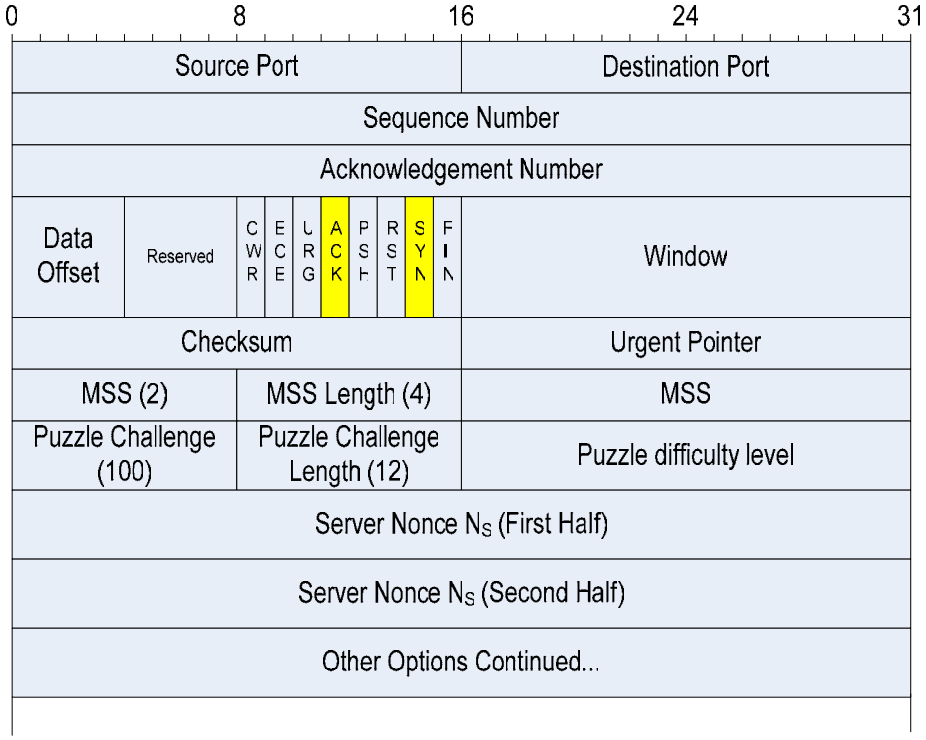

Figure 3-5: Puzzle challenge packet

```
Plaintext = N_S
Key[0] = Server ISN
Key[1] = Server Port || Local Port
Key[2] = Local IP address
While(Answer is not found)
        Key[3] = get_random_bytes( )
        Ciphertext = XTEA6(Plaintext,Key)
                If Ciphertext meets difficulty constraint
                        Return
                Else
                        Continue loop
```

Figure 3-6: Pseudocode for puzzle solving algorithm

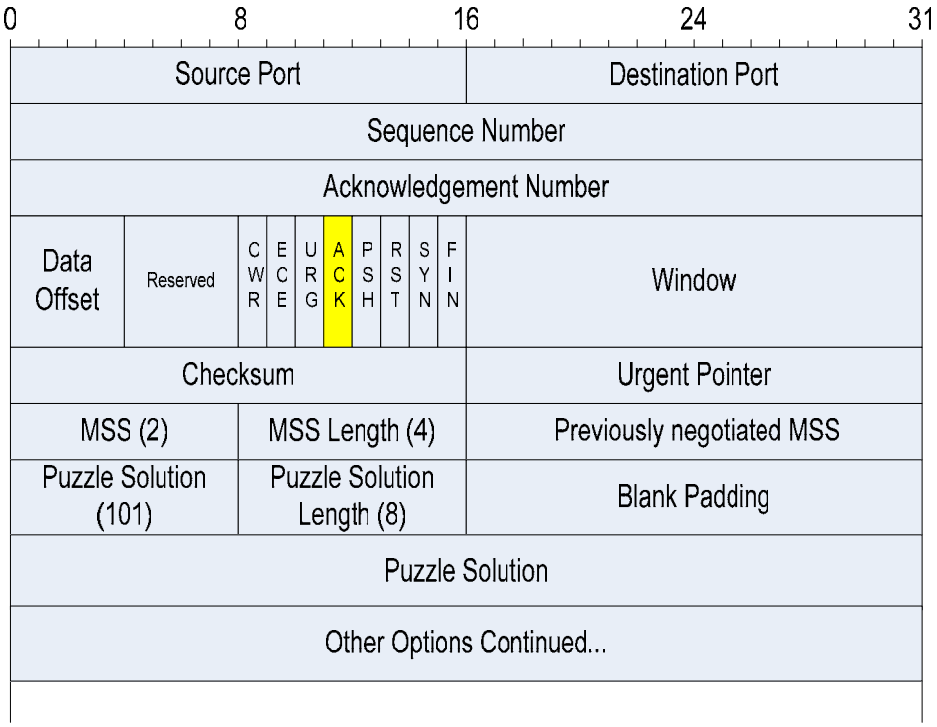

Figure 3-7: Puzzle solution packet

3.3.2. pTCP Implementation Details

pTCP was developed into the 2.4.22-10mdk (Mandrake) version of the Linux kernel [40]. The code that was modified was located in the IPv4 section of the current TCP implementation in Linux. The first step in modifying the kernel for pTCP, besides researching how everything worked together, was to embed specific additional state information for puzzles into the tcp_opt data structure, a structure that belongs to the larger sock data structure. The tcp_opt data structure stores most of the state information about a TCP connection as well as the information about the client at the other end. This location would be suitable because the puzzle information about each client should reside with the rest of connection information. The variables that were added to the tcp_opt structure were the following flags: *PuzzleCapable*, *PuzzleSolved*, and *Challenged*. In addition, the 64-bit server nonce, 32-bit puzzle solution, and the difficulty level were also stored in this structure.

When a client wishes to create a connection, the tcp_connect function is called that prepares and establishes the client's information (sequence number, options, etc.) for the initial SYN packet. When the information is established, the function eventually calls the transmit_skb function which builds the header and sends the packet to the lower layers of the network stack. A common structure in the TCP stack is the socket buffer structure, which is commonly named skb when it is a local variable. In this function when a SYN packet is being created the flag PuzzleCapable is set to one (PuzzleSolved and Challenged both remain equal to zero at this stage). This function calls tcp_syn_build_options so that it can further create the TCP header and embed the puzzle request. The tcp_syn_build_options function is used only in the initial three-way handshake to embed the additional options that are exchanged in the connection setup (Maximum Segment Size, Selective Acknowledge, etc.). This function was modified to build the extra options required for puzzles as well as the normal options that are usually exchanged. Therefore, no functionality was removed in pTCP. After the client sends the SYN packet to the lower layers of the network stack, it will eventually reach the correct destination. Until then, the client waits in the SYN_SENT state.

When the server receives the SYN packet from the client, the function tcp_v4_conn_request is called to process the request. This function calls tcp_parse_options which is responsible for parsing the TCP header options. While parsing the TCP options of the current header, the server can discover if the client is capable of solving puzzles by observing the Puzzle Request option sent by the client. The flag PuzzleCapable inside the tcp_opt structure on the server side is set to one. Before sending a SYN-ACK packet, the server examines the size of SYN queue. If the queue is almost full, which implies that there are a lot of connection attempts being made, the server will need to send a Puzzle Challenge. For the Puzzle Challenge, a SYN-ACK packet is created with the server nonce and difficulty level embedded into the header. Regardless of whether the client is capable of solving puzzles, a Puzzle Challenge is sent. We left this option open for future work; because it may be possible for a client to use a low-level software application that could interpret the puzzle information and respond with a solution. Nonetheless, the connection will never be established unless the client

provides solution. Following the transmission of the Puzzle Challenge, the state information is removed from the server, which makes it resilient to synflood attacks. If the server chooses not to send a Puzzle Challenge, a SYN-ACK packet is constructed without the puzzle information and following the transmission of that packet, the state information remains on the server.

When the client receives the SYN-ACK packet with a Puzzle Challenge, the tcp_rcv_synsent_state_process function is called and the options are parsed again so the difficulty level and server nonce can be retrieved. Once those values are read, they are stored in the tcp_opt data structure on the client side. The Challenged flag inside the tcp_opt structure is set to a one. Meanwhile, the PuzzleSolved flag remains zero. With the puzzle information it has retrieved from the SYN-ACK packet, the client performs a brute-force solving strategy to solve the given puzzle. When the puzzle is solved, the PuzzleSolved flag is set to one; the client creates an ACK packet (through the transmit_skb function), embeds the solution in the TCP header, and sends this packet to the server. From the client's perspective, the state of the connection is changed to "established".

When the server receives this packet it currently has no state information stored about that client or the connection, so it is normally dropped. However, a check was placed before the packet was discarded to call tcp_parse_options to examine the options field to check if it contained a puzzle solution. If the ACK packet contains a puzzle solution, the solution is verified by the server by using the current server nonce and difficult. If correct, the state information is re-initialized, the sequence numbers are initialized and set on both ends correctly, and the state of the connection is changed to "established". When the connection is established, the client and server can communicate normally. If the solution is not correct, it is ignored and the connection is not established and the current state information is deleted for that particular client. A modified version of the state-diagram for TCP can be seen in Figure 3-8 [8, 39].

3.4. Experiments with pTCP

To ensure that pTCP was operating correctly and performed well during an attack, we used the implementation and created a network testbed of several computers with the modified kernel to test how well pTCP operated during these attacks. In this section, we discuss some of the experiments that were performed with pTCP and present some of the results that were collected.

3.4.1. The Puzzle Algorithm

An important aspect to pTCP is the selection of the cryptographic algorithm used for the client puzzle. In addition to puzzles using XTEA6 and MD5 [36, 41], we also tested SHA-1 [36, 42]. According to our simulation results, the puzzle algorithm based on XTEA6 had the fastest solution verification time among the three that were examined. MD5 was also faster than SHA-1, so only XTEA6 and MD5 were implemented into the kernel.

To directly show how XTEA6 can improve the server's solution verification performance, we measured the amount of time spent verifying puzzle solutions within the Linux kernel. Our comparison of XTEA6 pTCP and MD5 pTCP is shown in Figure 3-9. These results show the major advantages of using XTEA6 instead of MD5. The MD5 puzzle scheme can verify 1000 puzzles in around 31,000 microseconds. Meanwhile, an XTEA6 puzzle scheme can verify 1000 puzzles in less than 4,000 microseconds. Since pTCP uses XTEA6 it is far more efficient because it can verify a large number of puzzles much faster than any other puzzle scheme. When traffic is heavy, pTCP can improve load conditions on the server which can in return reduce the connection times for clients. From the results in Figure 3-9, XTEA6 is the best choice for a client puzzle because it will increase performance on the server since the puzzle verification time is considerably lower than an MD5-based puzzle scheme. It should be noted that the solve time for the XTEA6-based puzzle could be faster than that of puzzles based on MD5 or SHA-1 (assuming the difficulty level are the same for all three algorithms). This also implies that

an attacker may solve the XTEA6-based puzzle faster. This fact should be considered by a server when setting the difficulty level of the puzzles.

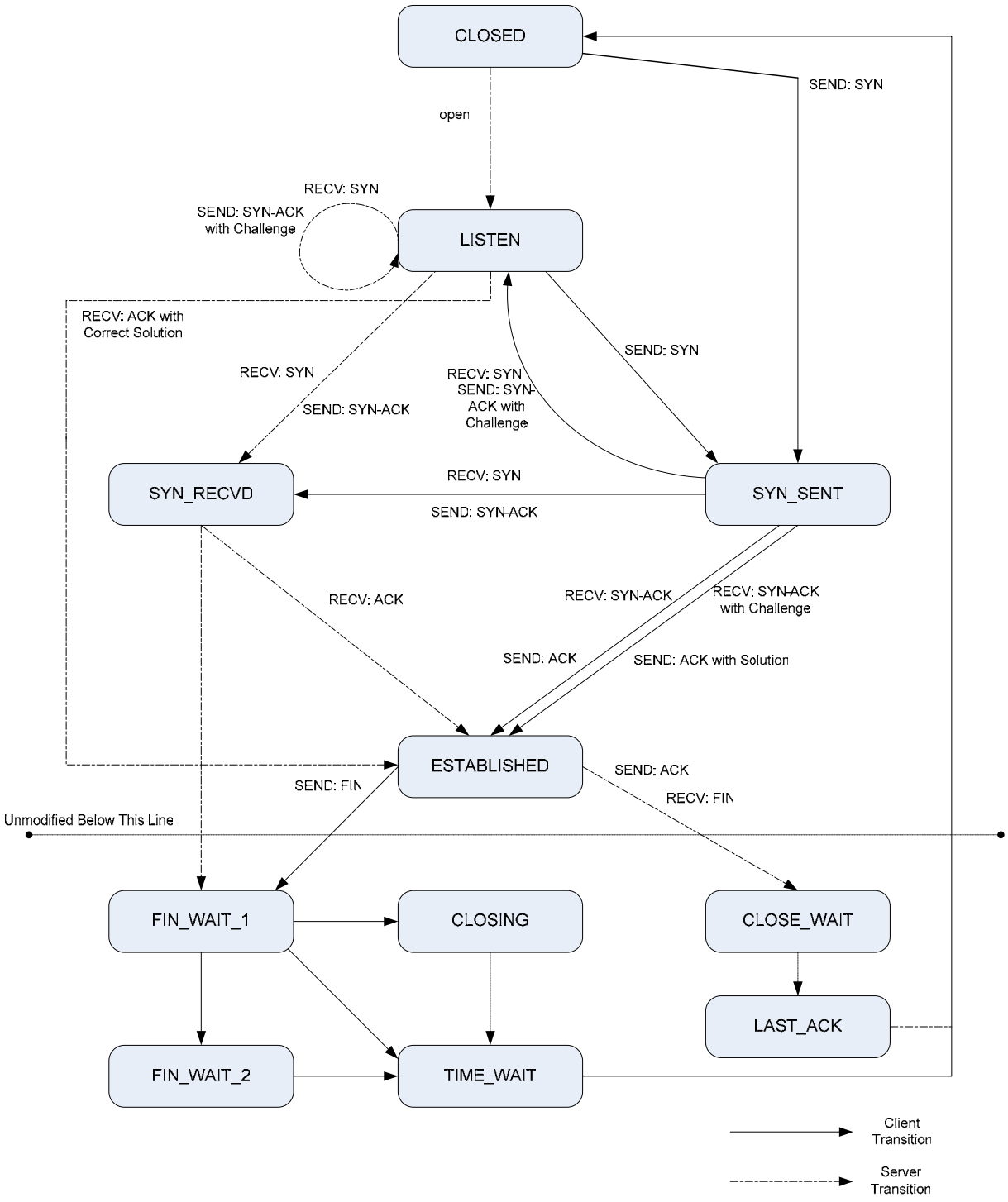

Figure 3-8: pTCP state transition diagram

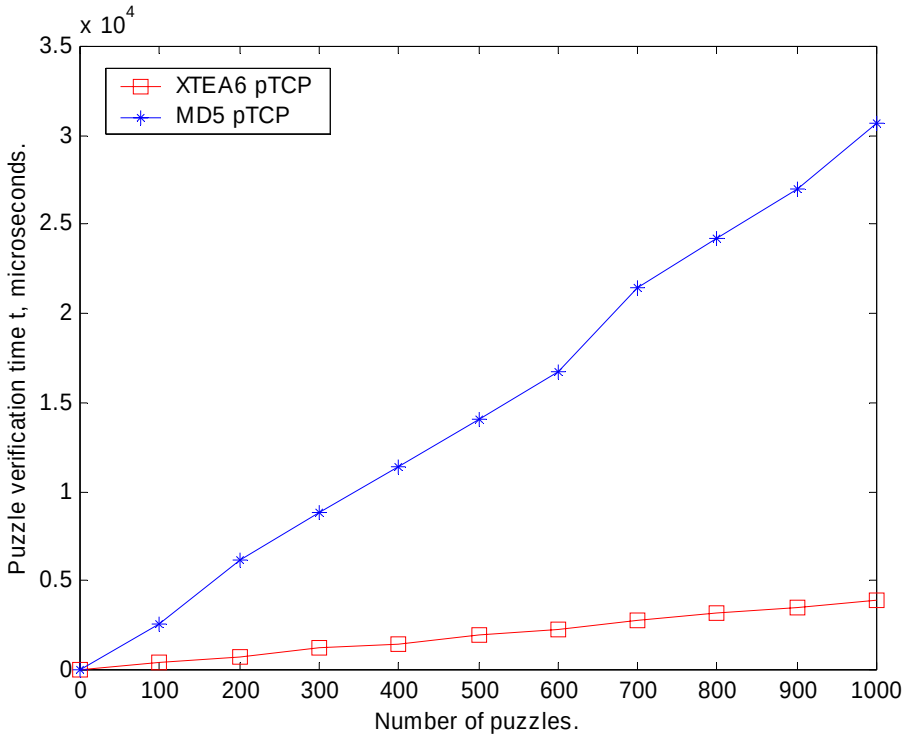

Figure 3-9: Puzzle verification times for pTCP

3.4.2. Modulation of the Puzzle Difficulty Level

In this simulation experiment, we created an environment that consisted of a server, a legitimate client, and an attacker. This environment will help determine the performance of pTCP and how it handles various attack scenarios. For our first simulation, the attacker executed 16 instances of a synflood attack program called synk4 [10]. In the standard TCP protocol, it created a sufficient amount of SYN packets to carry out a successful DoS attack. During this attack, a legitimate client could rarely complete a connection, if at all. With pTCP, a large number of half-open connections will signal the server to begin distributing puzzles.

In pTCP, an important quantity to measure is the amount of time needed to establish a connection versus the puzzle difficulty level. It is important to verify that increasing the difficulty level increases the puzzle solve time, which in turn increases the connection

time. For each connection attempt, the client is solving a puzzle and completing with the three-way handshake. In Figure 3-10, we show the average connection time versus the puzzle difficulty when the puzzle mechanism was enabled.

In this figure, one can observe that the client's connection time increases exponentially as the puzzle difficulty level is increased linearly. These results verify the fact that raising the difficulty level makes the puzzles more difficult, thus throttling the client's connection attempts. In addition, one can observe that as the difficulty level increases there is very little difference between XTEA6 and MD5 because the transmission time of the packet (propagation delay) dominates the differences between the two algorithms.

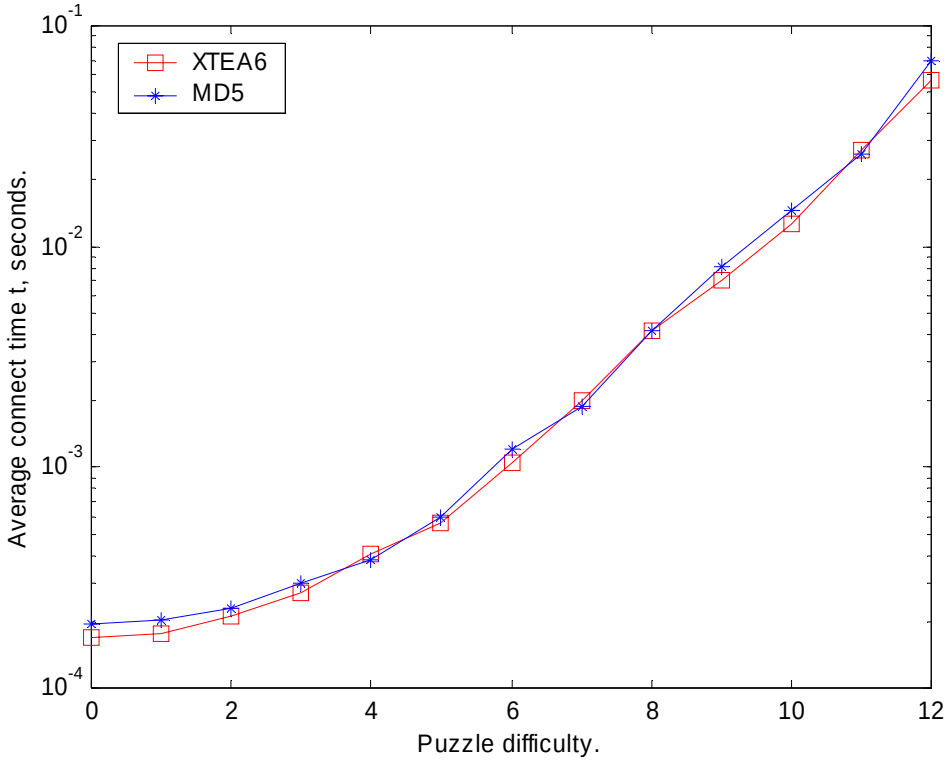

Figure 3-10: pTCP connection time versus puzzle difficulty

3.4.3. Performance of pTCP during a synflood Attack

In order to test the effectiveness of pTCP during a synflood attack, we compared the protocol against two versions of the TCP protocol—one with syncookies turned on and

the other with syncookies turned off. We also tested both versions of pTCP (puzzles based on MD5 and XTEA6).

The same simulation environment from the previous experiment was used again to test the performance of pTCP during a synflood attack. The attacker again executed 16 instances of the same synflood attack program. The tcp_syn_max_backlog[3] parameter on the server was set to the default size of 1024. The legitimate client made a large number of connections and we measured the amount of time it took to complete the connection. Our simulation results from this experiment are shown in Figure 3-11. In standard TCP with syncookies turned off, a legitimate client could not establish a connection with the server at all; every connection attempt had timed out after 5 retries, which from our observation is roughly 188 seconds. However, as expected, with standard TCP with syncookies turned on, every connection was completed. Recall that the syncookies scheme was designed to solely defend against synflood attacks. In our next experiment, under the same attack conditions, the server and the legitimate client both used pTCP, with the difficulty level set to zero. Due to the puzzle solve time, the puzzle verification time, and the increase in packet size, the connection times for pTCP were slightly greater than the connection times for syncookies. However, we noticed that XTEA6 pTCP was very comparable to syncookies because roughly 90% of the connections were completed by the same amount of time. The time difference between XTEA6 pTCP and syncookies for a connection request to complete is less than 30 microseconds. This small difference is due to the extra data being sent in each packet. The results show that pTCP can be effective in defending against synflood attacks and could be considered as an alternative to syncookies. The fact that pTCP can process TCP options (such as the maximum segment size) better than syncookies is another important advantage to pTCP. These TCP options are important for increasing performance for certain clients and servers where there is a larger amount of bandwidth. The inability to take full advantage of these options can have an impact on the performance of the connection throughout the client's entire session with the server. Having a mechanism that can prevent these attacks and still be able to recover the actions from the three-way handshake is the best choice for

[3] The tcp_syn_max_backlog parameter specifies the maximum number of half-open connections the server can store.

designing a modification to this protocol. Syncookies could be fixed to resend the options that were lost in the three-way handshake, but it was not done, since it would require a change to the client. We feel that if the operating system of the client is to be modified, it would be more appropriate to use pTCP because it can protect the server from more attacks.

3.4.4. Performance of pTCP in CPU-Exhaustion Attacks

A synflood attack is only one example of a resource-exhaustion attack. Another example of an attack is a CPU-exhaustion attack, which is a form of a connection-depletion attack. In this attack, the attackers attempt to exhaust the victim's CPU so that it can no longer communicate with other clients.

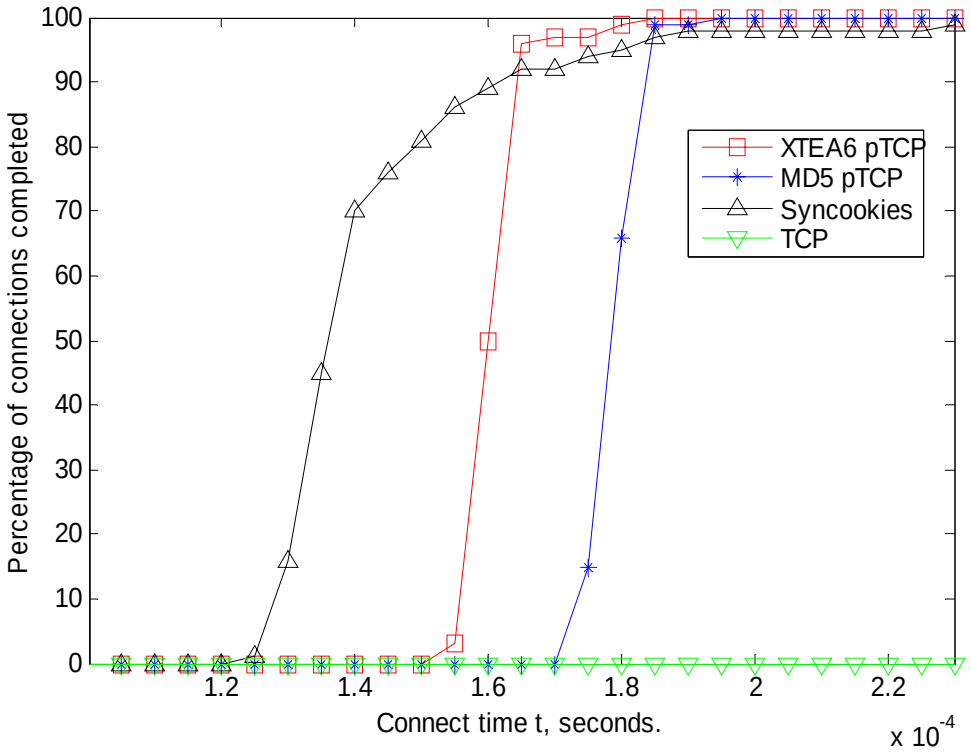

Figure 3-11: Percentage of connections completed versus connection times for pTCP, syncookies, and TCP

For our next experiment, we used a similar network testbed with 3 computers: a legitimate client, a victim, and an attacker. In this attack, the attacker attempts to exhaust the computing resources of the victim. This is done by making the victim perform expensive and meaningless operations that take up CPU cycles after a connection is established—thus the name CPU-exhaustion attack. In this attack, the attacker actually completes the three-way handshake.

While a CPU-exhaustion attack was under way, we measured the average time required to complete a three-way handshake between the client machine and the victim machine. One machine representing the legitimate client generated all the connection requests. With TCP with and without syncookies, some of the connection attempts timed-out while others simply took too long to complete (greater than 40 seconds). Due to an increase in connection attempts, it appears from the experiment that the half-open connection queue had reached its limit at certain times, which explains why syncookies outperformed standard TCP. As expected, pTCP outperformed syncookies because the puzzles effectively rate limited the connection attempts of the attacker machine. Because the attacker was forced to solve a puzzle for every connection attempt, the computation load of computing the solutions naturally slowed its connection request rate. This in turn mitigated the effect of the attack and allowed the victim to service requests coming from the legitimate client. Note that TCP with syncookies prevents the attacker from filling the half-open connection queue, but cannot rate limit the attacker's connection requests. The results are shown in Figure 3-12.

When comparing XTEA6 pTCP with MD5 pTCP, there was very little difference between the two. Unlike the synflood attack from the previous experiment, there was a moderately difficult puzzle for the client to solve. In the previous experiment, the difficulty level was zero, so it only required each client one trial to find a correct solution. As the difficulty level increases it appeared that the differences between the two protocols were very small, which is evident from the results in Figure 3-10.

In either version of pTCP, when the puzzle mechanism was enabled, the attacker that made a large number of connections was overwhelmed with the computational burden of solving a large number of puzzles. Thus, in this attack the utilization of the CPU was decreased because the cost of the connection was shifted from the server to the attacker. Therefore, pTCP allowed the server to remain available to other clients in spite of the attack.

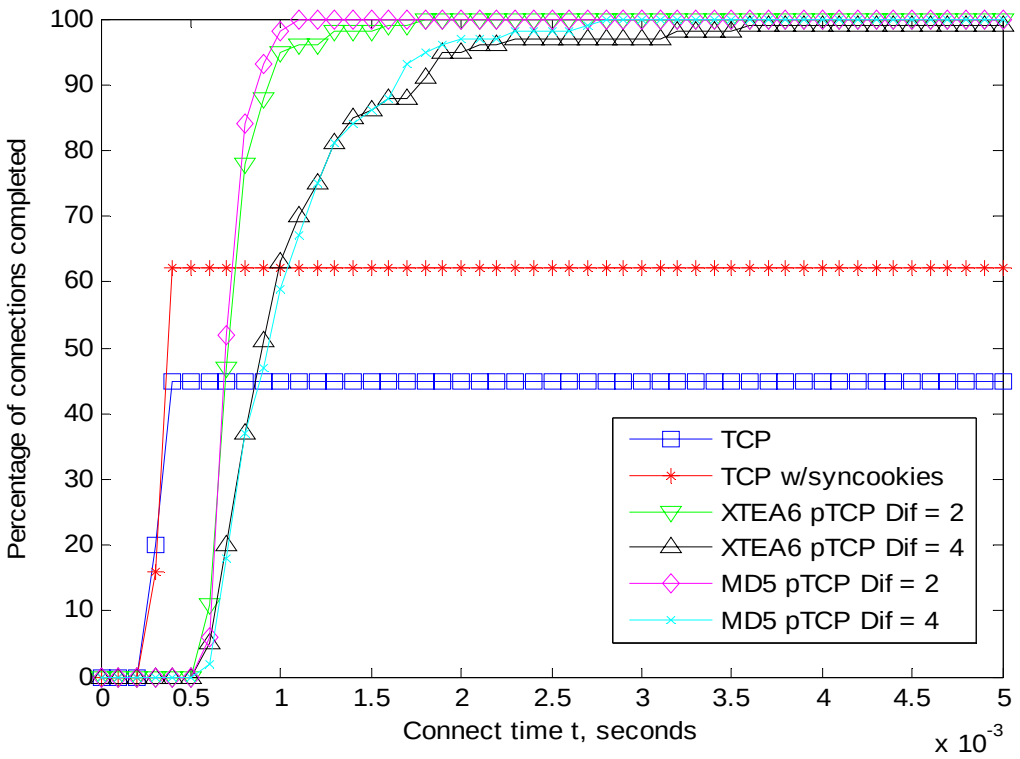

Figure 3-12: Percentage of connections completed versus connection times during a CPU Exhaustion attack

3.5. Deployment Scheme for pTCP

Deploying pTCP into the current Internet requires modification to both clients and servers. Unfortunately for all proposed implementations of TCP client puzzles, there does not exist any other way around modifying them that does not violate some of the key components to a client puzzle protocol discussed in Chapter 2. However, pTCP was designed with backwards compatibility in mind. When a server that is equipped with

pTCP and is not under an attack, a client that is not equipped with pTCP can still communicate with that server. Only during an attack and when a client does not have pTCP, that client will not be able to communicate with a server with pTCP.

3.6. Shortcomings of pTCP

pTCP was designed to handle specific resource-exhaustion attacks on the transport and application layers. Although it was not explicitly designed for the application layer, it does add a layer of protection for the application because the application layer relies on the transport layer. If weaknesses still exist, designing a client puzzle protocol at the application may solve some of the problems that can not be directly solved by pTCP. In these attacks there would not need to be a large number of connections or connection attempts, and the vulnerability is introduced by the application.

As the magnitude of the attack increases, the processing resources of the server or the bandwidth of the local network may become the target. In pTCP, we assumed that the server can process the incoming packets because we must assume that the server is able to send the Puzzle challenge and that each client can receive it. In a large attack, like a DDoS flooding attack, the server may not be able to process all of the incoming packets. In addition, the flooding of packets may result in a bandwidth consumption attack where the packets may never reach the server. While pTCP can defend against specific attacks at the transport layer, it does not provide a solid defense against large-scale flooding attacks. Since this attack is one of the most common attacks, we present another client puzzle protocol that is better suited to combat these attacks in the next chapter.

CHAPTER 4

4. IP Client Puzzles

Recently, deploying client puzzles at the IP layer, or IP client puzzles, has been proposed to mitigate flooding attacks on the network layer. IP client puzzles essentially act as a rate-limiter to malicious attackers that are flooding packets towards the victim. The common goal among all IP client puzzle protocols is to throttle the attackers so they do not overwhelm the victim. This chapter introduces the unique and novel design and simulation results of a network layer client puzzle protocol called Chained Puzzles. However, before introducing the specifics to Chained Puzzles, we first outline the assumptions in Table 4-1.

Table 4-1: Assumptions in Chained Puzzles

Assumption 1:	Attackers are generally more aggressive and send more packets than any legitimate client.
Assumption 2:	During an attack, there are fewer attackers than legitimate clients.

4.1. Technical Challenges in an IP Puzzle Scheme

In any kind of puzzle there are always two sides: the puzzle solver and the puzzle generator/verifier. In a connection-oriented protocol like TCP there are two ends to every connection: the client and the server. Thus, embedding client puzzles into TCP is relatively straightforward. However, in a connection-less protocol such as IP, the

realization of the two end points no longer exists. Thus, the framework for an IP puzzle protocol will need to be drastically different from a TCP client puzzle protocol.

An ideal approach is to allow each client to create their own puzzles (with some initial input from the puzzle generator) so that each consecutive puzzle is both unpredictable and difficult to solve by the client. Performing a three-way handshake for every puzzle will add a significant amount of communication overhead and could possibly lead to another DDoS attack itself. Therefore, the exchange of puzzle information should be kept to a minimum.

In an IP-layer client puzzle protocol, intermediate router(s) will be responsible for performing the puzzle generation and verification. Minimizing the computational and storage load on the router is of utmost concern. If this is not taken into consideration, an attacker could easily exhaust the storage capacity or the processing resources of the router. Therefore, the state information required by puzzles must be kept to a minimum and the puzzle verification should be done as close to line speed as possible so the router is not overwhelmed with this additional overhead and can still service other clients.

As discussed in Chapter 2, in the two papers that have proposed IP-layer puzzles, there have been two distinct approaches. Wang and Reiter [29] use two channels, where application data and puzzle data are placed in two separate packets. Feng et al. [28] employ one channel, where the data and puzzle information are kept together in one packet. In Chained Puzzles, we use one channel to avoid the free-riding problem discussed by Wang and Reiter in [29], but we introduce a unique method for puzzle creation by solution chaining that transforms a connection-oriented protocol into a connection-less for an ideal integration in the IP-layer.

4.2. The Client Puzzle for Chained Puzzles

The client puzzle used for Chained Puzzles is very similar to the client puzzle that was used in pTCP. It employs the modified block cipher encryption algorithm XTEA6,

which uses a 64-bit plaintext, a 128-bit key, and a 64-bit ciphertext. Since the traffic at the network layer could be TCP, UDP, or ICMP, the puzzle was redesigned to handle all types of traffic. In this puzzle, a 64-bit nonce called N_R is given to the client by a router is initially is used as the plaintext. In the next puzzle, the plaintext will be that client's previous puzzle solution. The key is comprised of a 32-bit hash of the client IP address, a 32-bit hash of the server IP address, and the 64-bit puzzle solution. By using a hash of the client and server IP addresses, we can support both IPv4 and IPv6. Similar to the puzzle from pTCP, the difficulty of the puzzle is controlled by the number of most significant bits that are equal to zero. Figure 4-1 shows a graphical representation of the client puzzle.

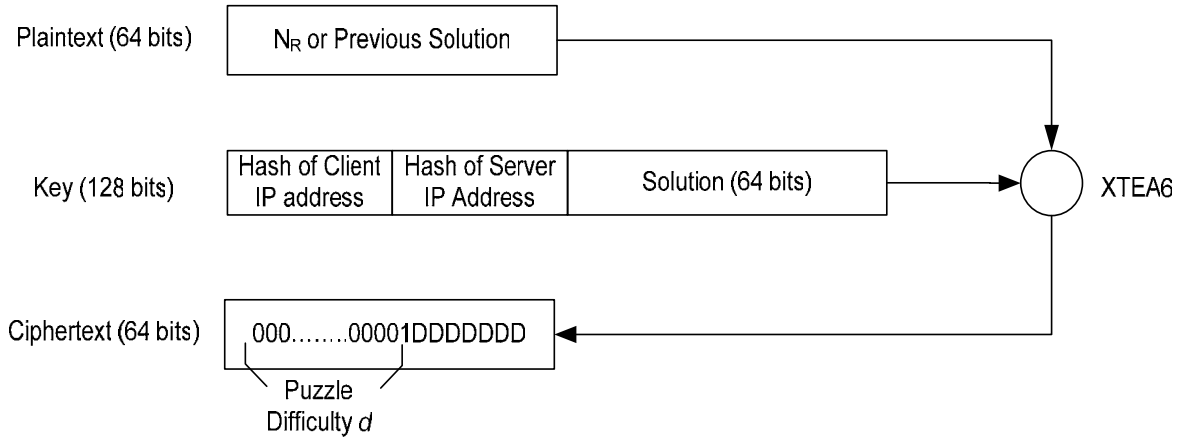

Figure 4-1: Client puzzle for Chained Puzzles

4.3. Chained Puzzles

4.3.1. Overview of Chained Puzzles

As stated earlier, DoS/DDoS mitigation is best performed as close to the source of the attack as possible to prevent a large amount of packets from converging onto the victim [6]. Therefore, puzzles should be generated and verified at the edge routers or border routers of an Internet Service Provider (ISP), rather than in the core of the Internet or at the routers closest to the victim. These routers are located closest to the source of the attack and are the optimal locations to perform the attack mitigation. In our scheme, we

are required to modify routers at each end of the network: the border routers near the source of the attack and the border routers near the victim of the attack. Thus, border routers will act as "puzzle servers". Throughout the remainder of this thesis, these routers are referred to as *puzzle routers*. Puzzle routers near the victim of the attack are simply designed to detect an ongoing attack, but can be used to generate and verify puzzles if the source of a different attack is within its own network. We assume that there exists a basic detection mechanism already in place that can determine if a flooding attack is underway. A puzzle router also exists near the source of the attack; this router is responsible for generating and verifying puzzles. Thus, there is a pair of puzzle routers at each end of the path that connects a client and a server. In order to handle puzzle generation and verification, the puzzle routers near the source is required to store state information. The puzzle router maintains a *Plaintext Table* for each client's flow, where each entry contains a 20-bit hash of the concatenated string of the client and server IP addresses, and either the initial router nonce, N_R, or the client's previous puzzle solution (both 64 bits). The clients all use N_R for the first puzzle, so initially the Plaintext Table is empty. After correct verification of the first puzzle for each client, an entry is created in the Plaintext Table. Thus, before the puzzle router stores state information for that client, the client must first solve one puzzle. This will help avoid attacks that target the storage capacity of the router.

Although there are a limited number of clients per puzzle router, to determine feasibility we must consider the amount of storage required in each puzzle router. For example, if each puzzle router serviced 10,000 clients with one flow each, then the size of the table would be 105 KB, which is small enough to be readily stored in the memory of any typical router. Existing IP-layer puzzle protocols require a puzzle-capable router to store a greater amount of state information. In [29], the authors use a bloom filter stored within the router to check for duplicate puzzle solutions from a large number of clients and estimate that the size of the filter would be 1.1 MB.

To introduce the specifics of our protocol, we have summarized the steps taken by the client and puzzle router in the event of an attack in Table 4-2.

Table 4-2: Overview of Chained Puzzles

Puzzle Router:

1. A puzzle router downstream near the victim detects the attack and sends an *ICMP Congestion Notification* upstream to other puzzle routers that may be forwarding packets from the attackers, which enables Chained Puzzles.
2. When puzzles are enabled, the puzzle router near the source sends an *ICMP Puzzle Challenge Packet* embedded with the initial router nonce, N_R, and the current difficulty level, d to every client serviced by that puzzle router. N_R is initially the same for every client and is refreshed for each client periodically through another ICMP Puzzle Challenge Packet.
3. Immediately after puzzles are enabled, the puzzle router will wait for the *Puzzle Transition Period* (PTP) before any new puzzles are verified unless a client sends a solution before that time expires (by IP Option 102). When that time expires, the puzzle router will begin to verify puzzles based on a certain probability. If the puzzle is correct or if it is not verified, the current solution is updated in Plaintext Table. If the puzzle is verified and the solution is incorrect, the packet is dropped.
4. The puzzle router may occasionally receive another ICMP Congestion Notification from downstream puzzle routers, which signifies to increase or decrease the difficulty level.
5. When N_R or d needs to be refreshed, the puzzle router sends all clients a new ICMP Puzzle Challenge Packet. After sending this packet, the router waits again for the Puzzle Transition Period before it verifies another puzzle. Before that time period has expired, it looks for the IP Option 102, which signifies the start of a new chain for that client.
6. The puzzle router may also receive an ICMP Congestion Notification to disable puzzles. The puzzle router then sends a different ICMP message back to each client to inform them to disable the puzzle mechanism.

Client:

1. If the client receives an ICMP Puzzle Challenge Packet from the puzzle router with the puzzle information for the first time, it solves the puzzle using the initial router nonce, N_R, and sends its next packet with the puzzle solution embedded into the IP options field of the header.
2. For every new packet, the client uses the previous puzzle solution as the plaintext for the next XTEA6 puzzle. The client solves this puzzle and embeds the solution into the IPv6 header of the packet, with the IP Option 102. Doing so, the client will be able to create a chain of puzzles. Every subsequent puzzle will be assigned the IP Option 101.
3. Periodically, a client may receive another ICMP Puzzle Challenge Packet from the puzzle router with either a new N_R or a different difficulty level. The client uses this new information to solve a new chain a puzzles. If this is the case, a new chain is being created and the client must mark this in the next packet by using the IP Option 102 to signify to the puzzle router that it is starting a new chain.
4. A client may also receive an ICMP message to stop solving puzzles. Following this message, the next packet is sent without a solution.

4.3.2. The Details of Chained Puzzles

In Chained Puzzles, when a puzzle router downstream detects congestion or heavy packet loss it will send an ICMP Congestion Notification message (of type 98) to each of the individual clients that it may suspect is involved in the attack. This packet will traverse its way to the client and will be intercepted by the puzzle router upstream. The puzzle router will see this notification and then begin forcing all clients to solve puzzles for every packet that it passes through the outbound links of the router. A similar ICMP message (of type 99) can be sent from downstream routers that signals that there is no more congestion and that puzzles no longer need to be solved. This detection method assumes that the IP addresses of the clients have not been spoofed. Since puzzle routers are located upstream they can easily perform the egress filtering operations required to combat IP spoofing within an ISP. Thus, a puzzle router will drop packets that do not have a correct IP address within the address range of its local network. Of course, an attacker may still spoof its IP address by using a valid one from within the local network. Concerns relating to this issue will be discussed later in Section 4.5.

When a puzzle upstream receives the ICMP Congestion Notification, it sends an ICMP Puzzle Challenge Packet to each of the clients in that network. This message contains the 64-bit initial router nonce, N_R, and the puzzle difficulty level, d. N_R is the same for each client. Each client takes this value, creates its own puzzle, as defined in Figure 4-1, and solves for the puzzle solution. For the first puzzle that is solved, the client inserts the solution in the IPv6 Hop-by-Hop Options field and gives that option the code 102. This code signifies that the client is starting a new chain of puzzles. For every following packet, the client uses the IP Option code 101. This option means that there is a puzzle solution attached, but that it belongs from the chain from the previous packet. This sequence of events is shown in Figure 4-2.

When the puzzle router enables puzzles, it waits for a certain amount of time before it begins to verify puzzles, called the Puzzle Transitional Period (PTP). This is because

packets may be in transit as puzzles are enabled. Those packets will not contain a solution and they should not be dropped because they were sent before the client was aware of the puzzle. The PTP is required to allow a client to receive the ICMP Puzzle Message, process it, solve the puzzle, and embed the solution in the next outgoing packet. When the router is waiting for this time period to expire, it is in *PTP Mode*.

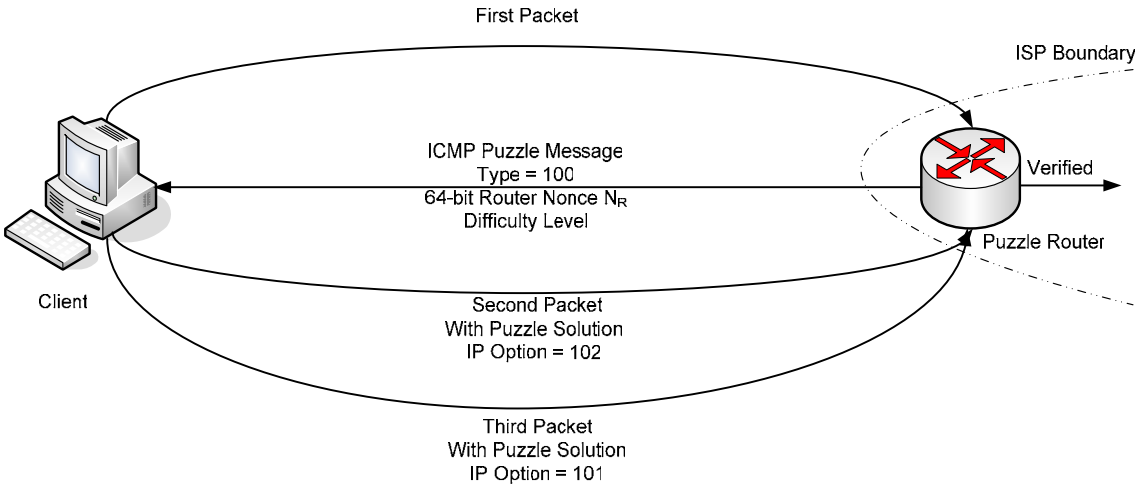

Figure 4-2: Initial client-router interaction

After enabling puzzles, the puzzle router will wait to verify puzzles until the PTP expires or until it receives the first solution from that client. During this time period, the client creates a new chain of puzzles and embeds the first solution in the IP Options field with the code number 102. The puzzle router detects this as the start of the new Plaintext Chain and verifies the solution. When the PTP expires, all puzzles are subject to verification.

Periodically, the puzzle router may wish to resynchronize the client with a new N_R, or increase or decrease the difficulty level of the puzzle. Whatever the case may be, this essentially resets the current chain of puzzles and forces the client to create a new one. The puzzle router will need to send another ICMP Puzzle Message to each client with the updated puzzle parameters. When this occurs, the puzzle router does not verify any puzzle until the PTP expires, or until it receives a puzzle solution (IP Option 102) from a particular client. This allows for a smooth transition to a new chain of puzzles.

The value for PTP needs to be carefully controlled to avoid giving an attacker a window of opportunity to flood packets. Thus, we can approximate the value for this time as shown in (4.1).

$$PTP = RTT_{C-PR} + \alpha \tag{4.1}$$

In this equation, the value of $RTT_{C\text{-}PR}$ is the round-trip time between the client and the puzzle router. This value can be approximated by each puzzle router. The value of α is added to account for any small processing delays observed by either end. This value will likely be determined through experiments with an actual implementation of Chained Puzzles.

The client is equipped with a *Puzzle Manager* software application that is responsible for interpreting the ICMP puzzle messages, solving a chain of puzzles, and then stamping each outgoing packet with the correct puzzle solution. Software such as this can be built into the operating system or possibly run as a standalone program that executes at a very low level. Wang and Reiter mention a similar application in [27]. They indicate that clients would be motivated to install this software if it acted as an incentive that clients could receive better performance during an attack.

In addition to the client-side software that needs to be modified, the firmware of the routers needs to be upgraded as well. The design of the puzzle router is slightly more complex, with the Plaintext Table, the PTP, and the handling of ICMP puzzle messages. A simple block diagram is shown in Figure 4-3 that depicts the actions taken by a puzzle router when a packet arrives.

When a new packet arrives at the puzzle router, it first determines if puzzles have been enabled by the downstream puzzle routers. If puzzles are disabled, the packet is placed in the output queue. If the puzzles are enabled, the router checks to see if it is in PTP Mode. When the puzzle router is in PTP Mode, it looks for an IP Option 102 to signal the start of a new chain. If the puzzle router discovers a packet with IP Option 102, it verifies the

puzzle with the most recent versions of N_R and d. If the packet does not contain a puzzle solution, it is forwarded by the puzzle router.

After the PTP has expired, the puzzle router will then need to verify every puzzle. First, it checks for a puzzle solution. If there is no solution, it drops the packet and sends an ICMP Puzzle Challenge packet with the current values of N_R and d. If the client supplied a solution in the packet, the puzzle router then checks to see if the chain needs to be reset by sending each client a new N_R or d. If the puzzle router determines that the chain needs to be reset, it sends the new puzzle information and enters the PTP Mode.

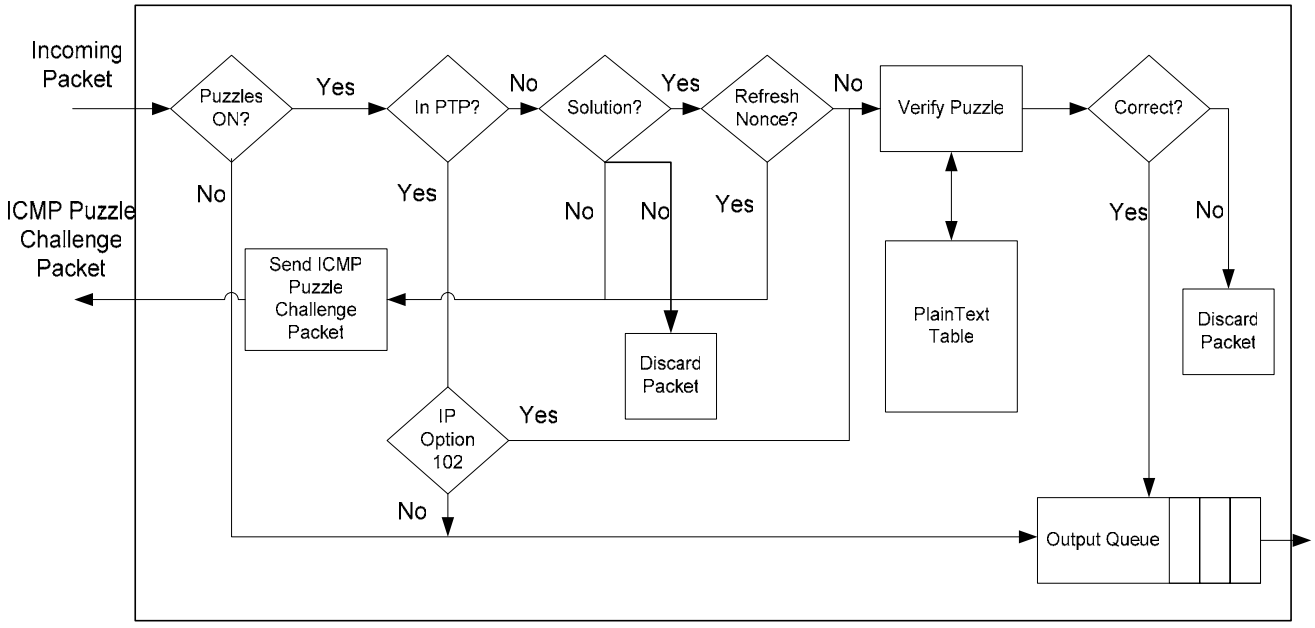

Figure 4-3: The forwarding process for incoming packets at a Puzzle Router

If the puzzle router does not reset the chain, it verifies the puzzle before forwarding the packet. If there is no entry in the Plaintext Table, N_R is selected as the plaintext in the XTEA6 puzzle. If that puzzle is correct, an entry is created in the Plaintext Table and the solution is placed in the table. This will help limit an attacker from filling up the Plaintext Table with false entries. Otherwise, if there is an entry in the Plaintext Table for that client, this value is used as the plaintext in the XTEA6 puzzle. If that puzzle solution is correct, the Plaintext chain is updated by placing the correct solution in the

Plaintext Table. Next, the packet is placed in the output queue to be forwarded by the router. If the puzzle solution is incorrect, an entry in the Plaintext Table is not created nor is the entry updated if one exists. Any packet with an incorrect solution will be dropped.

4.4. The Effectiveness of Chained Puzzles

A client puzzle at the IP layer is purposely designed to throttle any potential attackers when an attack is underway. In order to determine how effective Chained Puzzles would be we need to first examine the behavior of a typical attacker. The attacker, or the zombie, will on average make a larger number of requests than any single legitimate client. In a client puzzle protocol, the attacker has two realistic choices: to solve the puzzle or to supply a random solution and hope that it is correct. An attacker may alternate this behavior to create a more effective attack. Thus, we can define a *hybrid zombie* and the probability that it solves the puzzle as Q. A summary of all of the users in Chained Puzzles is shown in Table 4-3.

When the zombie solves a puzzle per packet, its sending rate is reduced significantly. The factor by which this rate is reduced is defined in (4.2). In this equation, k is the time it takes for the puzzle solver to execute the XTEA6 encryption function once and d is the difficulty level of the current puzzle. The function $S(k, d)$ is defined as the puzzle solve time.

Table 4-3: Classes of users in Chained Puzzles

Category	**Description of User**
Legitimate Client	A user that solves a puzzle for every packet.
Solving Zombie	An attacker that solves a puzzle for every packet.
Guessing Zombie	An attacker that guesses puzzle solutions.
Hybrid Zombie	An attacker that alternates between solving and guessing puzzle solutions.

$$S(k,d) = k \cdot 2^d \tag{4.2}$$

If we let t_p be the average processing time required to generate a regular packet, we can define the sending rate of a puzzle solver P in (4.3). In this equation, we assume that the time it takes to solve a puzzle will be much larger than the time it takes to generate a packet.

$$P = \frac{1}{t_p + S(k,d)} \cong \frac{1}{S(k,d)} \tag{4.3}$$

Thus, every puzzle solving user in Chained Puzzles will have the sending rate defined in (4.2). Unfortunately, this includes legitimate clients as well as the puzzle-solving attackers. When an attacker attempts to flood the network with packets, its sending rate will be severely reduced to the rate shown in (4.3). Thus, even if the attacker wants to send at a higher rate, it will be reduced to sending at a rate controlled by the difficulty level. Meanwhile, a legitimate client will notice a similar reduction in their sending rate, but it will not be quite as significant. In fact, if the client is sending at a slower rate than the rate shown in (4.3), it will not notice any delay in sending data. The only difference it may notice is the change in CPU utilization due to the puzzles.

If we let t_r be the average time it takes to process a packet by the router, the router's service rate R will be reduced as shown in (4.4). In this equation, k is again defined as the time it takes the puzzle router to execute XTEA6 once.

$$R = \frac{1}{t_r + k} \tag{4.4}$$

If the zombie does not solve a puzzle and supplies a random solution to a puzzle, its sending rate is not reduced. Therefore, the probability of the packet reaching the victim depends on the probability that an attacker solves the puzzle correctly and the probability that the attacker does not solve the puzzle, but guesses the correct solution. Correctly

guessing a solution depends solely on the difficulty level of the puzzle. With the puzzle difficulty of d, the attacker has on average probability 2^{-d} of guessing a correct puzzle solution. Thus, we can define the probability of an attacker having a single packet sent downstream (p_D) to the victim, which is shown in (4.5).

$$p_D = (1-Q)2^{-d} + Q = 2^{-d} + Q(1-2^{-d}) \tag{4.5}$$

From (4.5), when the difficulty level of the puzzle is low, regardless of the value of Q, the probability of an attacker's packet reaching the victim approaches one. Thus, it is in the best interest of the puzzle router to keep the difficulty level sufficiently high. However, when Q is zero, the probability of the packet reaching the victim solely depends on the difficulty level. Thus, if the difficulty level is high, it is in the best interest of the attacker to solve the puzzle. Conversely, if the difficulty is low, an attacker can modulate Q to create a more effective attack.

4.5. Security Concerns of Chained Puzzles

An important issue in Chained Puzzles is to prevent the protocol from becoming the source a new attack. In other words, this protocol should be resilient against future DoS attacks that may exist due to the newly implemented countermeasure. In Chained Puzzles, there are a few areas for concern that need to be addressed. These areas are the authentication of ICMP Congestion Notification messages sent to puzzle routers to enable or disable puzzles, the authentication of ICMP Puzzle Challenge messages sent from puzzle routers to the clients to begin or stop solving puzzles, and the nonce synchronization between the client and the router.

In our scheme, we rely on the puzzle routers downstream to be able to detect heavy congestion or packet loss and send the ICMP Congestion Notifications upstream to the other puzzle routers. For our scheme to work properly, these messages need to be authenticated. An attacker could easily spoof a router's identity and send ICMP Congestion Notification messages to several puzzle routers. This could cause puzzle

routers to enable puzzles, when it is not needed, and would decrease throughput for all clients beneath that puzzle router. In addition to the congestion notification messages, routers downstream can also send messages telling the puzzle routers to disable puzzles. If an attacker could forge this message then it could bypass the client puzzle mechanism altogether and proceed with its DDoS attack. Thus, control messages sent between puzzle routers need to be authenticated.

In a DDoS attack, an attacker has the ability to spoof the IP address of the zombies used in the attack, thus making it is difficult to discover the real source of the attack. In Chained Puzzles, the congestion notification messages need to be sent to the correct puzzle routers. If an attacker spoofs the IP address of its packets, then the congestion notification messages may be sent to the wrong router and its clients, the ones with the IP addresses that were spoofed. Therefore, we need to ensure that when a puzzle router downstream detects congestion it sends an ICMP message to the correct router. One way to solve this is to have each puzzle router upstream mark its IP address into the packet being sent downstream. When a router downstream detects congestion, it will know the correct location of the puzzle router for each packet it is receiving and then be able to successfully send the congestion notification message. However, as ingress and egress filtering are being implemented in networks more often, this will become less of a problem. This filtering process can even be built into the puzzle routers that forward the traffic.

In addition to the ICMP messages sent to the puzzle router, the puzzle router itself sends ICMP Puzzle Challenge messages to each client to notify them that they need to begin (or stop) solving puzzles. If an attacker can forge this message, it can send the same message to a client and force the client to solve puzzles when it is not needed. Thus, control messages sent to clients from puzzle routers also need to be authenticated. An authentication method for this can be done somewhat easily if we use a technique described in [16]. Since the puzzle router and the client are one-hop away from each other, the TTL value can be set to 255 when the puzzle router sends a packet to the client. Thus, the client will know that this message came from the puzzle router by examining

the TTL field. An attacker could not forge this message because the TTL would be decremented as it traversed through the network.

When puzzles are enabled, each client and the router are synchronized with a plaintext value. Originally this value is N_R, and afterwards it is the previous puzzle solution for that client. This value is used for the next packet because it represents the next plaintext to be used for the next puzzle. The router updates this value after forwarding each packet and the client updates this value after sending each packet. If the Plaintext Chain is broken, the legitimate client could be denied access for its subsequent packets, which causes another unique DoS attack. If the difficulty level is low, an attacker could guess a puzzle solution and spoof its IP address of one belonging to a real client within the local network and use the same destination IP address as that client, it would cause the plaintext value to change in the puzzle router's Plaintext Table. It is possible to spoof an IP address of another client on the same subnet. Recall that this is one of the weaknesses in ingress filtering. We call this attack a *Spoofed IP Guessing Attack*; similar to the *Guessing Attack* mentioned in Section 4.3, except it employs intelligent IP spoofing (i.e. it knows the correct range of source IP addresses to spoof and the correct destination IP address). One way to combat this is to ensure that the difficulty level is high enough to prevent an attacker from guessing puzzle solutions on behalf of another client. An attacker can guess a correct puzzle solution with probability of 2^{-d}; for example, if the difficulty level was set to five, the probability of an attacker guessing a correct solution is 1/32.

An extension to Chained Puzzles could be to utilize Probabilistic Puzzle Verification [29], where puzzles are verified at the router with a certain probability. This helps improve the processing resources of the router, but it may also give an advantage to an attacker. If Chained Puzzles were to employ Probabilistic Puzzle Verification, the Plaintext Chain can be easily broken when the puzzle router does not verify a puzzle solution. For example, if the router is only verifying 80 percent of the puzzles, an attacker can spoof an IP address of a real client within the local subnet, select the same destination IP address that the client is sending data towards, and send a random solution.

If the puzzle is not verified, the solution is updated in the Plaintext Table. Thus, the attacker has a 20 percent chance of breaking the chain between that client and the puzzle router. When the router verifies the next puzzle for the "real client", the solution will likely be incorrect. Therefore, a DoS attack can occur by targeting a single client rather than targeting the server. The impact of this attack can still have negative and averse effects if it is widespread. If an attacker recruited zombies spread out across the Internet, then began to deny each individual client access, it would still result in a large-scale DDoS attack.

When the Plaintext Chain has been broken, it will result in a sequence of incorrect puzzles for that client. One potential solution to this problem may be to resynchronize that client with a new N_R following the receipt of a sequence of incorrect puzzles. Of course, we must consider that an attacker could spoof the IP address of a client and then send incorrect random solutions as fast as possible, causing the client to be continuously resynchronized with the puzzle router. An attack such as this would be just as effective as a flooding attack because clients would not be allowed to access the server. Thus, the resynchronization of the Plaintext Chain should be carefully controlled.

4.6. Implementation Details of Chained Puzzles

Chained Puzzles was simulated using the Network Simulator NS-2 [43]. The NS-2 simulator code was modified to support the simulation experiments. The code that was modified was located within the address classifier object of an NS-2 node. With standard NS-2, when a client creates and packet and sends it to a server, it goes from the Application Agent, to the Transport Layer Agent, and then into the address classifier of the source node. Typically, the packet is destined for another node, so when the packet reaches the address classifier it is sent out onto the link to the next hop. If the destination of the packet matches the current node, it is sent to a port classifier and then to the Transport Layer Agent. A layout of an NS-2 node can be seen in Figure 4-4.

The address classifier was designed in such a way to store packets in a queue before sending them out onto the link. If the node was generating the packet, it would delay the transmission of the packet until waiting for *S(k, d)* seconds. Thus, with the legitimate client, each packet was delayed by that time before it was sent out on to the link. In the case of an attacker, the attacker would solve the puzzle with a given probability *Q* and delay the transmission of the packet. With probability *1-Q*, the attacker did not solve the puzzle and the packet was immediately sent to the next hop.

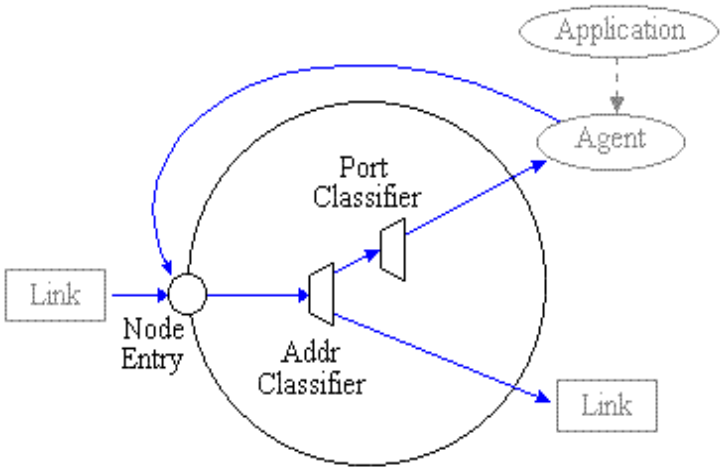

Figure 4-4: NS-2 node

The puzzle router is the "first-hop" router that receives the packet from the legitimate client or the attacker. Upon receiving that packet, the puzzle is verified and the router will delay sending the packet to the next hop for *S(k, 0)* seconds, which is equal to *k*. If either the client or the attacker did solve the puzzle, the packet is sent to the next hop after *k* seconds. If the attacker did not solve the puzzle, the packet has a probability of 2^{-d} that it will be sent to the next hop. Thus, when the difficulty level is sufficiently high enough, the packet will most likely be dropped. Regardless of the puzzle solution being correct or incorrect, the puzzle router will need to wait for *k* seconds before it can process the next packet because time was spent verifying the solution.

NS-2 was useful for simulating basic networks with a fair amount of traffic. However, as the size of the network increased and the amount of traffic increased, the time required to

run the simulation and gather results afterwards became too long. Thus, we began to experiment with OPNET to see if these results could be improved and if it were possible to simulate more complex and larger networks.

In addition to the modification of NS-2, Chained Puzzles was also simulated using OPNET Modeler [44]. OPNET was primarily used to simulate networks with a larger number of legitimate clients and attackers, which allowed us to determine the scalability of Chained Puzzles. Modifications were made, very similar to ones discussed with NS-2, to each client and first-hop router to emulate the puzzle mechanism. Simulations in OPNET proved to be more efficient because it decreased the amount of processing time required with NS-2, especially for very large networks.

Thus, with modifications to both simulators we can accurately simulate how a network would operate during a DDoS flooding attack with Chained Puzzles. The next step is to design a network and create a sufficient DDoS attack while using the standard IP and show how the conditions could be significantly improved by using Chained Puzzles. In the next section, the simulation results from these experiments are presented.

4.7. Simulation Results

To further investigate how effective Chained Puzzles would be in the event of a DDoS flooding attack, we first simulated such an attack using the modified and unmodified versions of NS-2. In our simulation, a tree network was created with 100 clients and 40 attackers. Because the network size was fairly small, we scaled the sending rates of the attackers and clients, along with capacity of each link. The bandwidth of the victim was less than the total combined bandwidth of all the legitimate clients and attackers, which will allow us to create a DDoS flooding attack. The attackers sent UDP traffic at a rate of 10,000 kbps, which was the maximum rate allowed by the attacker's link. The legitimate clients sent UDP traffic at a rate of 1,000 kbps. We randomly set the time clients began sending data, and synchronized the attackers to begin sending data at the same time. In

this topology, the attackers are spread out across the network, which means that every legitimate client is forced to solve puzzles.

In Figure 4-5, we have shown the Normal Packet Survival Ratio (NPSR), a measurement that has been described in previous literature [6], for the legitimate clients in this experiment. The NPSR simply means the number of legitimate packets that actually reach the victim divided by the total number of packets that were generated by the legitimate clients. In our simulations, we varied the value of *Q*, the probability that an attacker solves the puzzle. As the difficulty level increases, the congestion from the attack is alleviated because the number of dropped packets belonging to the clients approaches zero, because the NPSR reaches one.

In this experiment, it appeared from Figure 4-5 that level ten was optimal to defend against the attack in this network. At this level, there were very few packets being dropped and the client's throughput remained fairly high. With standard IP, there was a 77% packet loss for the legitimate clients. With Chained Puzzles at difficulty level ten, there was less than 1% packet loss. However, at higher difficulty levels the client's throughput dropped significantly. Figure 4-6 shows the total number of legitimate packets generated during the simulation. It shows that when the difficulty level had reached eleven or twelve, the sending rate of the client had reduced significantly.

During an attack, the difficulty level should be adjusted by the downstream routers (through more ICMP Congestion Notification packets) to alleviate the congestion. When the downstream puzzle routers are still detecting congestion even after puzzles are enabled, it can send an ICMP Congestion Notification that signals to the upstream puzzle routers to increase the difficulty level. Conversely, when there is no congestion, a puzzle router can send a similar message to decrease the difficulty level. This can be repeated until the puzzle mechanism is disabled. Thus, the puzzle will only be as difficult as necessary to sustain the availability of the server. In this network, with a difficult level of ten, the legitimate clients received better service when compared to standard IP, despite the small decrease in their sending rate.

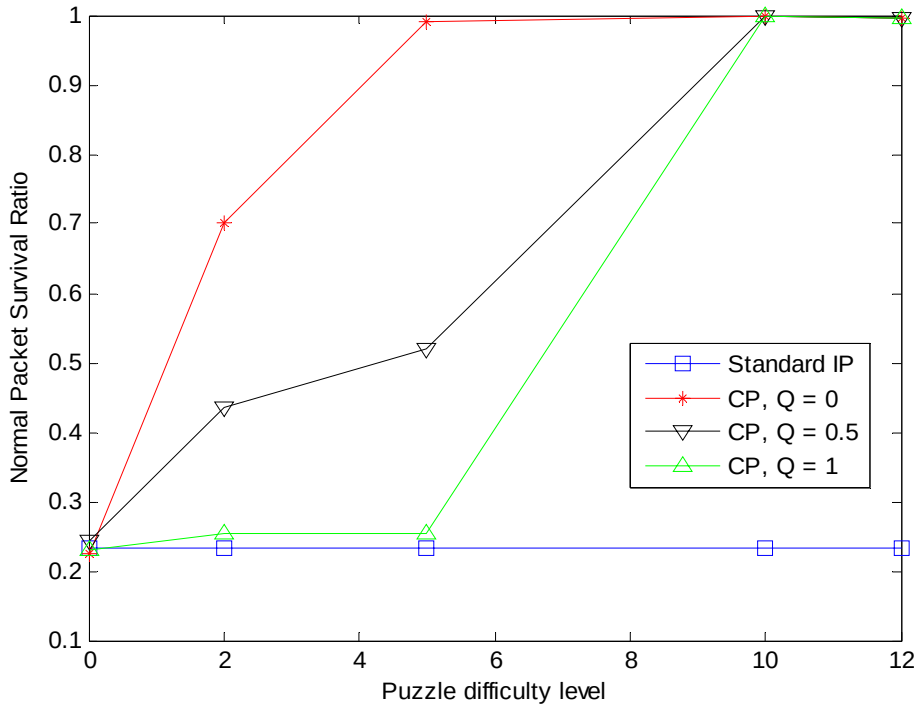

Figure 4-5: Normal Packet Survival Ratio for legitimate clients

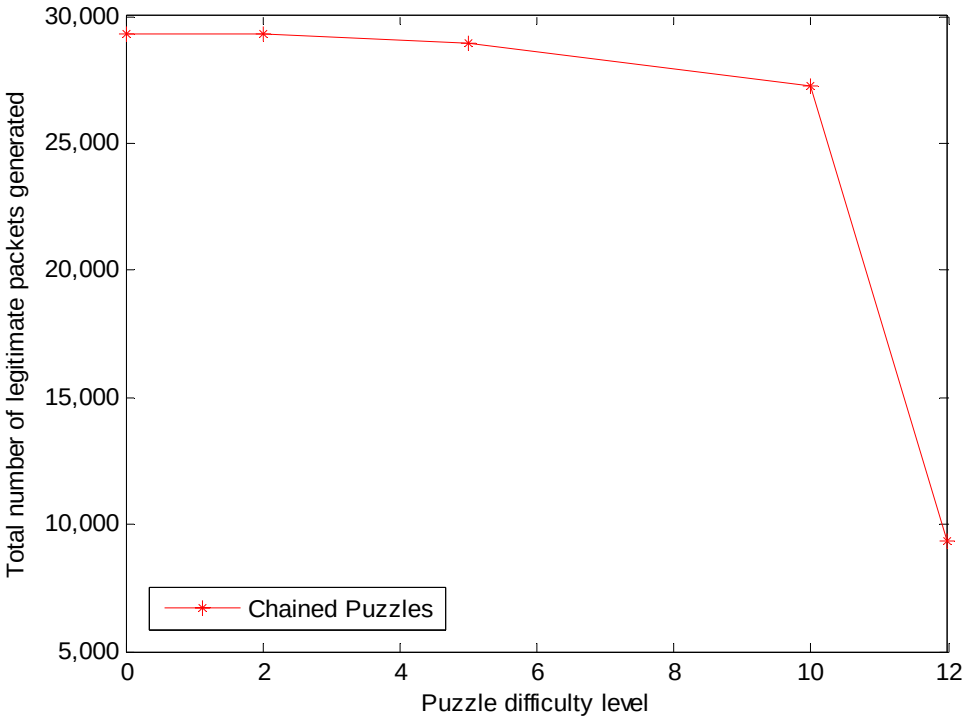

Figure 4-6: Total number of legitimate packets generated

For our next experiment, we utilized the implementation in OPNET to simulate a much larger network. This network consisted of 735 legitimate clients and 294 attackers (zombies). Again, the topology resembled a tree because the clients and zombies were the leaf nodes of the tree and the victim server was the root of the tree. The topology in this attack is considered to be a worst-case scenario, because the zombies are spread out across the network, meaning that every legitimate client is forced to solve puzzles. The bandwidth and sending rates for this experiment are the exact same as the previous experiment, only the size of the network is different. In this experiment, the NPSR was calculated versus the puzzle difficulty level and the number of legitimate client packets that were generated was recorded. The results of this experiment are shown in Figure 4-7 and Figure 4-8. When comparing the results from this experiment with the previous one, the difficulty level where the NPSR value reached one is much larger with a bigger network. In the larger network, a difficulty level of 16 alleviated the congestion near the victim. This was expected because the attack was considerably stronger. With standard IP, the NPSR was less than 0.02, which means that 98% of the legitimate packets were dropped. However, achieving an NPSR value equal to one did come with a cost. Figure 4-8 shows the total number of packets that were generated by legitimate clients. With a very high difficulty level, the total number of packets generated by legitimate clients was reduced significantly.

Thus, in Chained Puzzles there is a tradeoff between the throughput of the legitimate clients and the number of its packets that are dropped. When the client is not in the same subnet as the zombie, its sending rate would not be reduced due to the absence of the puzzle mechanism. However, when legitimate clients are in the same subnet as a zombie, their sending rate may be reduced. Despite this observation, Chained Puzzles still outperformed standard IP because the NPSR value was increased. However, the issue of the tradeoff mentioned between these two items still requires investigation and is discussed in more detail in Chapter 5.

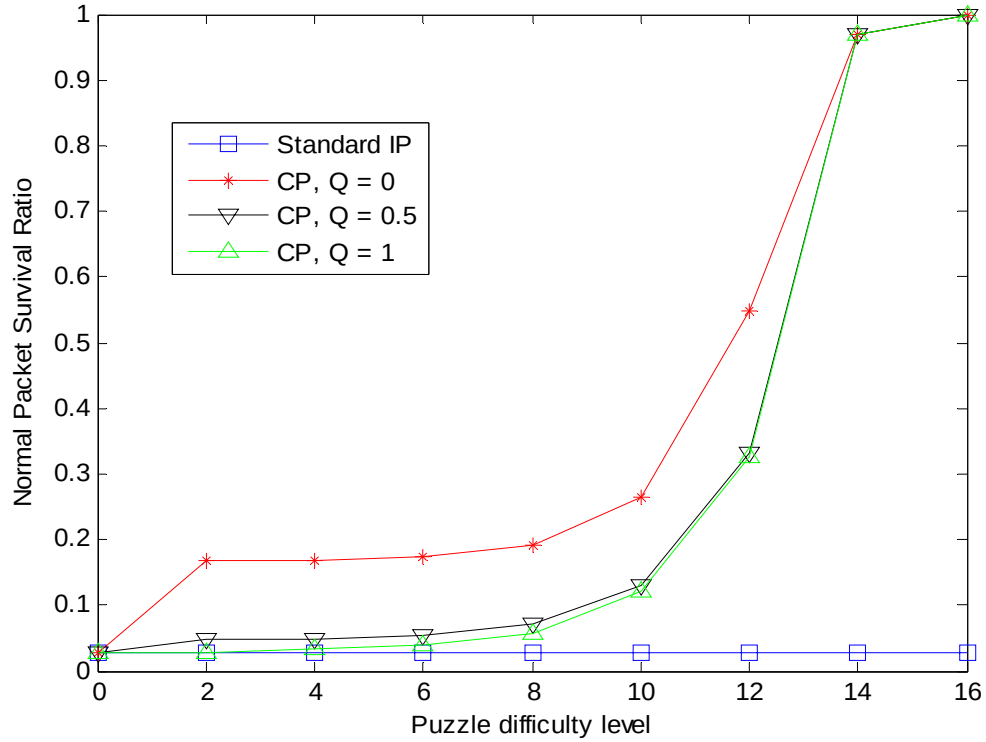

Figure 4-7: Normal Packet Survival Ratio for legitimate clients (Larger Network)

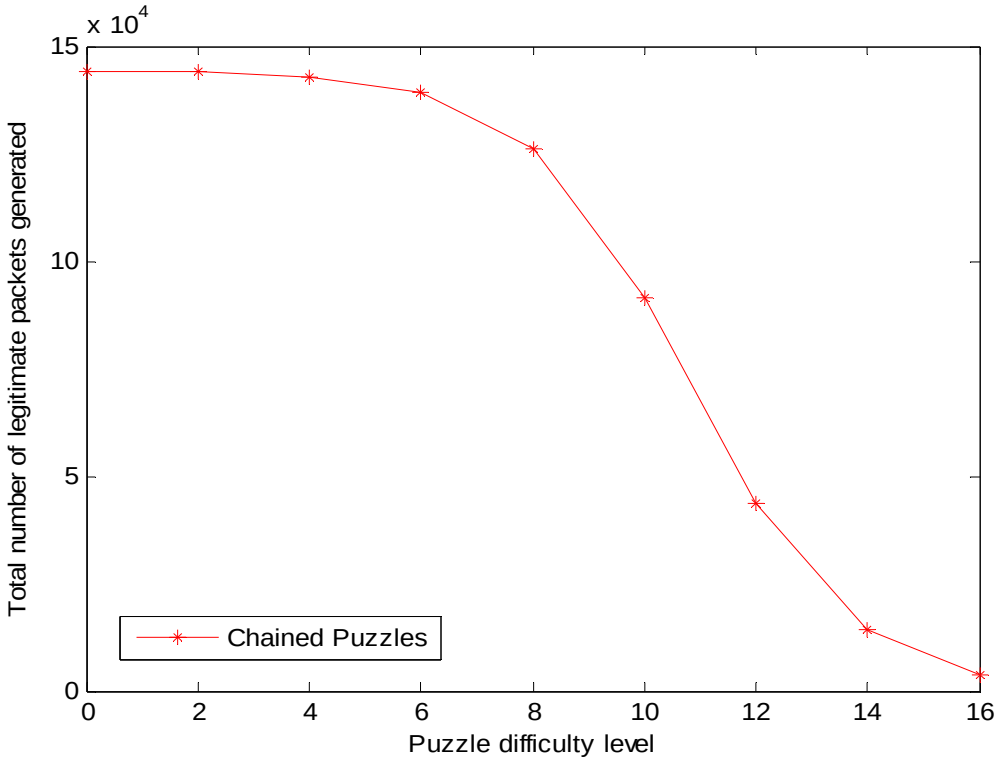

Figure 4-8: Total number of legitimate packets generated (Larger Network)

4.8. Deployment Scheme for Chained Puzzles

In order to deploy Chained Puzzles in the Internet, a pair of edge routers within ISPs or organizations needs to be upgraded to support the puzzle mechanism. Also, each client will need to install the Puzzle Manager software that will allow the client to support the puzzle mechanism. Wang and Reiter in [29] discuss the incentives for clients to install the software needed for puzzles. They claim that all Internet users would want this software if it meant that they would receive better service during an attack. Installing software on clients seems feasible if it is presented in that manner, but one of the main challenges that remains is to modify the edge routers throughout the Internet. If there exists an entry point into the network that does not pass through a puzzle router, an attacker will pursue clients in this network for their attack because they would not be affected by the puzzle mechanism. The downside to this scheme, and all client puzzle protocol schemes, is the need to modify existing devices in the Internet. However, Chained Puzzles requires only slight modifications. Changes are not required to any of the core Internet routers that link every system together. Thus, we believe that it is possible to implement Chained Puzzles in the Internet since the change required is minimal when compared to some of the other previously mentioned schemes. Since there needs to be a puzzle router upstream and downstream, companies or ISPs can install them together in pairs. Since puzzles are only deployed during an attack, it allows for gradual deployment. However, similar to pTCP, if a client cannot solve a puzzle, it will suffer during an attack.

CHAPTER 5

5. Conclusions and Future Work

In this thesis, various mitigation techniques for network-based DoS and DDoS attacks have been researched and studied. In particular, the topic of client puzzles has been researched more in-depth and has been applied at two different layers of the Internet stack: the transport and network layer. In recent literature, the concept of client puzzles has been explored in detail and has shown promise in mitigating the effects of an attack. This thesis has furthered the research performed in this area by designing, implementing, and experimenting with two different novel client puzzle protocols.

This thesis has presented the design and implementation details of pTCP, a client puzzle protocol that was integrated within the current TCP stack in Linux. The implementation allowed for realistic experiments with actual systems and in typical attack scenarios. By using pTCP, advantages were shown over standard TCP and other types of puzzle protocols during certain resource-exhaustion attacks. Experiments with pTCP were successful because they demonstrated that pTCP was capable of preserving the availability of the server and allowed clients to access the victim in spite of the attack.

This thesis has also presented the design and simulation of Chained Puzzles, a novel client puzzle protocol integrated into the network layer. The primary goal of Chained Puzzles was to defend a server against large-scale DDoS flooding attacks. By simulating this protocol with NS-2 and OPNET, the effects from DDoS attacks were alleviated and

the server remained available to legitimate clients when an attack was underway. In contrast, with standard IP the server was unavailable to clients because of the severe congestion.

Mitigating DoS attacks has always been a challenging and ongoing research topic in network security. The research that has been presented in this thesis could always be furthered to make improvements or new discoveries. The following sections present a summary of future work with pTCP, Chained Puzzles, and the general design of client puzzle protocols.

5.1. Future Work with pTCP

The solve time for a computational puzzle will always depend upon the processing power of the computer that is solving the puzzle. In such a scheme, a more powerful client will always have the upper hand over less powerful clients. This drawback is shared by most puzzle schemes, including pTCP. As part of our future work, we plan to explore methods that are able to adjust the level of difficulty for each potential client. By "customizing" the difficulty level for each client, we can prevent more powerful clients (possibly malicious ones) from having an unfair advantage [30]. This concept is known as fairness. The notion of fairness is an important research issue in client puzzles, because more effective protection against DoS attacks can be provided by distributing harder puzzles to clients that demonstrate attacker-like behavior. In all client puzzle protocols, the puzzles may become too difficult to solve for weaker legitimate clients and could create a DoS in return, causing weaker clients to spend more time solving a puzzle. pTCP could be extended if it could be designed to distribute puzzles of varying difficulty levels based on a reputation from that client.

Another area for future work with pTCP would be to research how well it performs with various applications. As stated in Chapter 2, puzzles have also been proposed to combat spam mail. Since pTCP places a cost on the connection for each client, it is conceivable

that pTCP could be used to limit the effects of spam mail. In addition to spam mail, there are several other application layer vulnerabilities that could be protected with pTCP. The work that is presented in this thesis could be extended by different experiments with the protocol and different applications.

5.2. Future Work with Chained Puzzles

This thesis has presented a novel framework for IP layer client puzzles called Chained Puzzles and has presented simulation results during a DDoS flooding attack. However, there is much more work that could be done with this protocol to the test its scalability. Simulations can be furthered by simulating this protocol in a realistic network with many more attackers and legitimate clients. Simulations such as this would greatly enhance the results that were gathered. In addition, an implementation of Chained Puzzles within a small network testbed would allow further consideration of certain issues such as the maximum size of the Plaintext Table, the length of the PTP, transitioning issues of the resynchronization of the Puzzle Chain, and determining the time between nonce resynchronization.

As stated before with pTCP, an issue of concern in Chained Puzzles is if the difficulty level of the puzzles can become too high for legitimate clients to solve. For instance, is having a NPSR value close to one considered an improvement if the throughput of the client is reduced significantly. DoS attacks do not only involve dropping packets, they can be just as effective when they degrade the service by significantly reducing the client's throughput. Improvements can certainly be made if the issue of fairness was considered. In Chained Puzzles, fairness needs to be researched and investigated thoroughly, especially for larger networks, so that Chained Puzzles can still maintain the availability of the victim while still granting better service to the legitimate clients.

5.3. Future Work with Client Puzzles

In all client puzzle protocols to date, the solve time of a puzzle is always relative to the processing power of the client's CPU. Future work with client puzzles will most likely involve designing new and different types of puzzles. As mentioned earlier in Chapter 2, memory-bound puzzles have been proposed as a better approach than the traditional computation-based puzzles. A claim with memory-bound puzzles is that the solve time is not as widely distributed as computational puzzles among computers of varying power. In other words, having more memory may not help improve the chances of solving the puzzle faster, when compared to a weaker to a client with less memory. An interesting research topic would be to modify the two schemes presented in this thesis with memory-bound puzzles and then simulate attacks with both attackers and clients of varying strengths to determine if memory-bound puzzles would make a significant improvement.

Another way to improve the design of a client puzzle is to have the difficulty level increase the hardness of the puzzle in a linear fashion. The puzzles based on XTEA6 that have been introduced grow exponentially difficult with a linearly increasing difficulty level. This is evident from the results shown in Figure 3-10 in Chapter 3. Having a linearly-difficult puzzle would allow for a broader range of difficulty levels that could be used before the puzzle became too difficult to solve. One of the most interesting methods to improve puzzle design is to investigate time-lock puzzles that were originally presented by Rivest [45]. A time-lock puzzle would allow for more accurate and difficult puzzles and the server would not have to consider the processing power or the amount of memory of any given client. However, the scheme presented in this paper requires the expensive generation of two large distinct prime numbers. At the present time, there is no known time-based cryptosystem that is efficient enough to be used in a client puzzle protocol. However, investigating a new kind of puzzle that does not rely on making assumptions about the power of the adversary is a topic of future work for all client puzzle protocols.

5.4. Conclusions

Client puzzles is a novel method to defend against DoS attacks, but their scalability is an issue that would greatly benefit from further research. If an attacker has a large amount of resources and computing power by possessing a large number of zombies, the effectiveness of client puzzles may not be as great as originally expected. However, as this thesis has shown, client puzzle protocols did improve the resiliency of a server during certain types of attacks. The idea behind client puzzle protocols, or a proof-of-work protocol, is a promising topic that may lead to improvements and advances in computer networking and security in the future.

As mentioned in the first chapter, mitigation is only one facet of a DoS countermeasure. Defeating DoS attacks involves studying various vulnerabilities in computer networks and designing a number of mechanisms that can prevent all of these problems from surfacing. Although, the real key to defending systems from network-based attacks is constant vigilance and awareness. Devising countermeasures will always help thwart known attacks, but since DoS attacks are only limited by the imagination of the attacker, combating DoS attacks requires an in-depth knowledge and a technical understanding of the fundamental problems in computer and network security. In addition to the mitigation techniques that have been presented in this thesis, defeating DoS attacks also involves preventing machines from being converted to handlers or zombies. It requires the accurate detection of attacks, and then having an efficient methodology to perform forensics following an attack. The combination of these methods will prove to be the most effective means to combat the serious threat of DoS attacks and to allow the Internet to become more reliable, useful, and secure.

REFERENCES

[1] M. Richtel. "Yahoo Attributes a Lengthy Service Failure to an Attack," in NY Times Online Article available at: http://www.nytimes.com/library/tech/00/02/biztech/articles/08yahoo.html, February 8, 2000.

[2] D. McGuire and B. Krebs. "Attack on Internet Called Largest Ever," in Washington Post Online Article available at: http://www.washingtonpost.com/ac2/wp-dyn?pagename=article&contentId=A828-2002Oct22&notFound=true, October 22, 2002.

[3] E. Skoudis. *CounterHack. A Step-by-Step Guide to Computer Attacks and Effective Defenses.* Prentice Hall, Upper Saddle River, NJ, 2002.

[4] A. Belenky and N. Ansari. "On IP Traceback," in *IEEE Communications Magazine.* Vol. 41, Issue 7, July 2003, pp. 142-153.

[5] J. F. Kurose and K. W. Ross. *Computer Networking. A Top-Down Approach Featuring the Internet, 2nd Edition*. Addison Wesley, Boston, MA, 2003.

[6] R. K. C. Chang. "Defending against flooding-based distributed denial-of-service attacks: a tutorial," in *IEEE Communications Magazine*. Vol. 40, Issue 10, October 2002, pp. 42-51.

[7] P. Ferguson and D. Senie. "RFC 2267 – Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing." January 1998.

[8] J. Postel. "RFC 793: Transmission Control Protocol," September 1981.

[9] CERT® Advisory CA-1999-17 Denial-of-Service Tools. Available at: http://www.cert.org/advisories/CA-1999-17.html. December 1999.

[10] Packet Storm Security. Online security reference website available at: http://packetstormsecurity.org/

[11] J. Lemon. "Resisting SYN flood DoS attacks with a SYN cache", in *Proceedings of USENIX BSDCon,* February 11-14, 2002, pp. 89-98.

[12] D. J. Bernstein. "SYN cookies" Online Reference Available at: http://cr.yp.to/syncookies.html

[13] C. Jin, H. Wang, and K. G. Shin. "Hop-Count Filtering: An Effective Defense Against Spoofed DDoS Traffic," in *ACM Conference on Computer and Communications Security (CCS'03)*, October 27-31, 2003. pp. 30-41.

[14] A. Yaar, A. Perrig, and D. Song. "StackPi: New Packet Marking and Filtering Mechanisms for DDoS and IP Spoofing Defense," CMU Technical Report, February 2003.

[15] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. "Controlling High Bandwidth Aggregates in the Network," in *ACM SIGCOMM Computer Communications Review*, Vol. 32, No. 3, July 2002, pp. 62-73.

[16] J. Ioannidis and S. M. Bellovin. "Implementing Pushback: Router-Based Defense Against DDoS Attacks," in *Proceedings of Networks and Distributed Systems Security (NDSS'02)*, February 6-8, 2002.

[17] A. D. Keromytis, V. Misra, and D. Rubenstein. "SOS: An Architecture for Mitigating DDoS Attacks," in *IEEE Journal on Selected Areas in Communications,* Vol. 22, Issue 1, January 2004, pp. 176-188.

[18] D. G. Andersen. "Mayday: Distributed Filtering for Internet Services," in *4th USENIX Symposium on Internet Technologies and Systems (USITS)*, March 26-28, 2003, pp. 31-42.

[19] K. Lakshminarayanan, D. Adkins, A. Perrig, and I. Stoica. "Taming IP Flooding attacks," in *ACM SIGCOMM Computer Communication Review (Papers from HotNets-II)*, Vol. 34, Issue 1, January 2004, pp. 45 – 50.

[20] R. C. Merkle. "Secure Communications Over Insecure Channels," in *Communications of the ACM*, Vol. 21, Issue 4, April 1978, pp. 294-299.

[21] W. Diffie and M. E. Hellman. "New Directions in Cryptography." *IEEE Transactions on Information Theory,* Vol. 22, No. 6, November 1976, pp. 644-654.

[22] A. Juels and J. Brainard. "Client Puzzles: A cryptographic defense against connection depletion attacks," in *Proceedings of Networks and Distributed Systems Security (NDSS '99),* February 3-5, 1999, pp. 151-165.

[23] T. Aura, P. Nikander, and J. Leiwo. "DoS-Resistant Authentication with Client Puzzles," in *Lecture Notes in Computer Science,* Vol. 2133, 2000, pp. 170-177.

[24] X. Wang and M. K. Reiter. "Defending Against Denial-of-Service Attacks with Puzzle Auctions (Extended Abstract)," in *Proceedings of the 2003 IEEE Symposium on Security and Privacy,* May 11-14, 2003, pp. 78-92.

[25] B. Bencsath, I. Vajda, and L. Buttyan. "A Game Based Analysis of the Client Puzzle Approach to Defend Against DoS Attacks," in *Proceedings of SoftCOM 2003, International Conference on Software, Telecommunications and Computer Networks,* October 7-19, 2003, pp. 763-767.

[26] D. Dean and A. Stubblefield. "Using Client Puzzles to Protect TLS," in *Proceedings of the 10th USENIX Security Symposium,* August 13-17, 2001.

[27] B. Waters, J. A. Halderman, A. Juels, and E. W. Felten. "New Client Puzzle Outsourcing Techniques for DoS Resistance," in *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS'04).* October 25-29, 2004, pp. 246–256.

[28] W. Feng, E. Kaiser, W. Feng, and A. Luu, "The Design and Implementation of Network Puzzles", in *Proceedings of IEEE INFOCOM 2005*, March 13-17, 2005.

[29] X. Wang and M. K. Reiter. "Mitigating Bandwidth-Exhaustion Attacks using Congestion Puzzles (Extended Abstract)," in *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS '04).* October 25-29, 2004, pp. 257–267.

[30] W. Feng. "The Case for TCP/IP Puzzles," in *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*, August 25-27, 2003, pp. 322–327.

[31] C. Dwork and M. Naor. "Pricing via Processing or Combating Junk Mail," in *Advances in Cryptology – Crypto '92*, August 1992, pp. 139-147.

[32] C. Dwork, A. Goldberg, and M. Naor. "On Memory-Bound Functions for Fighting Spam," in *Proceedings of the 23rd Annual International Cryptology Conference (CRYPTO 2003), Lecture Notes in Computer Science,* Vol. 2729, August 17-21, 2003, pp. 426-444.

[33] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. "Moderately Hard, Memory-bound Functions," in *Proceedings of the Network and Distributed System Security Symposium (NDSS'03)*, February 6-7, 2003, pp. 25-39.

[34] A. Back. "Hashcash – A Denial of Service Counter-Measure," Unpublished Manuscript. Available at: http://www.hashcash.org/papers/hashcash.pdf.

[35] D. Wheeler and R. Needham. "TEA, a Tiny Encryption Algorithm," Unpublished Manuscript. Available at: http://www.ftp.cl.cam.ac.uk/ftp/papers/djw-rmn/djw-rmn-tea.html. November 1994.

[36] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Inc., 1996.

[37] J.Kelsey, B. Schneier, and D. Wagner. "Related-Key Cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA," in *1997 International Conference on Information and Communications Security (ICICS'97)*, *Lecture Notes in Computer Science,* Vol. 1334, November 11-13, 1997, pp. 233-246.

[38] D. Wheeler and R. Needham. "TEA Extensions," Unpublished Manuscript. Available at: http://www.cl.cam.ac.uk/ftp/users/djw3/xtea.ps

[39] J. Crowcroft and I. Phillips. *TCP/IP and Linux Protocol Implementation.* John Wiley & Sons, Inc., New York, 2002.

[40] Mandrake Linux. Available at: http://www.mandrakelinux.com/en-us/

[41] C. Devine. MD5 Source Code. Available at: http://www.cr0.net:8040/code/crypto/md5/.

[42] D. Ireland, P. Gutmann, and AM Kuchling. SHA-1 Source Code. Available at: http://www.di-mgt.com.au/crypto.html#sha1

[43] The network simulator – ns-2. Available at: http://www.isi.edu/nsnam/ns/

[44] OPNET Modeler: Network Simulator. Available at: http://www.opnet.com/

[45] R. L. Rivest, A. Shamir, and D. Wagner. "Time-lock puzzles and timed-release crypto," Unpublished Manuscript. Available at: http://theory.lcs.mit.edu/~rivest/RivestShamirWagner-timelock.pdf. March 1996.

VITA

Timothy John McNevin was born on September 4, 1981 in Corning, New York. In 1999, he graduated from Jefferson Forest High School and in the fall of that same year he enrolled in Virginia Polytechnic Institute and State University. In the spring of 2003, he completed his undergraduate degree and achieved a Bachelor of Science Degree in Computer Engineering with a minor in Computer Science and graduated Cum Laude. As an undergraduate he was a student member of IEEE and a member of Eta Kappa Nu (HKN).

In the fall of 2003, he began to pursue his Masters degree in Computer Engineering at Virginia Tech under the guidance of Dr. Jung-Min Park. Following his Masters degree, he will begin working with BAE Systems in Reston, VA.