Analysis and Development of a Prototype for the Detection and Mitigation of HTTP Based Distributed Denial of Service Attacks

Stuart Gilbertson

Submitted in partial fulfilment of
the requirements of Napier University
for the Degree of
BEng (Hons) Computer Networks and Distributed Systems

School of Computing

December 2009

Authorship Declaration

I, Stuart Gilbertson, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:

Date: 25/11/2009

Matriculation no: 04002110

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

IT Administrators are constantly faced with threats as a result of the Internet (West, M. 2008). There is a huge range of solutions on the market today to assist and defend against such threats, but they are either costly or complicated to set up and configure properly. The assaults circulating on the Internet at this very moment range from malicious viruses designed to destroy or corrupt data to technologically advanced robot networks with intelligent coding that allows them to intercommunicate with each other, and send back captured private data like bank account numbers and credit card details to the owner of the malware (Hoeflin, D. et al. 2007).

Distributed Denial of Service attacks are getting more and more advanced as a result of sophisticated malware being released into the wild. As a result, new methods of mitigating these attacks need to be designed and developed. This thesis aimed to read similar related work done and gain an in-depth examination of DDoS taxonomy, bot nets, malware and mitigation methods. The thesis also attempted to take this research and use it in a prototype system that could potentially be used in a real environment to detect and mitigate an active DDoS attack on a server.

The main aim of this thesis was to analyse current Distributed Denial of Service botnets, malware and taxonomies to determine the best prototype to design and develop that could detect and mitigate an attack on a server. A human verification prototype was developed and implemented that would require user input to validate the visitor to a site was a real person. This system would only trigger if the visitor sent too many requests for the site per minute. If the visitor failed to validate, the system firewalls their IP address.

This thesis and prototype, although slightly inefficient at high load levels, could potentially help to mitigate a medium-scale DDoS attack on a website. The prototype does indeed detect a user that exceeds the threshold set, and it does then forward them on for verification. The prototype also then creates entries that simulate the IP address of that user being blocked at a firewall level. However, the prototype fails to appreciate any false positives that may occur. If a user was to exceed the threshold and then fail validation, their IP address would be firewalled on the server permanently (or until an Administrator flushed the firewall).

Contents

List of Tables

List of Figures

List of Code Snippets

Acknowledgements

Firstly I would like to thank my supervisor Prof. Bill Buchanan for his continued support and assistance that he has provided throughout the course of not only my thesis, but my University degree. I would also like to thank Dr. Gordon Russell for agreeing to be my second marker and showing interest in the novel approach taken when dealing with a DDoS. Lastly, I would like to thank my better half, Samantha, for putting up with my high stress levels towards the conclusion of the project.

1 Introduction

1.1 Introduction

This introduction section is intended to give the reader justification for the project by providing clear context by analysing the issues related to Denial of Service attacks faced by Administrators and IT professionals. It is important to continually develop technology and make efforts to create and deploy new methods of mitigating the attacks that occur on the Internet every day. Malware infects vulnerable computers all the time and the malware is constantly being re-written by attackers. Malware is getting more sophisticated (Zaytsev, V. 2009), and the reason for its existence is to allow attackers to control as many computers that do not belong to them as possible on the Internet. Interestingly enough, on 19th November 2009, UK2.net (a large Internet Service Provider in the UK) suffered a huge attack bringing down the majority of their services for at least three hours.

1.2 Context

IT Administrators are constantly faced with threats as a result of the Internet (West, M. 2008). They can be responsible for a plethora of different services and resources on a network at any one time and the challenges involved in ensuring service availability are vast (Franklin, J. et al. 2007). As a result, these Administrators rely on specialised hardware and software to combat attacks from external sources. There is a huge range of solutions on the market today to assist and defend against such attacks, but they are either costly or complicated to set up and configure properly.

One of the major problems is that the Internet protocol is designed to be as light-weight as possible, meaning security considerations are usually an afterthought, and usually too late. The assaults circulating on the Internet at this very moment range from malicious viruses designed to destroy or corrupt data to technologically advanced robot networks with intelligent coding that allows them to intercommunicate with each other, and send back captured private data like bank account numbers and credit card details to the owner of the malware (Hoeflin, D. et al. 2007).

It is these compromised machines (often referred to as zombies or bots) that this project focuses on, in particular, the ones that are used for attacking web servers and affecting the availability of that type of service. These robotic networks (or more commonly known as “botnets”) are usually controlled by one person, the attacker. This person has the ability to command the botnet and have a coordinated attack pointed at a website to bring that service offline. Figure 1.1 shows the typical taxonomy of a botnet. The victim could be any service from HTTP to DNS.

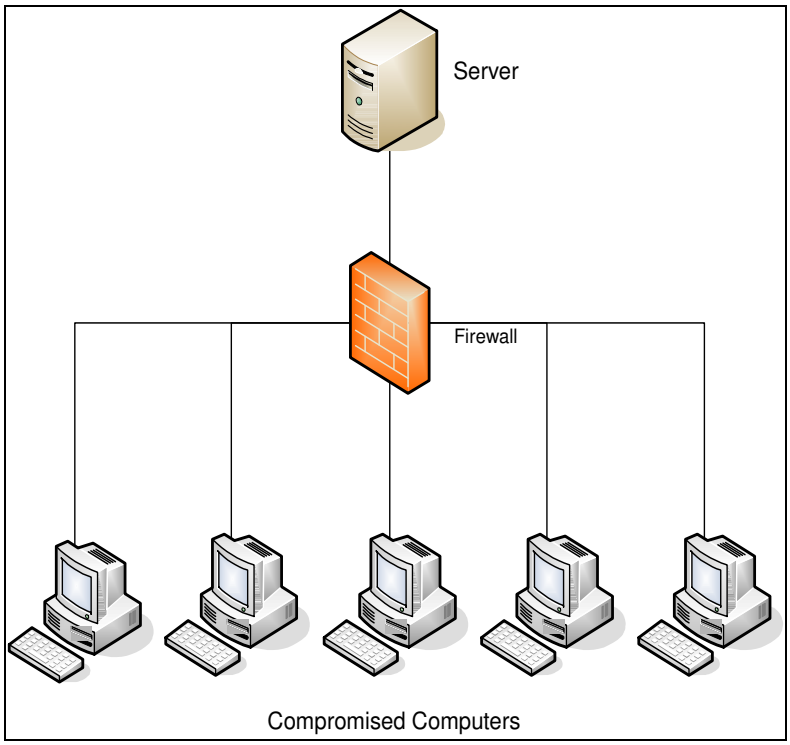

Figure 1.1: A taxonomy of a stereotypical botnet

1.3 Aims and Objectives

This thesis aims to analyse and develop a working prototype system that can detect and mitigate a web-based Distributed Denial of Service attack. To achieve this aim, the thesis has the following objectives:

- Analyse current DDoS taxonomies and attack signatures.
- Develop a working prototype to detect and mitigate an HTTP-based DDoS.
- Evaluate and test the prototype to determine the effectiveness of it and to clearly show any strengths or weaknesses.

1.4 Background

A Denial of Service (DoS) attack is defined as one computer disrupting another to prevent legitimate users of a service from using that service. There are various ways in which a DoS Service attack can be carried out, these include attempts to flood a network with traffic which results in the bandwidth being consumed and directed attempts to disrupt a specific service based on known flaws (CERT, 2001). With today's computing power and bandwidth resources, it is increasingly difficult for one single machine to bring down another by simply exhausting its resources.

One recent DoS exploit (see Appendix 4) discovered in the web server software *httpdx* (Jasper, 2009) results in it crashing when processing a request. Because this request can be issued remotely from any computer connected to the Internet and does not require a large amount of resources to carry it out, it is exceptionally dangerous and can cause catastrophic failure with the services. The likelihood of a

server being brought offline as a result of this kind of attack is high, and most software vendors will look to patch the vulnerability immediately.

This thesis researches the next level of DoS attack, a Distributed Denial of Service Attack. The example of a botnet previously given is a reasonable overview of how a Distributed Denial of Service (DDoS) is set up. Malware with the botnet code is sent out to insecure and un-patched computers on the Internet using various malicious methods. One of the most common methods for infecting insecure computers is by embedding malicious code on websites and simply waiting on the user of the computer to visit the site (Mavrommatis, P. et al. 2008). Once these machines become infested by the code they join the botnet. Having one computer sending a request for a service once a minute is not a problem, in fact, it is quite normal. The problem comes when a vast number of computers request the service at the same time (or in a short period of time). This is what effectively happens during a DDoS. Normal service is sustained because the resources available on the server are not exceeded by the requests the server has to respond to. As soon as these resources are exhausted, the services become unavailable.

A DDoS simply exhausts the service resources by coordinating hundreds of requests for that resource at the same time. Figure 1.2 shows the typical taxonomy of an active DDoS attack. With just five computers making a request to the victim every minute it is unlikely the victim (for example a web server) would have trouble being able to serve the requests. If we let each compromised computer represent 5,000 compromised computers, however, it becomes clearly apparent that for every minute, 25,000 requests for the website must be processed by the server. That equates to over 416 requests a second and at this stage, without DDoS mitigation software or hardware it is likely that the server's resources (for example, CPU, Memory or Network Bandwidth) would be completely exhausted.

Complete DDoS mitigation is built around four key stages:

- Accurately detect bad traffic from good.
- Maintain reliable infrastructure.
- Have a plan for scalability to meet the resource demands.
- Mitigate the attack – do not just detect it.

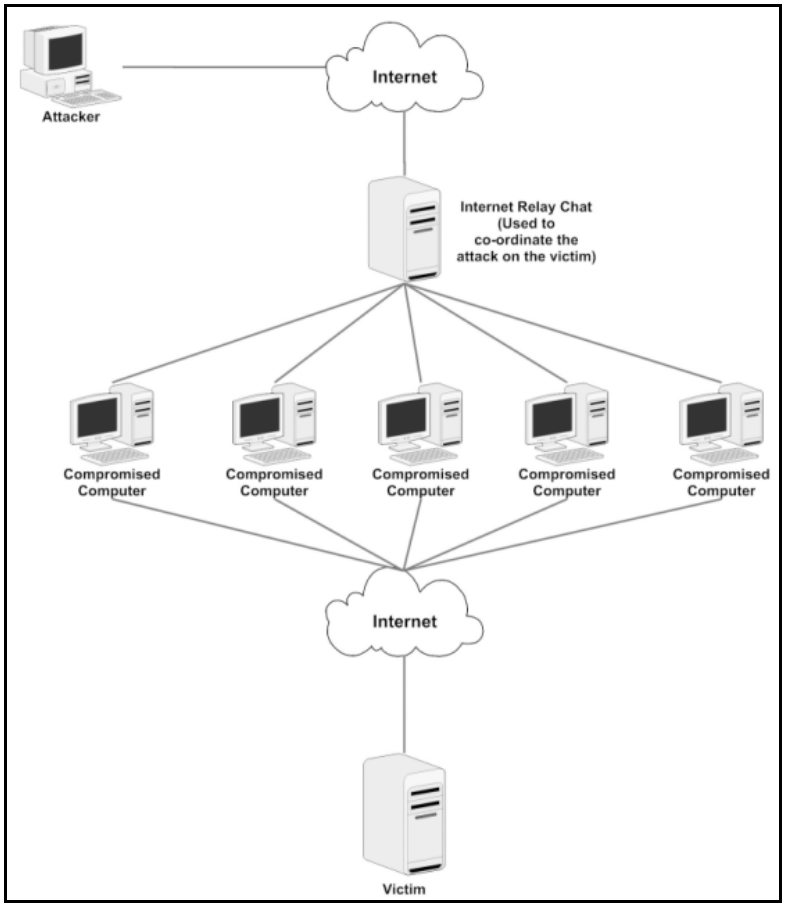

Figure 1.2: A simple Distributed Denial of Service attack

Strategies for mitigating such attacks are available, but are either difficult to set up or are costly to purchase and maintain. As a result, many small, medium and even large businesses fail to prepare their IT infrastructure in the correct way to defend against these types of attacks.

In February 2000, Yahoo, Amazon, eBay, CNN and ZDNet were just a handful of businesses that suffered downtime or exceptionally slow websites as a result of a DDoS attack on their infrastructure. This unavailability of service costs organisations thousands of pounds every hour in lost revenue. Yahoo suffered losses of approximately $500,000 USD for the three hours it was unavailable and Amazon made losses in the region of $600,000 USD whilst its services were unavailable for approximately 10 hours (Kessler, G. 2000). Almost ten years on these attacks are still ongoing, are still a problem, and are becoming more common.

In some cases, an attacker might want to try and profiteer from owning his bot-net. In doing so, he can either send a letter warning an organisation that he is about to target them next and demand a ransom. Or he has the option to target them immediately and then demand payment to stop the attack. In the case of Authorize.net (A U.S. credit card processing firm) the attacker actually sent a letter warning that the attack was imminent if payment was not made. TheRegister.co.uk (an online IT news website) reported the incident, adding that the gambling industry has experienced progressively more sophisticated attacks (Leyden, J. 2004).

As recently as August 2009 the popular social-networking website *twitter* – a popular real-time messaging system - suffered a denial of service attack on their site. Their technicians worked around the clock to resolve the problem, but for a few hours at least, visitor numbers halved. Figure 1.3 shows the drop in visitors to the Twitter site as a result of the denial of service.

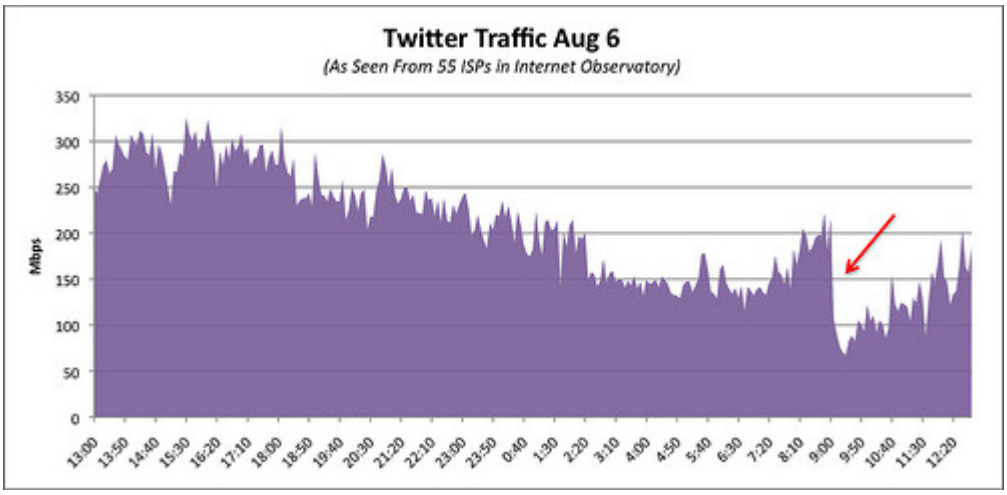

Figure 1.3: The fall in visitor numbers when Twitter was unavailable. (Labovitz, C. 2009)

1.5 Outline

This thesis provides in-depth details about what solutions are already available for detecting a DDoS attack and what solutions already available on the market for mitigating such an attack. It then reviews these solutions and determines which ones are the most effective, where they are weak, and how they could be improved. The design stage of the project seeks to put to good use the previous research done and tries to employ that work in developing a working prototype that could be deployed in a real-world scenario. The thesis then documents how the prototype was designed and evaluates and tests the deployed solution to determine effectiveness and weaknesses.

2 Literature Review

2.1 Introduction

This chapter outlines the history of the IT technologies used when dealing with DDoS, the taxonomy of a DDoS, how a DDoS is mitigated and what software and hardware is used to achieve a successful defence against an attack. This is an important part of the project, as having a full understanding of the methodologies and a technical understanding of the taxonomy of a DDoS is crucial for understanding how to design and implement a working prototype that might effectively mitigate an attack. The main things involved in defining the taxonomy of a DDoS include signatures of attack, existing malware, current active Botnets, and methodologies of these.

This literature review defines DDoS taxonomy so that a prototype that deals with a specific service can be designed successfully. It looks at how to detect attacks based on the signature of the requests, and how people have analysed these log files and to what success. It lastly looks at how and what methods of mitigation are available and being used (and also which ones are not used for various reasons) to defend against these attacks.

2.2 DDoS Taxonomy

Martin (Martin, J. et al. 2002) outlines that there are numerous ways to classify a DDoS attack and the number of defence methods for each is difficult to comprehend. The attacker's goal during a DDoS is to disrupt the victim's services as much as possible. The motives behind these attacks can be anything from a simple dislike of the victim to political or business reasons. In the majority of the cases, the victim of the attack is not actually the intended target, but suffers as a result due to the physical way a DDoS operates.

Four popular bots currently active and participating in DDoS attacks are Agobot, SDBot, RBot and Spybot. One of the main issues with analysing and understanding these bots is that they are constantly evolving and many variants exist in the wild coded by others, hacked by more. Table 2.1 shows an overview of the four popular bots and their capabilities.

Controlling a botnet is not a simple task. Discovering your own botnet, and then organising your botnet can be difficult at best. Because the compromised machines join the botnet randomly (victims being recruited through viruses, worms, etc.) the botnet owner does not know exactly which machines will become available in the botnet swarm. A way to command and control the botnet is required. Dagon (2005) identifies that this problem is solved by coding the bots in a way that they will

connect to a Command and Control (C&C) or Relay server. This server's job is to allow the botnet owner to communicate with the bots without the botnet owner having to send the command individually to each bot. When a victim becomes infected by a piece of malware, the infected machine then contacts and joins the C&C server. The botnet owner can see this "JOIN" and knows that they have secured another bot in their "cloud".

Table 2.1: An overview of DDoS Bots (Dulary, N., et al. 2006)

Bot Name	Language	Capabilities	# of Variants
Agobot	C++	Is written in a modular way so allowing new attacks to be added with ease. Most comprehensive set of DDoS attack tools at its disposal including: SYN Flood, UDP Flood, HTTP Flood. Simple command line issued via a relay server will initiate the attack. Issuing "ddos.stop" will stop the attack.	Over 600 different versions.
SDBot	C++	This bot focuses on just ping and UDP flooding tools. It is possible to download a different variant (SYN Flood Edition) which includes more advanced TCP SYN flooding attacks. The following commands are available with this bot: UDP Flood, Ping Flood, SYN Flood.	Over 1800 different versions.
RBot	C++	This bot has the following commands available for attacking: SYN, ACK, Random Flooding. ICMP Flood, Ping Flood, UDP Flood.	1600 different versions.
Spybot	C	SYN, SpoofdSYN and PING are the only tools at this bot's disposal. However, Spybot has more spreading abilities than the rest of the bots making it even more dangerous.	Only 200 versions.

This form of distributed computing is simple, and as a result, works effectively. One method of communicating with the botnet is by way of an Internet Relay Chat server or IRC. IRC is designed to be a stable platform for chat and as a result, is an ideal infrastructure for deploying a communication channel for a botnet. Most IRC software is open source and freely available to download and install on one's own hardware. As a result, it is relatively straightforward to deploy a working chat server on a machine without anyone realising. The bots can be coded so that once infecting a computer, they automatically seek out this IRC server (either using an IP address or a Domain Name) and join one of the chat-rooms (known as "channels") and await the botnet owner's commands. Figure 2.1 shows a screenshot of multiple bots connecting to an IRC server and channel (#BOT-CHAN).

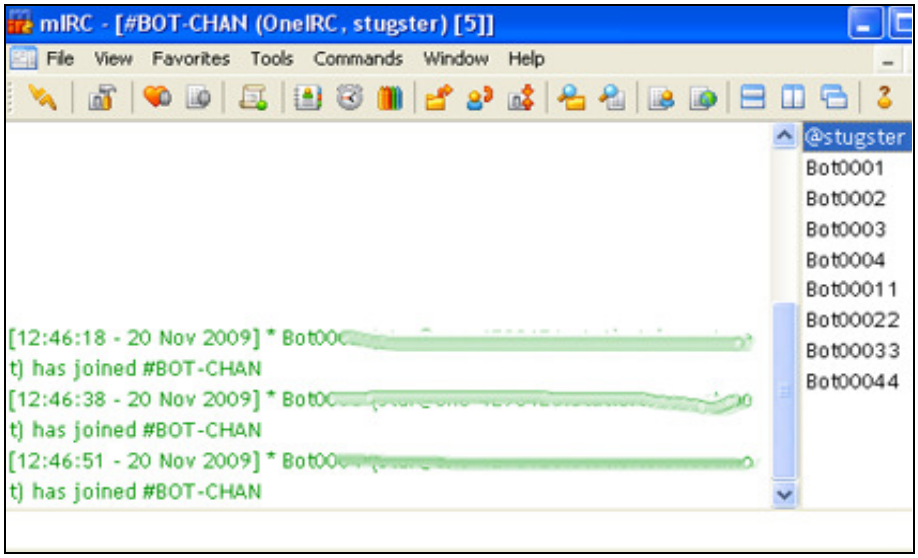

Figure 2.1: Bots connecting to IRC chat room

It is interesting to consider that if all machines connected to the Internet were actually secured properly, DDoS attacks would be uncommon. Despite numerous notifications, numerous attacks, and numerous reports, people still neglect to maintain computers properly.

Martin (2002) defines that there are two main classifications of DDoS attacks based on the vulnerability of the targeted victim. The first is a *protocol attack*, the second a *brute-force attack*. Protocol attacks seek to exploit known bugs in specific software, or known methods that will consume an exceptionally large amount of CPU time, Memory or Hard Disk. A popular protocol attack is a TCP SYN attack. In a TCP SYN attack the attacker sends multiple requests to the victim that are never completed (Chen, W., et al. 2005). Due to the nature of the TCP protocol, the server must wait for an acknowledgement of its response from the attacker which never materialises. As a result, the TCP sockets are quickly filled a brute-force attack simply seeks to directly consume available resources without targeting a specific service vulnerability. The attack creates multiple, seemingly genuine, requests to the server for information. By doing so, and with the number of attacking hosts more than the server can cope with, the service is soon unavailable.

One area of research is defence against DDoS at the source of the attacks. The work of Mirkovic (Mirkovic, J., et al. 2002) on a defence system deployed at the source-end of the network is fundamental to research in this specialised field. Their system analyse the packets as they leave the compromised machine's network, and estimate the number of packets that should return. They state that all DDoS attacks should be stopped as close to the source as possible. By filtering packets in this way, we save hundreds of Gigabytes of bandwidth as a result.

Their D-WARD system monitors the behaviour of each node the source network communicates with and actively looks for signs of communication difficulties (Mirkovic, J., et al. 2002). This system then analyses the situation and calculates the probability of an active DDoS attack. The D-WARD system then imposes a rate limit

on the outgoing packets from the source. Once the situation has been confirmed (i.e. packets are still suspicious) the D-WARD system restricts the rate even further. Assuming the DDoS continues from the source, the rate will be limited to such a point that almost no data is able to be sent from the source at all. Once the attack subsides or packets appear more genuine, the D-WARD system then slowly increases the rate of traffic to allow faster connections (Mirkovic, J., et al. 2002).

2.3 Detection of HTTP Based Attacks

In terms of an HTTP based DDoS attack, detecting one is not straightforward. The biggest problem is that the traffic looks genuine. When a DoS occurs, it is obvious to the administrator an attack is ongoing as a result of the source IP address. With a DDoS attack (specifically an HTTP-based in this instance) the source IP addresses are all different. As a result, the attack simply looks like a busy website instead of a malicious attempt to bring the service down.

Analysis of log files is not a popular field of research in Computing according to Valdman. His paper which includes a section on HTTP Server Logs (Valdman, J., 2001) helps to analyse an example in Figure 2.2 which shows a period of 10 minutes of real traffic where a DDoS attack starts. The Unique Hits (Normal Traffic) is the data gathered on the same day a month before the attack occurs. We can clearly see a difference between the Unique Hits (Normal Traffic) and the Unique Hits (DDoS Attack). During the DDoS attack, the "GET /" requests increase from 15 per minute at 1 minute, to over 90 requests a minute just 6 minutes in to the attack. It is clear that something is going on, but whether this is just a popular website or an active attack is unknown. Looking at just one IP address which is known to be participating in the attack, we see the traffic looks normal.

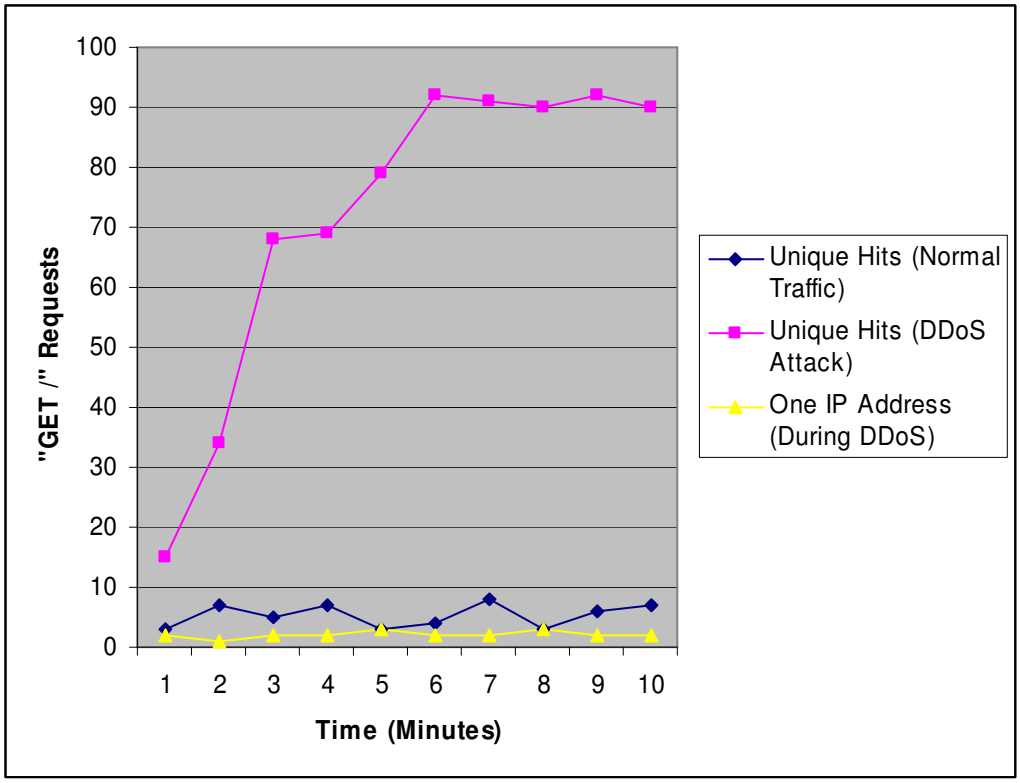

Figure 2.2: DDoS Attack Data over 10 minute period

It is possible to look into the attack in more detail, namely the signature of the request. In the case of the DDoS attack on Equiphase Limited (a small web hosting company), they suffered a DDoS attack targeted at a client for over 24 hours. Equiphase provided the raw data for analysis after the attack subsided. Figure 2.3 shows an IP address known to be compromised. When analysing just one IP address from the attack, we can see that each request is for "GET /". Straight away we can deduce that this IP address may be a compromised machine, as it is unusual for a host to request the root (index) page of a website time and time again.

	A	B	C	D	E	F	G	H
1	78.163.*.*	-	-	[06/May/2009:09:46:35	+0100]	"GET / HTTP/1.1"	200	16955 "-
2	78.163.*.*	-	-	[06/May/2009:09:46:35	+0100]	"GET / HTTP/1.1"	301	237 "-"
3	78.163.*.*	-	-	[06/May/2009:09:46:38	+0100]	"GET / HTTP/1.1"	200	16955 "-
4	78.163.*.*	-	-	[06/May/2009:09:46:39	+0100]	"GET / HTTP/1.1"	200	16955 "-
5	78.163.*.*	-	-	[06/May/2009:09:46:41	+0100]	"GET / HTTP/1.1"	301	237 "-"
6	78.163.*.*	-	-	[06/May/2009:09:46:41	+0100]	"GET / HTTP/1.1"	301	237 "-"
7	78.163.*.*	-	-	[06/May/2009:09:46:41	+0100]	"GET / HTTP/1.1"	301	237 "-"

Figure 2.3: IP Address Data from Equiphase DDoS Attack

A genuine request for a website would normally send only one GET for the root page and multiple other GET requests for additional files. These files could be images on the site, included CSS files, or additional pages. Figure 2.4 shows an example of a GET request for a CSS file (minimal.css) which would usually be sent after retrieving the original index HTML from the initial "GET /" request.

```
80.192.121.84 - - [19/Nov/2009:09:46:45 +0000] "GET /minimal.css"
"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-GB; rv:1.9.1.5) Gecko/20091102
Firefox/3.5.5 GTB6"
```

Figure 2.4: GET request for the minimal.css file

In 2004, the University of Florida sustained a massive DDoS attack on a web server caused by the Netsky worm. The work done by Wiens (Wiens, J. 2004) is fundamental to understanding the Netsky worm and in particular, contains a similar DDoS signature to the one suffered by Equiphase. Figure 2.5 shows a sample of the signature Weinis worked with from the Netsky virus. In this Figure, we can compare the original GET requests from Equiphase with the GET request from the University of Florida. The biggest difference is in the last section of the log where the client-browser will identify itself to the server. In the case of the machines infected with the Netsky virus, this does not occur. As a result, it is relatively easy to detect an attack and filter those kinds of packets. Unfortunately, Equiphase was not as lucky, and each attacking host identified a (possibly fake) browse and version number. It could be assumed that the Equiphase attack was the result of a Netsky virus variant, possibly modified to make detection and mitigation even more difficult.

```
aa.bb.cc.dd - - [03/May/2004:21:02:21 -0400] "GET / HTTP/1.1" 200 3333 "-"
"-"
```

Figure 2.5: The signature of the 2004 Netsky virus

Another method of detecting an ongoing DDoS attack is simply watching the network adaptor's bandwidth consumption. In the case of Equiphase, the relatively small attack consumed 320 GB of data in the month of April 2009, substantially more than would normally be transferred by a small web host. Figure 2.6 shows the data transferred over a three-month period and clearly demonstrates the difference between a normal month and a month in which a DDoS occurred (over twice as much data was transferred – with the DDoS only lasting 24 hours). It should therefore be relatively easy for an administrator to question the site owner as to whether the surge in visitors is due to marketing efforts or to an unsolicited attack on the website (that is, assuming the site is still available).

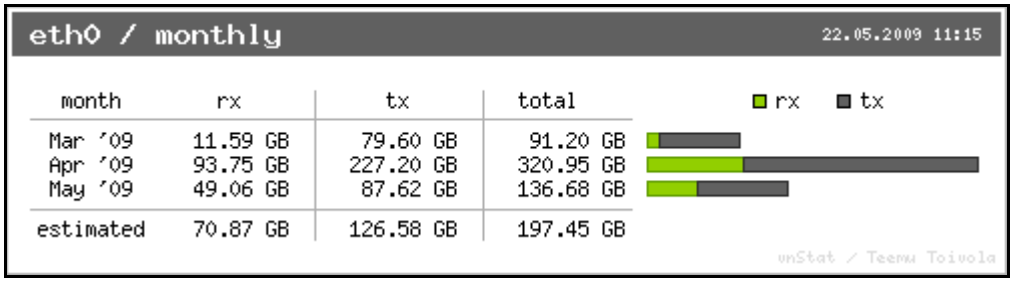

Figure 2.6: Bandwidth consumption for a three-month period from Equiphase

2.4 Current Mitigation Methods

It is important to understand the current methods of mitigating a DDoS attack, from the simple methods to the more advanced and sophisticated technical methods.

2.4.1 DNS Management

The first and easiest way to mitigate a DDoS attack is to simply point the domain affected to a different IP address – for example a loop-back IP address; "localhost" (127.0.0.1). Assuming the compromised update their local DNS cache, or query a DNS server that updates, by doing this, the compromised machines simply send the

request to their own loop-back address rather than out onto the Internet. There are a few drawbacks to this method of mitigation however. The most obvious is that the website will no longer be available to normal users, therefore although saving bandwidth resources, the attackers are still winning by denying access. Weins, or the people involved in deploying the mitigation system within the University of Florida when they suffered a DDoS Attack used this method of mitigation to manage the traffic flow temporarily until additional supporting solutions could be implemented (Weins, J. 2004).

Secondly, the bots might not just target the domain name. It is entirely possible that they target an IP address associated with the domain and not the domain itself. If this occurs, then the IP address the site is hosted on should be changed, and the DNS updated to reflect this. If the Time To Live on the DNS is a high number, this could take hours to propagate on the Internet, causing exceptional downtime. On some occasions, the compromised machines will target the root domain name (Ballani, H., et al. 2002) and not sub domains, meaning that one could simply route the domain to a loop back address and ensure an A record exists for the 'www' sub domain to an active IP address. This method would prevent the DDoS attack, but any users failing to enter 'www.' in their web browser would not be directed to the active website.

This is a fast but dirty way of mitigating a DDoS attack and should only be used in an emergency whilst investigating the attack and looking at other methods to defend against it. DNS management's main advantage is that it is a relatively fast way of temporarily mitigating a DDoS attack. It allows the administrator to null route the domain affected (Ballani, H., 2002) so that the server can be managed and prepared to deal with the attack once the domain is brought back online.

It cannot be guaranteed that the attacking botnet is actually pointing to the server using DNS (Champagne, D., 2006). If the botnet is set to point to the IP address, this method of mitigation becomes completely useless.

2.4.2 TARPIT

The TARPIT method of mitigating a DDoS attack is achieved by setting the TCP window size to just a few bytes. The TCP protocol is designed in a way that the "compromised machine" must not send any more data until the "ACK" the initial packet has been received. This method effectively hangs the client machine for a set period, and detection becomes easy - if the "compromised machine" sends further packets within that timeout period, we can assume it is a compromised machine.

Normally it would be assumed that adding a DENY to iptables would be sufficient to block the attack (Molsa, J., 2005). In theory this would work, the denied IP address would be unable to send requests to the server but the packets are still being sent, as a result, bandwidth can still be heavily consumed. In addition, the attack could simply become a SYN flood attack instead which could still compromise the server (Stewart, J. 2007).

As a result of setting the TCP window to just a few bytes, and because the compromised machine has not received the ACK from the initial packet sent, it is forced to keep trying to resend those few bytes again and again. As a result, the machine never gets to the stage where it is able to send full GET / requests therefore if the requests are never sent, they are never received or processed by the server – a huge saving in resources.

The University of Florida successfully mitigated such an attack when they underwent a massive DDoS attack in 2004. They made use of a TARPIT system which is effective at slowing down worm propagation rates (Northcutt, S., et al., 2002).

2.4.3 Hack-back system

Worms are commonly used to compromise the host machines thus building the attacker's botnet. One useful addition the attacker might include in his worm is a backdoor. The Netsky worm comes in many variants, and has previously targeted large organisations such as the University of Florida. It was discovered that the Netsky variant contained backdoors which allowed the attacker to upload executable code which could further compromise the machine (Wiens, J., 2004, p. 11). This means that when a DDoS attack is detected, the IP address of each compromised machine would be connected to from a clean host and code to remove the infestation could potentially be uploaded.

The problem with this method of mitigating a DDoS is that to carry out the fixes, one would have to gain 'unauthorised' access to the compromised machine; thus breaking the law in doing so. It could also potentially take a long time to process each individual IP address considering that the number of compromised machines that might be attacking the victim at the time can easily be well in excess of 1,000. Lastly, it would not be long until the attacker realised this hack-back method was being utilised, and they would simply 'patch' the worm and release yet another variant. Jordan Wiens explains that:

> "There is no substitute for proper management of hosts via legitimate methods, and being able to push out virus updates and other patches through normal channels. In an emergency situation, however, such an action could be considered as a possible response." (Wiens, J., 2004, p. 12)

The hack back idea could be an ideal solution, but only if we assume the source code of the worm which infected the compromised machine is always going to be the same. It is impossible to guarantee that we would be able to gain access via the remote back-door of the machine, or that the source code would never be updated by the botnet owner. As a result, this makes this method unsound as a mitigation method. In addition to this, it would require all the countries of the world agreeing to change their laws, for example in the United Kingdom we have the Computer Misuse Act – namely accessing a computer which one does not have permission. It is unlikely that this change in any legal system would ever take place.

2.4.4 Load Balancing

It could be argued that load balancing is not actually a true method of mitigating a DDoS because it does nothing to prevent the attack from occurring. It only allows for the load to be spread across multiple servers. Load Balancing should never be used as the only method of mitigating a DDoS attack, but should always be used in conjunction with other methods to provide further resilience during an attack.

It usually takes time to detect and mitigate a DDoS. There are usually numerous different IP addresses that require blacklisting or denied in the firewall, therefore having load balancing might ensure that services remain available until all compromised machines have been blocked from the network. Lastly, using only load balancing could result in a costly bandwidth bill at the end of the month as a continued and sustained attack is capable of using in excess of Gigabits per second.

There are sophisticated software solutions (Zhang, 2000). that make load balancing services extremely simple to set up, and easy to manage. The Linux Virtual Server Project is one such product that can be used to rapidly set up a load balanced environment for FTP, HTTP, etc. The main problem with load balancing to combat an attack is that it does nothing to prevent the packets arriving and being served by the service (Chi, K-H. et al, 2009). It simply scales the resources to cope with the demand which does nothing to fight the DDoS. Load balancing therefore, should only be used alongside real mitigation methods as a way to improve availability of services, but not guarantee them.

2.4.5 Cisco Traffic Anomaly Detector (TAD)

Cisco Systems Inc. (one of the most popular hardware vendors for networking devices) understands the reality of the treat to businesses as a result of DDoS attacks. In their white paper, they demonstrate the impact of DDoS attacks and additionally, the multiple points of vulnerability and failures a network can have as a result of a DDoS attack. Figure 2.7 (Cisco Systems Inc., 2004) shows server-level DDoS attacks where an individual server within an ISP can be the target, but also at the gateway level where the ISPs router infrastructure resides. These DDoS attacks can result in the ISP backbone suffering and routers and DNS server availability being lost.

It may seem suitable to look at the possibility of a Cisco Intrusion Detection System for detection of an attack. These devices are capable of detecting application layer attacks, but are unable to detect DDoS attacks that use valid packets. According to Cisco, most of today's DDoS attacks use valid packets. In addition, they only detect packets and do nothing to deny the packets or mitigate an attack (Cisco Systems Inc., 2004). As a result, Cisco has deployed additional hardware to withstand a DDoS attack. The Cisco Traffic Anomaly Detector acts a detection device and sends alerts to the Cisco Guard XT. The Cisco Guard XT is a high performance mitigation device that puts suspect traffic through a five stage analysis and filtering process. This process is designed by Cisco to remove malicious traffic whilst allowing all genuine packets access to the resources (Cisco Systems Inc., 2004).

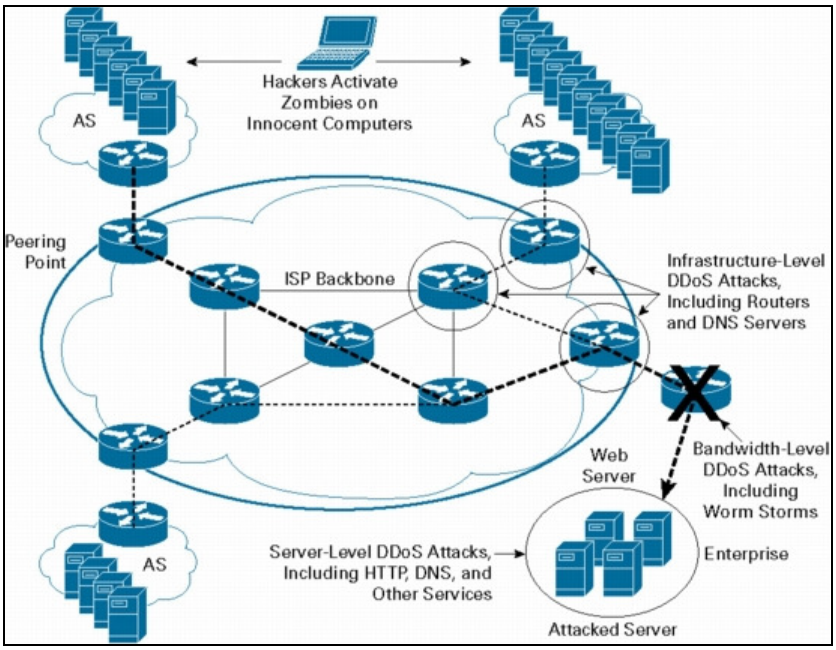

Figure 2.7: Cisco's diagram of the effects of a DDoS attack on the Internet (Cisco Systems Inc., 2004)

2.4.6 CAPTCHA

CAPTCHA is a simple method for human verification. Random letters, words, numbers or phrases are displayed on-screen for the human to read. It is an acronym of Completely Automated Public Turing test to tell Computers and Humans Apart. The Carnegie Mellon University processed forms in the United States for trademark of the term, but abandoned this in April 2008 (Edward, P., 2007). With this the user is required to input what they have read on-screen into a text box and submit their response to the server. The server – usually a web server – will then compare that response with the original text and if it matches can assume the user is a human being.

CAPTCHA is widely used nowadays on most websites. Its most popular use is for web-based contact forms to prevent and reduce spam. When a user wants to submit information online, they are required to enter a verification code to ensure they are not an automated bot. Figure 2.8 shows an example of a CAPTCHA form from Google.

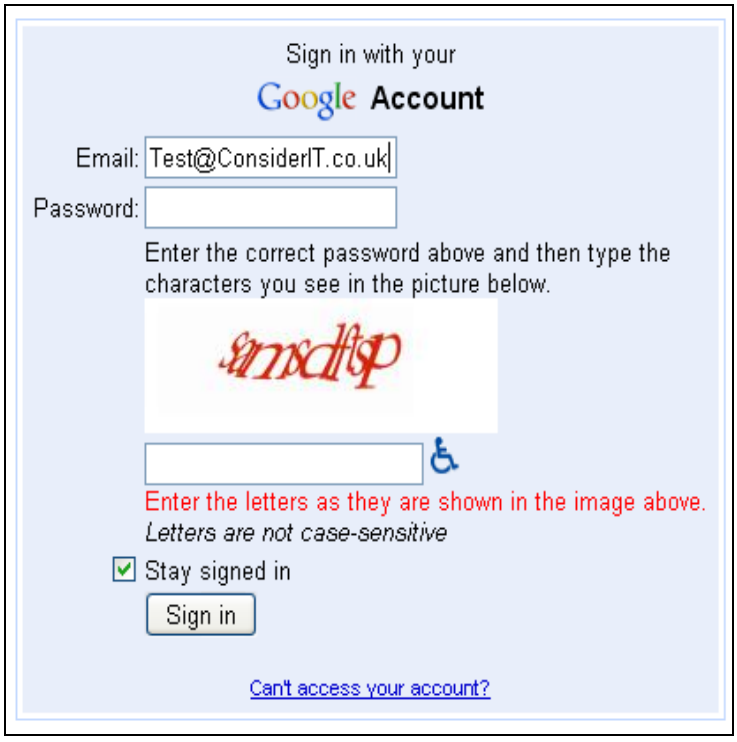

Figure 2.8: Google's CAPTCHA form

2.5 Conclusions

Load balancing can be useful for helping sustain service availability. By deploying a load balanced system alongside additional mitigation solutions, service interruptions should be minimised. TARPIT has already proven itself to be a successful way to mitigate a denial of service attack using TCP protocol rules to its advantage. However, this thesis is aimed at developing a new methodology that could help limit and manage a denial of service attack, and as a result, TARPIT will not be integrated into the final design. The Cisco Anomaly Detector and Guard XT are also successful methods of mitigating medium-sized attacks. The disadvantage for this system is that it is costly to implement. It also fails to slow down the attack from the host-side meaning that bandwidth would still be consumed at a high rate.

From the research completed in the literature review, CAPTCHA has not previously been used as a way to mitigate an attack. Although it is evident there would be no way for it to slow down an attack from the host-side, it may well be a solution that could work in place of one of the more expensive systems on the market today. As a result, at this stage it may be that CAPTCHA would be a suitable platform for taking further into the design stage.

3 Design

3.1 Introduction

To develop a working prototype that detects and mitigates HTTP based DDoS attacks, there must be adequate design of a system before it can be implemented. The system was designed so that it could be deployed in a Distributed environment (that is, the CAPTCHA part need not be on the same physical server). The system also integrates directly with Apache making deployment of the Detection module (Mitigate.c) exceptionally easy for any novice IT administrator to set up.

The problem scenario is when too many hosts (or compromised computers) request the website at the same time. The server cannot cope with the requests and service availability is affected. Figure 3.1 shows a basic overview of the network topology used in the prototype system. The system detects when an individual host has requested a website too many times and then requires that host to verify they are a human. Certain exceptions to this rule exist (for example, Search Engines are exempt from having to verify at all and are always given access to the site without hindrance).

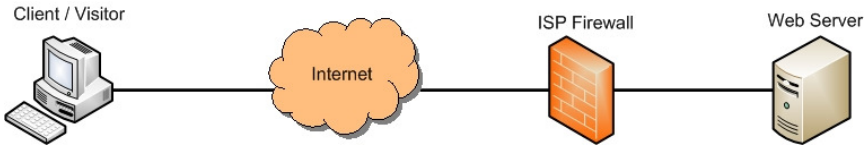

Figure 3.1: Overview of the network topology

3.2 System overview

All compromised machines are automated attacks, therefore using a method that checks whether a request originated from a real human being should help to deny the attack. The system will determine whether an IP address has requested a page too many times in a given limit. Figure 3.2 shows an overview of the prototype. Figure 3.3 shows an example of the flow of data and the rules that determine whether a request should be allowed or denied. The system comprises two custom modules that allow for the mitigation of an HTTP based attack.

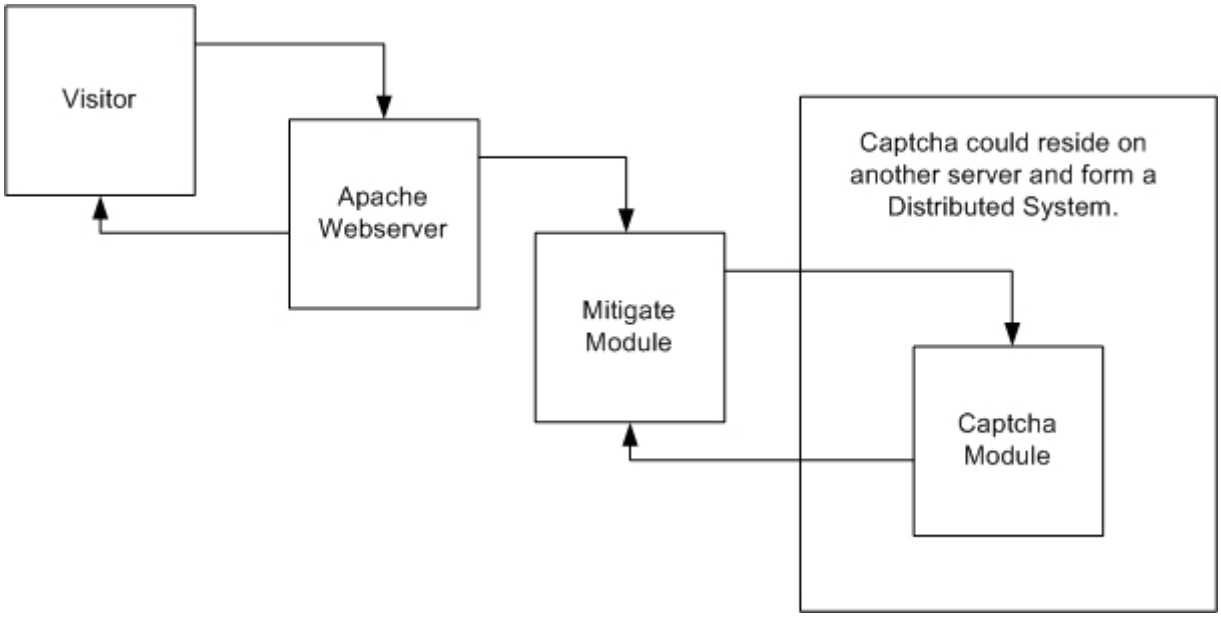

Figure 3.2: Overview of the prototype

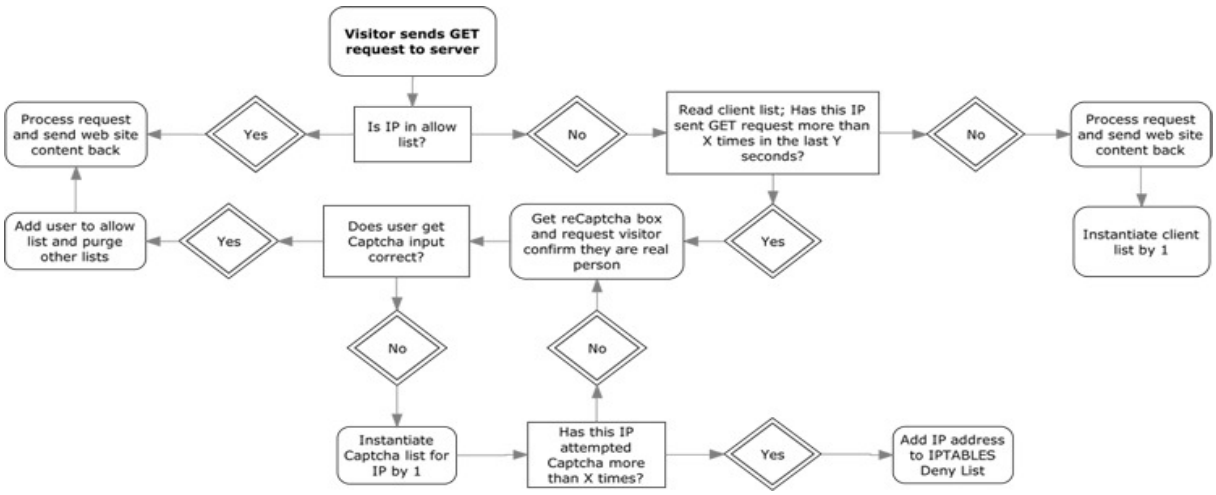

Figure 3.3: Flow diagram for CAPTCHA system

3.3 Apache Module

The Apache Web Server will be used to serve web page requests for the prototype. The server allows for custom modules to be included, and a basic module for detecting a possible DDoS attack and then mitigating it will be developed. The module will detect the number of times a user has requested the web page in a given time.

Figure 3.4 shows a more detailed view of how the three components interact with one another. The apache web server runs with the module compiled. Whenever a user visits a website hosted on the Apache web server, the Mitigate.c (once compiled, becomes mitigate.so) module is called.

The module should have the ability to define an "allowed.hosts" file for known IPs we do not want to "challenge" with the captcha.php module. The module should also have the ability to send a user to another server to process the captcha.php

module (saving resources on the primary server until the user has been confirmed as human). The module should have a variable that can be changed which defines the requests per minute that an IP address can send to the server before triggering the captcha.php module.

When the limit for number of access requests per minute is exceeded, the module should forward the user to the captcha.php module for verification of the user making the request. If verified, the captcha.php module will send back a confirmation to the mitigate.c module, if not then the user will be presented with the CAPTCHA page again and again (until the captcha.php limit is exceeded).

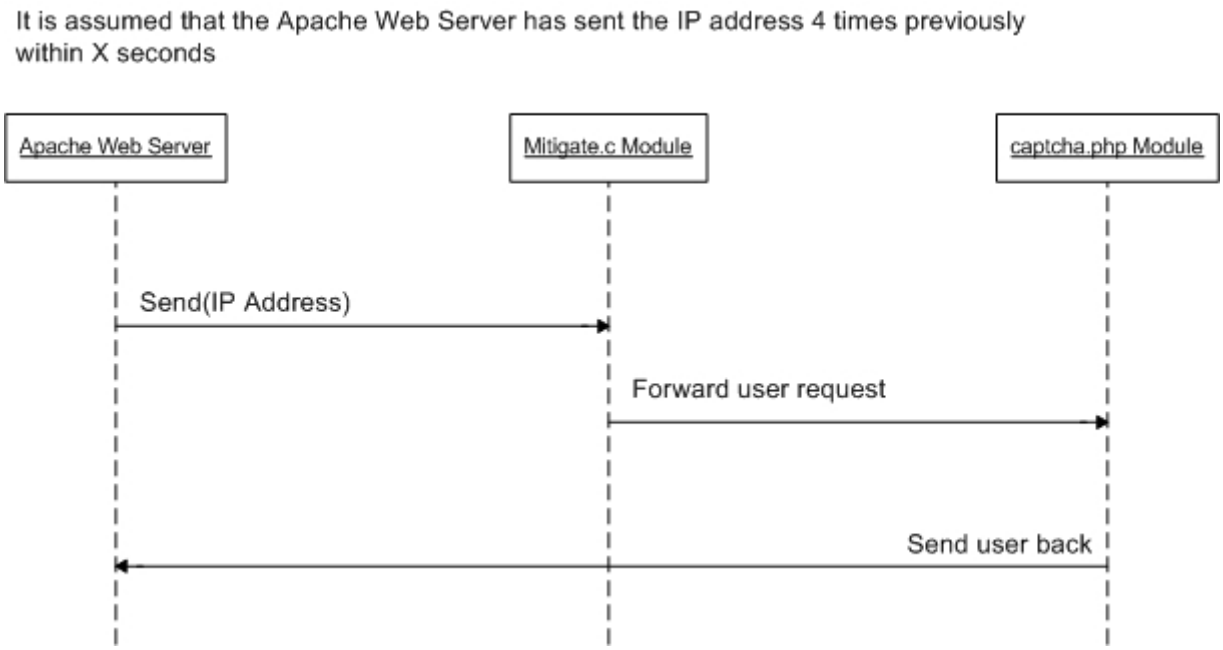

Figure 3.4: UML Diagram of Apache Module calls

3.4 PHP Module

Once the request has been handed from the mitigate.c module to the PHP module, it is the PHP modules responsibility to determine whether or not the request was made by a human or an automated robot. The PHP module makes use of the ReCAPTCHA Project, a free, hosted CAPTCHA service which the owners claim process over 30 million CAPTCHA requests a day.

The PHP module should create a variable which holds the number of attempts the user has taken. It should also have a variable which holds the maximum number of attempts the user is allowed. If the user exceeds the threshold of this limit, the IP address they are using will be denied using iptables.

If the request is processed correctly by the user, the Captcha.php module should send back this result to the mitigate.c Apache module so that it may add the IP address to the allowed list of hosts. As a result of this inclusion the user will never be required to verify themselves again whilst using that web server (or until the list is purged by an administrator). Figure 3.5 shows the process as described previously of the Captcha.php system. It needs to be established how the DENY will be added to

the IPTABLES firewall rules, as PHP normally runs under the web server user (either nobody or apache) and therefore would not normally have permission to configure the firewall in Linux.

As this is a prototype, a workaround was used was to simply deny access using '.htaccess' file. This will deny the user access to the requested website, but will not deny them access to the server. In a real world situation, this is of no real use, but is useful as verification that the prototype is working as designed.

To make designing and implementing the system speedier, the ReCAPTCHA Project have readily available libraries for us to use which lets us call the CAPTCHA and process the requests without having to code a great deal.

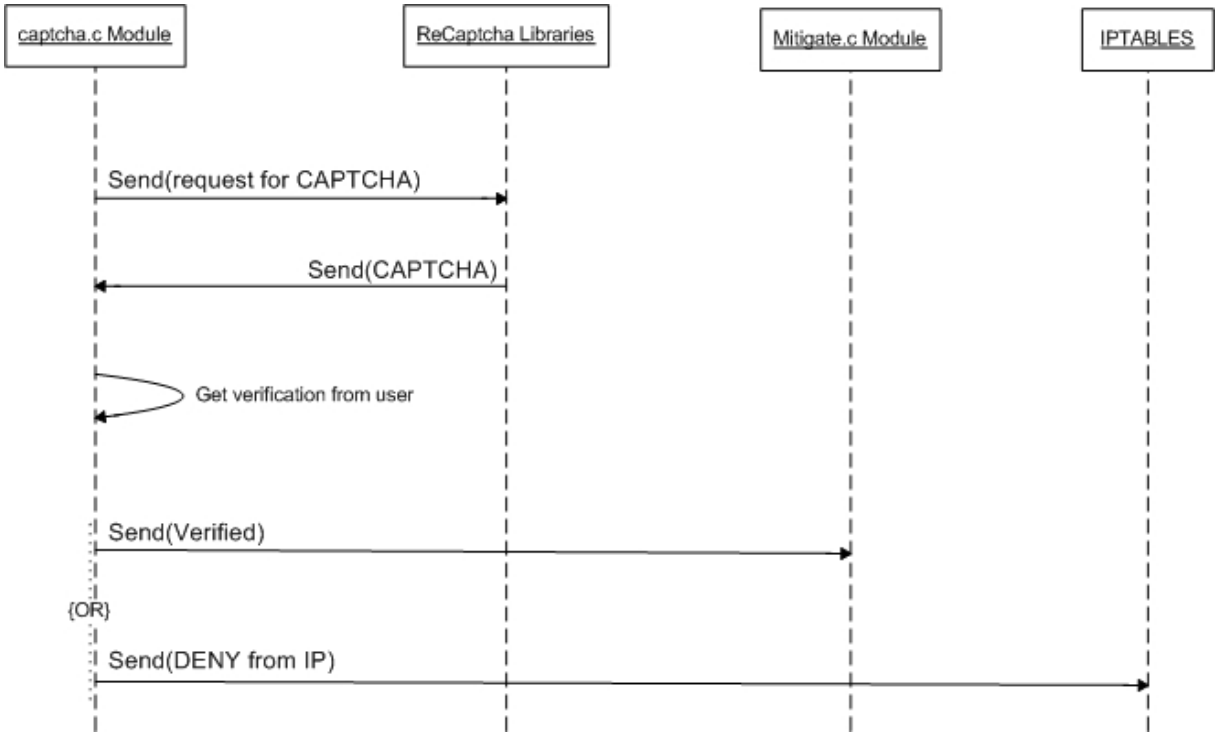

Figure 3.5: UML Diagram of Captcha Module

3.5 Design Considerations

One of the drawbacks of using this system is the problems we are faced with as a result of human error, understanding, or general willingness to actually complete the CAPTCHA form correctly. It may be that as a result of a person being asked to fill in the CAPTCHA form after so many attempts that they decide to simply leave the website altogether, causing a drop in visitor numbers. If the website utilising the CAPTCHA Solution is on an eCommerce site for example, this could have a severely negative impact on the organisations sales.

Another problem that needs to be addressed is the ability of the user to actually see the text. If the user is blind and using special accessibility software to allow them to visit websites using other methods, normally a CAPTCHA could prevent them going any further. Thankfully, ReCAPTCHA offer support for people with visual

disabilities in the form of audio clips. This presents the user with an audible sentence to type in and complete the verification process.

One of the problems with using a human verification system is when activated a search engine spider like "GoogleBot" will not be able to access the website. This will have a dramatic effect on the website's search engine ranking. To resolve this problem, it would be easy to simply create a list that contains all the IP addresses that should never be subjected to a human verification check. Using Google as an example search engine, it became apparent that this method would not work.

As it transpires, Google simply do not publish a list of IP addresses associated with their spiders; and for good reasons. They may wish to change their IP addresses in future, or add new IP addresses to the list. Publishing a list would require constant updating, and this is simply not a feasible option. It is imperative that a method of allowing these search engines unrestricted access to the server so as to preserve the website ranking, so an alternative method must be designed.

The simple method chosen is to do a few DNS tests on the IP address. All of Google's search spiders have a DNS entry corresponding to the "googlebot.com" domain name, Figure 3.6 shows this. It is not enough to simply have a reverse DNS check (i.e. checking what domain the IP address points to) as is it simple for a *spoofer* to just create an A record for the IP address. Instead, a forward lookup (i.e. checking what IP address the domain points to) will guarantee the validity of the search engine (as only the owner of the domain can point the record to an IP address) (Anon., 2006. *How to verify Googlebot*).

To summarise, when the request is received by the server, the server should do a check to see if the associated DNS entry is in an "allowed.hostnames" file. This way, all known search engines that comply with reverse DNS, will always be allowed through the system without being stopped by the CAPTCHA system. A downside to this is that for every request a lot of system resources are used; reading the IP address, checking the DNS, reading the "allowed.hostnames" file, etc.

```
> host 66.249.66.1
1.66.249.66.in-addr.arpa domain name pointer crawl-66-249-66-
1.googlebot.com
```

Figure 3.6: One of Google's IPs and Reverse DNS

4 Implementation

4.1 Introduction

This chapter outlines the main infrastructure of the servers and any other hardware that is used as a platform for the test server and the test hosts, along with the implementation of the custom Apache module, and the implementation of the CAPTCHA/PHP-based module.

4.2 Infrastructure

This section is sub-divided into two sections, Hardware and Software. Whilst analysing the problem and researching potential ways to implement the system, it became clear that the Apache Web Server with the Mitigate Module, and the PHP based CAPTCHA module would not necessarily have to be residing on the same server. As a result, the infrastructure could be implemented with multiple servers. One server could process the general website requests whilst the other could process the verification requests.

4.2.1 Hardware and Software

For this project, 2 computers were used. The first computer (Computer A) hosts the websites, the apache module, and the PHP module. The second computer (Computer B) acts as individual hosts on a network. It generates TCP traffic on port 80 that closely resembles genuine and automated requests for a website. Computer A is a VPS server hosted at VPS.net. The benefits of using this kind of virtual environment is that if required the system can be scaled up and down as necessary without having to physically change around the hardware. Tables 4.1 and 4.2 show an overview of the various hardware and software used as a platform for the development and implementation of the prototype system.

Table 4.1: Hardware architecture used in the implementation and evaluation

Computer A (1 node)	
Dedicated Processor	0.4GHz
RAM	256MB
Storage	10GB
NIC	100mbps
Bandwidth	250GB/month

Computer B (1 node)	
Dedicated Processor	1.4GHz
RAM	128MB
Storage	20GB
NIC	100mbps
Bandwidth	N/A

Table 4.2: Software architecture used in the implementation and evaluation

Computer A (1 node)	
Operating System	CentOS 5.4
Web Server:	Apache 2.0.63
Database:	MySQL 5.0.81
PHP:	PHP 5.2.11
Resource Monitoring:	SysStat 9.0.6

Computer B (1 node)	
Operating System	CentOS 5.4
Traffic Generation	Wbox
Browser	Firefox

4.3 Apache Mitigate.c Module Implementation

In developing an Apache module, we need to make use of the readily available Apache API framework which allows us to communicate efficiently between the web server software and the module. This API holds a library of various methods and functions that we can call at any time during the module when required. Code Snippet 4.1 shows the Apache libraries being called at the start of the module (Threebit, 2003). Next the variables are defined that are required to change during the configuration, testing and evaluation stage of the prototype (Code Snippet 4.2). The rest of the code which is written in C is called upon when a visitor requests a site from Apache (see Appendix 6).

```
#include "httpd.h"
#include "http_config.h"
#include "http_core.h"
#include "http_log.h"
#include "http_protocol.h"
```

Code Snippet 4.1: Apache API Inclusion

```
#define REQUESTS_PER_MINUTE 3
#define CAPTCHA_LOCATION   "http://uni.considerit.co.uk/Captcha.php"
#define ALLOWED_FILE        "allowed.hosts"
```

Code Snippet 4.2: Variables being defined

4.3.1 HTTP Requests

When a request is sent to the server, we already have a list of clients open as a linked list. The visiting client's IP address (using the available library from the included Apache API: ap_get_remote_host) and the time is checked against the client list to see if we already have that hostname in the list. If we do, we increment the list and check to see if the client has exceeded the number of requests per minute limit. Code Snippet 4.3 below shows this process.

Once the check has been completed to see if the client has exceeded the threshold, we then check the stored "allowed.hosts" list on the hard disk. It may seem strange at first to check the allowed IP addresses list before checking to see if the client has exceeded the threshold. The reason for this is that the list is store on the disk, and not in memory.

For the request per minute client list, this is stored in memory and so is quickly read and written to in comparison with the hard disk list. Therefore, we only check the

stored list when absolutely certain we need to (i.e. the visitor is about to be forwarded to the captcha.php module).

If the client is not in the allowed.hosts list, we then redirect them using a HTTP_TEMPORARY_REDIRECT (that is, a 307 Temporary Redirect in Appendix X) and leave the module.

```
hostname = ap_get_remote_host(r->connection, r->per_dir_config, REMOTE_NAME, NULL);

  if(cl = client_exists(hostname))
  {
    if(!cl->done)
    {
      if(++cl->num == REQUESTS_PER_MINUTE)
      {
        if(apr_table_get(allowed, hostname))
          return DECLINED;

        r->content_type = "text/html";
        apr_table_set(r->headers_out, "Location", CAPTCHA_LOCATION);

        apr_table_set(allowed, hostname, "1");

        fprintf(log, "%s\n", hostname);

        return HTTP_TEMPORARY_REDIRECT;
      }
    }
  }
```

Code Snippet 4.3: Checking incoming HTTP requests

4.3.2 Client Exists

If the client is in fact in the request per minutes list (in memory) and therefore the list exists, we add them to the list immediately. Code snippet 4.4 shows the code that adds the client to the open list. The current time is taken from the server, and if the current request subtracted from the last_request is greater than 59, we purge the list. This removes any old records for that IP address that are outwith the range of our thresholds.

```
  current = time(NULL);
  if(current - last_req > 59)
    client_purge();
  last_req = current;
  return DECLINED;
```

Code Snippet 4.4: Add client to the linked list

4.4 Captcha.php Module Implementation

The Captcha.php module is relatively straightforward in that all it does is call the ReCAPTCHA "applet" and display it in a browser for the user to process, it then counts the number of attempts made and either sends a success back or firewalls the IP address. The Captcha.php program is minimal by design, as if it were a large file; a DDoS would still overload the server by way of having to process a large Captcha.php file. As a result, the prototype for the Captcha.php part of the project is

less than 10KB in size and there is no back-end database (MySQL) to read/write to, thus limiting resource usage further.

The system includes the ReCAPTCHA source from the reCAPTCHAlib.php library files. This file allows the system to talk directly with the ReCAPTCHA servers, and actually pulls the ReCAPTCHA applet using an iframe. By using an iframe, we let the client's browser do the transfer with ReCAPTCHA directly, saving even more resources. The snippet of the *parsed* Captcha.php code in code snippet 4.5 clearly shows the iframe inclusion pulling directly from **http://api.recaptcha.net**

```
<iframe src="http://api.recaptcha.net/noscript?k=6Lf3GAkAAAAAALlC4PzWSaOQzPhjgu-
6EGyirjbx" height="300" width="500" frameborder="0"></iframe>
```

Code Snippet 4.5: Include reCAPTCHA iframe

In code snippet 4.6 we see the original source of Captcha.php, before being parsed by the web server. The raw PHP code is setting up the session and including the library that is used to produce the iframe:

```
session_start();
require_once('recaptchalib.php');
```

Code Snippet 4.6: Start session and include reCAPTCHA library

A session in PHP is a way of supporting data in variable form, but at the servers end. Once a session has started, we can create variables about that visitor and the session will remain (server side) until we delete it. As a result, we can use PHP's sessions function to hold the number of attempts a user has taken to try and get the CAPTCHA correct. In code snippet 4.7, we can see the use of the session function. The first line instantiates the recaptcha_attempts variable by 1. The code then goes on to check if the session is equal to three (see Appendix 7 for more of this PHP code). If it is, we can deny the user (as they have given three unsuccessful attempts at answering the CAPTCHA).

In this prototype, the captcha.php module simply denies the user using .htaccess. In a real-world scenario, we would need to look at denying the IP address either using iptables, or at a border gateway.

```
$_SESSION['recaptcha_attempts']++;

if($_SESSION['recaptcha_attempts'] == 3)
{
  $ip = $_SERVER['REMOTE_ADDR'];
  file_put_contents('.htaccess', "deny from $ip\n", FILE_APPEND);
```

Code Snippet 4.7: If statement to check whether IP should be denied

4.5 IP Address filtering

If an IP address is spoofed, that is the source address if forged, then the system will still suffer a DDoS attack. There is no way that this prototype would be able to defend a server against an attack that makes use of spoofed IP addresses.

There is a sustained argument, however, that all ISPs should be egress filtering all packets outbound from their network. Security policies should be implemented that prevent the ISP from allowing IP addresses that could not have originated from within the network from leaving their network. Simply put, if an ISP owns and uses a range of 62.31.64.0 – 62.31.64.255 (62.31.64.0/24), an IP address of 112.114.23.21 should never be able to send data from within that network out onto the Internet. This argument could be taken a step further to suggest that all ISPs are responsible for egress filtering all packets that are suspected to be participating in a DDoS attack. Unfortunately, not all ISPs do this kind of filtering and spoofing is potentially a real problem scenario.

4.6 Conclusions

After several unsuccessful attempts, the module compiles and Apache loads the module without errors. The CAPTCHA module was verified to be working by sending a request from a browser directly to it and monitoring the results after three failed submissions. As both of these work to this degree, the evaluation of the development can take place. It is too early, at this stage, to say whether the aim of successfully developing a working prototype has been achieved.

5 Evaluation

5.1 Introduction

It is important at this stage to now evaluate the prototype that has been implemented to ensure that the objectives have been met, to identify any problems or weaknesses in the prototype, and to ensure the prototype is actually working and a success. The evaluation is broken up into sub-sections, with each of these subjects broken up further. These further sections are the findings from the testing phase of the project, the effectiveness of the prototype in those situations, and conclusions about that particular environment and how the prototype works within it.

The first aim of this project was to research what solutions are currently available to detect and mitigate an attack, and how an attack can be detected in general. The second aim of this project was to produce a working prototype that might be effective in mitigating an ongoing Distributed Denial of Service attack. This section of the project now deals with testing and evaluation of the prototype in differing situations and it attempts to find weaknesses in the prototype.

5.2 Test Controls

Before carrying out any testing or evaluation, it is imperative that controls are set up so that we can guarantee the results. Apache normally runs with multiple server processes. As a result, it is going to be difficult to target the exact CPU, Memory, and other resource usage that our module might utilise. We could take an average resource usage over all processes, but for the testing results to be as accurate as possible, we will reconfigure Apache slightly. For all tests, Apache will have its configuration changed so that it only runs one process ever. In a real environment, this will restrict the number of visitors, but will allow us to accurately capture the resource usage of the prototype and the effectiveness of it when Apache only runs on one process.

For the duration of the experiments, the hardware used will be constant. Although possible, there will be no scaling of Server A throughout the evaluation process. Lastly, the page requested will be a constant size of 20KB.

5.3 Individual Host Tests

Testing and evaluation of the prototype when only one IP address is querying the server for a web site is shown in this section. Server B requested the website address X times per minute. Various thresholds will also be analysed. Thresholds are the amount of requests per minute a user can make before being forwarded to Captcha.php.

To gather statistics on the resource usage of the protocol, we initially compile Apache without the module (therefore no DDoS mitigation). The tests are performed, and then the tests are performed again with the module compiled and loaded in Apache. The following variables and constants were recorded during the testing stages:

Apache Module Loaded: Yes, No

Number of Requests sent from Computer B per minute: 1, 3, 5, 10, 50

Number of Requests per Minute allowed at Computer A: 1, 3, 5, 10

The results were generated using SysStat on Server A and exported to a CSV file for analysis.

5.4 Multiple Concurrent Hosts

The following testing was done using Computer B with multiple IP addresses bound to the same network card. By doing this, the requests can appear to be coming from multiple different hosts and are received by their IP address. This saves on having to set up multiple machines to act as compromised computers. Figure 5.1 shows an example of the wbox software sending a request from a bound IP on the network adaptor to the web server and then during the fifth request (number 4) is redirected to the CAPTCHA page using a 307 temporary redirect (see Appendix 8).

```
ServerB:~/stuart/wbox # ./wbox uni.considerit.co.uk
WBOX uni.considerit.co.uk (109.123.66.58) port 80
0. 200 OK    20,480 bytes    54 ms
1. 200 OK    20,480 bytes    58 ms
2. 200 OK    20,480 bytes    48 ms
3. 200 OK    20,480 bytes    72 ms
4. 307 Temporary Redirect    (801) bytes    48 ms
```

Figure 5.1: wbox software sending multiple GET requests

By sending the request from multiple hosts, we cause the Apache module to increase the memory footprint for storing the individual IP addresses of each visitor to the site(s). As a result of these changes in the memory, the software might work unpredictably or might not work in mitigating a DDoS attack.

5.5 Results Analysis

These results were generated using freely available linux software: SysStat. SysStat was configured to monitor only the Process ID that Apache was running on (Apache was also configured to only ever run with one process and no child threads).

The base line of Table 5.1 shows that only a small footprint of CPU time and Memory is needed to satisfy the requests on a standard deployment of Apache with only one host requesting the page.

Module not loaded with one host sending a request:

Table 5.1: Module not loaded with one host sending a request every X minutes.

Sending Rate	CPU %	Memory %
1/min	0.1	1
3/min	0.1	1
5/min	0.1	1
10/min	0.4	1
50/min	0.5	1.1

The results for one host sending a GET request X times per minute when the limit is above this threshold are inconclusive. There is no delay in responding to the request, and there is no abnormal server load (i.e. when five GET requests are sent per minute and the Module's limit is 10 per minute).

Figure 5.2 shows the effects on the server and the CPU % and Memory % usage as a result when the module is forced to forward the request on to the captcha module.

```
Module Loaded
1 Host Sending GET 10 times per minute when module limit 5 per
minute

ServerB:~/stuart/wbox # ./wbox uni.considerit.co.uk
WBOX uni.considerit.co.uk (109.123.66.58) port 80
0. 200 OK    20,480 bytes    54 ms
1. 200 OK    20,480 bytes    58 ms
2. 200 OK    20,480 bytes    48 ms
3. 200 OK    20,480 bytes    48 ms
4. 307 Temporary Redirect    (801) bytes    75 ms
5. 307 Temporary Redirect    (801) bytes    75 ms
6. 307 Temporary Redirect    (801) bytes    75 ms
7. 403 Forbidden    813 bytes    72 ms
8. 403 Forbidden    813 bytes    72 ms
9. 403 Forbidden    813 bytes    72 ms

CPU % Usage at 3: 0.1%
RAM % Usage at 3: 1%

CPU % Usage at 4: 0.3%
RAM % Usage at 4: 1%

CPU % Usage at 7: 0.1%
RAM % Usage at 7: 1%
```

Figure 5.2: 1 Host Sending GET 10/min when Module Limit 5/min

It is obvious that there is a slightly higher delay when transferring the request to the captcha page. Normal response is in the region of 48 milliseconds, but when the server has to process a different set of instructions (that is, the IP address should be forwarded to captcha.php) the time it takes to receive the Temporary Redirect is slightly longer. This time delay does not appear to be excessive however. Figure 5.3 shows multiple hosts sending GET request with the same settings.

The IP address range used is 192.168.0.10 through 192.168.0.50 (i.e. 40 "independent" hosts sending GET requests). Each uses its own version of wbox to bind to an IP

address. The results for IP address 192.168.0.15 are shown in Figure 5.3 below. Each wbox process was started at the same time using a pre-written batch script.

It is interesting to note here that when the server is under stress from multiple hosts requesting, the Temporary Redirect that should be set on the 5th request within a minute is delayed until the 8th request (number 7 in Figure 5.3).

```
Module Loaded
192.168.0.15 Sending GET 10 times per minute when module limit 5
per minute (whilst other hosts send same requests)

ServerB:~/stuart/wbox # ./wbox uni.considerit.co.uk
WBOX uni.considerit.co.uk (109.123.66.58) port 80
0. 200 OK    20,480 bytes    54 ms
1. 200 OK    20,480 bytes    68 ms
2. 200 OK    20,480 bytes    68 ms
3. 200 OK    20,480 bytes    73 ms
4. 200 OK    20,480 bytes    68 ms
5. 200 OK    20,480 bytes    48 ms
6. 200 OK    20,480 bytes    84 ms
7. 307 Temporary Redirect    (801) bytes    86 ms
8. 307 Temporary Redirect    (801) bytes    86 ms
9. 307 Temporary Redirect    (801) bytes    86 ms
10. 403 Forbidden    813 bytes    70 ms
11. 403 Forbidden    813 bytes    70 ms
```

Figure 5.3: Results from stress testing

As a result of this delay, there may be underlying problems with using this prototype as a successful way of mitigating a large scale attack. However, the prototype does indeed redirect the user, albeit slower than anticipated, so in a way it could be claimed a success. The user is successfully redirected to the CAPTCHA module (at point 7 in Figure 5.3) and then after three unsuccessful attempts at CAPTCHA, is denied at the software level.

5.6 Conclusions

Generally, the module is successful in dealing with a DDoS attack. It may be that during a large scale attack, the module needs time to catch up and filter the bad traffic from the good. With load balancing in place, this module should work well. It is important to remember that the server that was used in the implementation and evaluation stages of the thesis was a very low power machine and had limited resources available. In a real scenario, the prototype would (hopefully) be running on a much more powerful server that should be able to sustain such a demand from the module.

6 Conclusions

6.1 Thesis Conclusions

Distributed Denial of Service attacks are getting more and more advanced as a result of sophisticated malware being released into the wild. As a result, new methods of mitigating these attacks need to be designed and developed. This thesis aimed to read similar related work done and gain an in-depth examination of DDoS taxonomy, bot nets, malware and mitigation methods. The thesis also attempted to take this research and use it in a prototype system that could potentially be used in a real environment to detect and mitigate an active DDoS attack on a server.

The thesis makes use of some recent papers to justify the claims made and to verify the statements. There appears to be a large number of IT professionals, like McMillan (McMillan, R. 2009), who appreciate the real threats and risks of DDoS on the Internet, and conversely there are those who do not. Whether people believe the threat exists is irrelevant. There are businesses online at this moment in time suffering as a result of being attacked by a DDoS botnet and losing sales and customers as a result.

The design methodology of the prototype was loosely followed using the waterfall model. Sequentially working through the requirements and design stages to the implementation and then the verification, evaluation and testing stages. It may have been beneficial to look at other ways to carry out the prototype as if predictions made in the design stage were inaccurate then the end result during the implementation stage could have been a failure.

The testing methodology used during the evaluation stage of the thesis was White Box testing. An understanding of the prototype was needed to allow test cases and experiments to be set up which allowed identification of the prototypes flaws and weaknesses. Testing between the two modules was also done in this way (307 Temporary Redirect and then 403 Forbidden) to ensure that the expected result was actually achieved.

6.2 Achievement of aims and objectives

The first aim of the thesis was to analyse current DDoS taxonomies and attack signatures. From looking at real data and malware like NetSky, the thesis can claim success within this aim. This thesis covers various DDOS taxonomies and shows a number of methods to mitigate ongoing attacks. The literature review details previous work done in the subject area and ways in which people have managed to successfully mitigate DDoS in real life scenarios.

The thesis shows successful development of a working prototype that detects and mitigates a web-based (or HTTP-based) DDoS attack. The prototype was tested and confirmed to be working well at a low load. The aim of evaluating and testing the prototype to determine the effectiveness of it has not been achieved.

The prototype shows signs of resource use at low levels (i.e. when 40 hosts are requesting a page at the same time). In a real world situation, the expected number of hosts requesting a page per minute can be in excess of 100,000. In a situation like this, the prototype would fail. It could be argued that although the prototype does not work, the evaluation does show a true likeness to what would happen should a true DDoS attack be inflicted on the prototype, and how it would perform during such an attack.

6.3 Critical Analysis/Self appraisal

I believe that this system, although slightly inefficient at high load levels, could potentially help to mitigate a medium-scale DDoS attack on a website. The prototype does detect a user that exceeds the threshold set, and it does then forward them on for verification. The prototype also then creates entries that simulate the IP address of that user being blocked at a firewall level. However, the prototype fails to appreciate any false positives that may occur. If a user was to exceed the threshold and then fail validation, their IP address would be firewalled on the server permanently (or until an Administrator flushed the firewall). As a result, my prototype fails to take into consideration the real ways humans interact with computers and does not achieve the best results it could.

I failed to appreciate the time it would take to learn and develop a module in C. It took a considerable amount of time to learn how C works, and even with my understanding of PHP, C was difficult to learn. As a result of the time it took to get the module to compile, parts of the prototype sometimes do not work. Sometimes it takes longer for the prototype to actually catch a host over the threshold and direct them to the CAPTCHA module (this is usually only when a high load is put on the server by way of many multiple concurrent hosts).

From my Distributed Systems module, I learned the fundamentals that allowed me to design the system in a way that the CAPTCHA module and the apache web server did not have to reside on the same machine. Using HTTP protocols, the CAPTCHA module could reside anywhere on the network, or even the Internet. My Linux module taught in third year helped me quickly grasp the fundamentals of setting up the server in the first instance.

If I were to go back to the design stage and start development again, I would take more time to work out the Apache module. The module was written in no particular order, and is exceptionally messy. It is a wonder it actually compiles, let alone runs successfully and passes data onto the CAPTCHA module. Considerations that were taken in the design stage (for example, reading the allowed.list every time a host visits the site) were in fact changed in the module (the allowed.list is only checked

when the threshold is reached) so that resources (in the example's case, the hard disk) could be conserved but more importantly, so the module was much faster to respond (reading from the hard disk is a slow process I ultimately realised).

6.4 Recommendations and Future Work

It would be recommended to thoroughly test the Apache module (mitigate.c) and the PHP module (captcha.php) on different servers to verify that it works as a distributed system. The prototype was originally designed this way as a rudimentary form of load balancing for the system – verification of human beings would not have to be done on the same server that was serving normal websites. As it transpires, time was a factor when actually being able to do this part and although it was not an aim of the thesis, it would have been good to verify this part of the prototype worked successfully.

The prototype does however work as an integrated system on one server. The deployment of the modules is simple and highly configurable. It should be noted that the prototype does not currently add to any Linux firewall IPTABLES, but only creates a deny in the .htaccess file. This would be a recommendation of future work with the prototype. It would be beneficial to be able to create a form of handler that could take the output from PHP and create rules on a Linux firewall (or even a gateway router-firewall) so that actual packet dropping could occur. The easiest way to do this would probably be from some sort of CRON job running every minute reading a list the captcha.php module creates as a result of failed human verifications. This list could then be fed to a Cisco PIX firewall or other device for updating.

As a result of the literature review of this thesis, it is apparent that server-side or source-side mitigation (that is, at the end of the victim) are of minimal use. They can be claimed a success, but the Internet still suffers as a whole due to the significant increase in traffic from the DDoS attack. Ideally, a prototype that is light-weight, open-source (for community-based updates) and free, could be deployed at the compromised computer's ISPs (or all ISPs) to prevent a DDoS attack ever leaving their network in the first place. In an ideal world of course, users would keep their operating systems up to date with security patches and virus definitions to prevent the malware that creates these botnets from ever infecting the computers in the first place.

References

Franklin, J., Paxson, V., Perrig, A., & Savage, S.(2007). An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants

Hoeflin, D., Karasaridis, A., & Rexroad, B. (2007). Wide-scale Botnet Detection and Characterization

CERT. (2001, June 4). *Denial of Service.* Retrieved November 19, 2009 from http://www.cert.org/tech_tips/denial_of_service.html

Mavrommatis, P., Polychronakis, M., & Provos, N. (2008). Ghost turns Zombie: Exploring the Life Cycle of Web-based Malware

Lee, R., & Speecht, S. (2004). Distributed Denial of Service: Taxonomies of Attacks, Tools and Countermeasures

Martin, J., Mirkovic, J., & Reiher, P. (2002). A Taxonomy of DDoS Attack and DDoS Defense Mechanisms.

Kessler, G. (2000, November). *Defenses Against Distributed Denial of Service Attacks*. Retrieved 10 May, 2009 from http://www.garykessler.net/library/ddos.html

Leyden, J. (2004, September 23). *US credit card firm fights DDoS attack.* Retrieved 20 September, 2009 from http://www.theregister.co.uk/2004/09/23/authorize_ddos_attack/

Labovitz, C. (2009, August 6). *Where Did All the Tweets Go?* Retrieved 10 August, 2009 from http://asert.arbornetworks.com/2009/08/where-did-all-the-tweets-go/

Dulay, N., Sloman, M., & Thing, V. (2006?). A Survey of Bots Used for Distributed Denial of Service Attacks.

Dagon, D., Dwivedi, S., Grizzard, J., Gu, G., Lee, W., Lipton, R., & Zou, C. (2005). A Taxonomy of Botnets.

Mirkovic, J., Prier, G., & Reiher, P. (2002). Attacking DDoS at the Source

Wiens, J. (2004). Sticky Worms or: Responding to a Distributed Denial of Service Attack with a Selective TARPIT.

Stewart, J. (2007, June 25). *HTTP DDoS Attack Mitigation Using Tarpitting.* Retrieved 20 August 2009 from http://www.secureworks.com/research/threats/ddos/

Cisco Systems Inc. (2004). *Defeating DDOS Attacks.* Retrieved 10 September 2009 from http://www.cisco.com/en/US/prod/collateral/vpndevc/ps5879/ps6264/ps5888/prod_white_paper0900aecd8011e927.html

Edward, P. (2007, October 19). *CAPTCHA*. Retrieved 13 August 2009 from http://tarr.uspto.gov/servlet/tarr?regser=serial&entry=78500434

Anon. (2006, September 20). *How to verify Googlebot*. Retrieved 2 May 2009 from http://googlewebmastercentral.blogspot.com/2006/09/how-to-verify-googlebot.html

Jasper. (2009, April 14). Ultralight http/ftp server. Retrieved 19 September 2009 from http://sourceforge.net/projects/httpdx/

West, M. (2008). Threats That Computer Botnets Pose to International Business

Valdman, J. (2001) Log File Analysis

Ballani, H., & Francis, P. (2007). A Simple Approach to DNS DoS Mitigation

Champagne, D., & Ruby, L. (2006). Scope of DDoS Countermeasures: Taxonomy of Proposed Solutions and Design Goals for Real-World Deployment

Northcutt, S., & Novak, J. (2002). Network Intrusion Detection

Molsa, J. (2005). Mitigating denial of service attacks: A tutorial

Chen, W., He, Y., Sha, E., & Xiao, B. (2005). An Active Detecting Method Against SYN Flooding Attack

Threebit. (2003). *Introduction to Module Development.* Retrieved 8 August 2009 from http://threebit.net/tutorials/apache2_modules/tut1/tutorial1.html

Zaytsev, V. (2009, October 6). *W32/Xpaj Botnet Growing Rapidly.* Retrieved 20 November 2009 from http://www.avertlabs.com/research/blog/index.php/2009/10/06/w32xpaj-botnet-growing-rapidly/

Zhang, W. (2000). Linux Virtual Server for Scalable Network Services

Chi, K-H., Yeh, T-T., & Yen, L-H. (2009) Load Balancing in IEEE 802.11 Networks

McMillan, R. (2009, November 13) *DNS Problem Linked to DDoS Attacks Gets Worse.* Retrieved on 24 November 2009 from http://tech.yahoo.com/news/pcworld/20091113/tc_pcworld/dnsproblemlinkedtoddosattacksgetsworse

Appendix 1: Project Overview

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Title: Analysis and Development of a Prototype for the Detection and Mitigation of HTTP Based Distributed Denial of Service Attacks

Overview of Project Content and Milestones

There is a problem on the Internet that causes many ISPs, clients and website owners unavoidable downtime. Swarms of compromised machines join together to attack legitimate websites or services to bring them offline or disrupt them for a period of time. The motives for this can be anything from extortion to simply a disgruntled person. This project aims to develop a prototype to detect and mitigate these kinds of web-based attacks and look at the current malware, bot nets, and mitigation options available at the moment.

The Main Deliverables

The main deliverables of this thesis are to identify current threats from bot nets online, their signatures, how they operate and what methods can be used to detect and mitigate these attacks. Lastly, a working prototype for detecting and mitigating such attacks will be designed, implemented and evaluated. The thesis will also establish what unique methods network professionals have implemented their methods of mitigating an ongoing attack.

The Target Audience for the Deliverables

The target audience for this thesis is network administrators and networking professionals in small business networking organisations. The problem of a distributed denial of service attack is a severe risk and any new working methods that mitigate such an attack will be of interest to these people.

The Work to be Undertaken

The work to be undertaken consists of:

- Analysis of current DDoS mitigation methods
- Analysis of current DDoS detection systems
- Research in current DDoS methodologies
- Design prototype for detection and mitigation of DDoS
- Implementation of DDoS Mitigation Prototype
- Testing and Evaluation of DDoS prototype.

Additional Information/Knowledge Required

It is important to keep up with the industry in terms of the current malware used, research in the field is important before beginning the design stages of a prototype to detect and mitigate DDoS attacks.

Research of hardware and software solutions that can be used to help build a platform for the prototype, and the hardware and software to build the prototype itself.

Information Sources that Provide a Context for the Project

Research papers, News websites (DDoS attacks are usually picked up by the media), Industry websites (IEEE, and so on), software vendors, hardware vendors, and businesses.

The Importance of the Project

Businesses are under threat as a result of the Internet being misused by others. It is important to continually test and evaluate potential solutions to mitigate attacks from these attackers.

The Key Challenge(s) to be Overcome

Understanding how DDoS works by looking at the taxonomy of it and the malware/bot networks used. Understanding why they exist, and who is responsible. Research what each one has in common with the other – look at signatures of the attacks, analyse these. Developing a system that actually mitigates an attack to some degree.

Appendix 2: Second Formal Review Output

A copy of this document is held at the School of Computing office in Merchiston Campus.

Appendix 3: Diary Sheets

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 04/03/2009 **Last diary date: N/A**

Objectives:

- Changed project from CMS/CRM based software development to an investigation into DDoS detection and mitigation.
- Read paper on Russia mafia//crime syndicates involvements in DDoS

Progress:

Looked into how DDoS works and what benefits the owner of the bots actually receives from initiating an attack.

- Disgruntled Employee attacking employer site
- Competitor attacking the site to bring more custom to their own
- Black Mail – demanding payments for the bot net to stop
- Boredom – children/teenagers taking advantage of the Internet's open-ness

Dealt with live bot-net attacking a web-server in April/May. Logs printed as evidence. Traffic graphs, access logs, and location of all attackers.

Supervisor's Comments:

Look for papers related to the classification of DOS threats (Taxonomy of DDoS Attacks and DDoS Defence Mechanisms)

Look also for analysis techniques and for defence mechanisms

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 15/04/2009 **Last diary date: 04/03/2009**

Objectives:

- Look for more research papers that involve DDoS taxonomies and methods of mitigation. Avoid looking at white papers; these are heavily biased towards the manufacturer.

Progress:

Read more research papers. Signature of NetSky and DDoS from the Inside.

Looked at ReCaptcha as a possible solution for integrating into the final prototype.

Supervisor's Comments:

The taxonomy papers are important in detecting and classifying the threats. Try and use these detection/classifications to structure parts of your literature review, such as: Attack methods, Propagation techniques, and so on..

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 07/05/2009 **Last diary date: 15/04/2009**

Objectives:

- Research where the main problems lie in DDoS
- Methods involved in preventing DDoS
- Netsky-x,y,z Bots
- Upstream Saturation Investigation

Progress:

Research concludes that the main problems involved with DDoS are:

- Upstream Saturation of the ISP
- Server Daemon being overloaded with requests
- Other services on the same server being brought down as a result

Netsky bot virus/worm has back-door access to allow the owner to change settings/run commands. There is a consensus amongst IT pro's that this could be a way to stop DDoS from these bots ("Hack Back" to the client). *Respondign to a DDoS attack with a selective tarpit.*

Supervisor's Comments:

Have a search for a BotNet architecture and how they operate.

Also look for "Large-scale" attacks such as for Google/Ebay/etc.

Make sure you are up to date on the latest bots.

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 11/05/2009 **Last diary date: 07/05/2009**

Objectives:

- Outline project scope clearly
- Investigate recent DDoS Threat (Cornficker)

Progress:

Research Project Scope Outline:

- Research different bot nets and different types of DDoS attack
- Reasons for DDoS attacks, why, who, where.
- Detecting a DDoS attack. Algorithms for the main ones?
- Mitigating a DDoS attack:
    1) Cisco Firewalls?
    2) Linux IPTables Firewalls?
    3) Apache, Lighthttp, Nginx for HTTP attacks?
    4) Other options/mix of solutions? Customised software?
    5) Snort? Ethereal? Etc.
- Load Balancing options.

Supervisor's Comments:

A big part of your project is to define what you will DESIGN, what you will IMPLEMENT and then how you will EVALUATE it.

Have a think about the evaluation tools and experiment at this stage.

Look at Host Vs Network detection.

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 18/05/2009 **Last diary date: 11/05/2009**

Objectives:

- Legal Implications – developing world, allies, etc.
- User willingness to open attachments
- Windows XP SP2 – Efforts to prevent it at the client-side.

Progress:

It appears that DDoS attacks originate from countries whose IT infrastructure is still in a development stage. It is a possibility that these countries where the attacks originate do not have proper solutions to inspect the egress traffic as it leaves their networks.

Willingness to open attachments in emails seems to be the main method for infection. Security holes and other weaknesses in the operating systems are not necessarily at fault. The nature of computers makes it very difficult to prevent clients from being “hijacked”.

Windows XP SP2 has additional features to throttle TCP packets when leaving the local computer. This should help to limit the effects a DDoS has on a wide-scale basis. Question: How effective is this really going to be? Is there perhaps a link between upgrading your Service Pack and opening unknown attachments anyway?

Supervisor’s Comments:

Start developing and/or finding suitable evaluation tools, such as to simulate real traces of botnets.

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 22/05/2009 **Last diary date: 18/05/2009**

Objectives:

- Analyse the real log files and traffic graphs from the latest DDoS attack.
- Look at options for actually creating a test environment for preventing DDoS (UK2 may be willing to sponsor it via VPS.net)
- Interview large corporate bodies in relation to DDoS and what steps they currently take if/when they come under attack.

Progress:

Looked at bot net activity and read research papers in relation to Florida University and their huge DDoS attack.

Will speak to UK2.net about how they manage DDoS.

Supervisor's Comments:

Good to see that you have real-world traffic. Start to think about analysing it and determining the patern of operation.

After this, try to develop a Bot simulator which gives you a basic footprint to detect.

Think about the success rate of your IDS such as false positives, response times and so on.

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 22/05/2009 **Last diary date: 22/05/2009**

Objectives: **2nd Meeting: ONGOING**

- Log at log analysis techniques
- Look at gathering log files for analysis

Progress:

Read research papers

Developed written plan for working schedule

Supervisor's Comments:

You need to decide whether you want to focus on an offline analyser for your detection system, or a real-time detection system.

A real-time will allow you to detect and ban a host while it is accessing the site. So this method may be preferential.

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 29/05/2009 **Last diary date: 22/05/2009**

Objectives:

- Set up VPS server for testing and evaluation of the prototype
- Set up host computer

Progress:

Analysed log files and noticed patterns in the signatures of the logs.
Read research papers on Bot Net Taxonomies

Supervisor's Comments:

Try and reflect on the basic signatures for detection and some of the problems that would occur – especially for false positives

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 18/06/2009 **Last diary date: 29/05/2009**

Objectives:

After reading more papers, complete an accurate literature review.

Progress:

Started work on the literature review. Looking at different mitigation methods on the market today – Cisco have a solution for this that may be included?

Supervisor's Comments:

There are some good taxonomy papers which you should read. Try to look widely at mitigating methods which you can discuss in your literature review and then maybe focus on one or two for your main evaluation/implementation

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 30/06/2009 **Last diary date: 18/06/2009**

Objectives:

Start looking at designing the prototype: how will you integrate it with a server? How will you receive the requests? Log file analysis?

Progress:

Partially completed literature review.
Looked at a Log File Daemon type system to read the log files every X seconds.
This could be beneficial to triggering a block if the IP address appears too many times?

Supervisor's Comments:

Good to see you have some code being developed. The speed of detection will relate to how fast you can read your logs. What happens if the logs are fairly large; will this slow down the system?

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 08/07/2009 **Last diary date: 30/06/2009**

Objectives:

How will you report and deny access to an attack? What methods are available and how do they achieve this?
Consider other implications: bandwidth consumption, other servers/services on the same network. How will they be affected?

Progress:

Log File Daemon might be too resource consuming – Investigated ways of integrating with Apache directly (Modules and API).

Supervisor's Comments:

You need to justify why you are focussing on Apache, and try and create an overall design/abstraction for your system.

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 30/07/2009 **Last diary date: 08/07/2009**

Objectives:

Learn the basics of C and putting together a module using the Apache API Code.

Progress:

Read various websites and C tutorials. Grasped an understanding of how it works and how data is stored in memory.

Supervisor's Comments:

Make sure you have tested your code fully and that it can be fully evaluated later on.

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 03/08/2009 **Last diary date: 30/07/2009**

Objectives:

Restructure the thesis document as shown. Look at including references where possible to all work done.

Look at other ways of developing the prototype, not just Apache, what about Microsoft solutions? If they are not your choice, why not? What reasons can you give?

Progress:

Looked at ways to mitigate a DDoS using a Cisco Guard XT device
Looked at mitigating DDoS using DNS and TARPIT

Supervisor's Comments:

Make sure you have other important areas in the literature review such as Bot V Human section, where you can go over other methods which can theoretically be used in a prototype and then be critically evaluated.

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 20/08/2009 **Last diary date: 03/08/2009**

Objectives:

Look at finalising the Literature Review – include an introduction and conclusion where appropriate.
Look towards structuring the design in an academic way – develop models to help a reader understand how the individual systems operate. What modules are used? UML Diagrams?

Progress:

Developed UML diagram for system overview – Apache > Mitigate Module > Captcha Module.

Developed more detailed diagrams for each individual module.

Supervisor's Comments:

Try to focus your introductions in providing a context and overall aim with the chapters. The conclusion is also important and needs to be reflective about what is being taken forward and what is being discussed.

Good to see some formal design and some detail of the design of the modules.

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 01/09/2009 **Last diary date: 20/08/2009**

Objectives:

Look at finalising the Literature Review – include an introduction and conclusion where appropriate.
Look towards structuring the design in an academic way – develop models to help a reader understand how the individual systems operate. What modules are used? UML Diagrams?

Progress:

Developed UML diagram for system overview – Apache > Mitigate Module > Captcha Module.

Developed more detailed diagrams for each individual module.

Supervisor's Comments:

You need to reflect on the other methods that could be used instead of Captcha! It is important to look widely at the other technologies – even if it's just to confirm the choice you have made is correct.

Also pull out some code snippets to support your implementation stages and explain in some detail what is going on.

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 08/09/2009 **Last diary date: 01/09/2009**

Objectives:

Narrow down the project title.
Look at completing the design stages – what areas have you found could be improved? What issues are you going to have?

The design stage might have opened up new issues, are these now addressed in the Literature Review?

Progress:

Project aim and title now specifically targeting HTTP-based attacks. Need to include Captcha and Google information in literature review to make reader aware of these.
Background information needs added to Lit Review.

Design stage shows some key problems that may occur; review of these in Lit Review will be done.

Supervisor's Comments:

Also start to think clearly about the overall aim as you will need to decide how well you've achieved. If you make it too wide, it might be difficult to justify that the end result matches the aim.

Also, think about the title and make sure it reflects the thesis contained.

Student: Stuart Gilbertson **Supervisor: Prof. Bill Buchanan**

Date: 04/10/2009 **Last diary date: 08/09/2009**

Objectives:

Look at finalising the design and start planning how to evaluate the prototype.

Does the prototype work? Does it work under stress? What situations will it work well and what situations will it not?

How quickly does the server respond without it, how quickly does it respond with it? What happens with more than one host attacking at a time (Distributed DOS).

Progress:

Looked at ways to evaluate.
Researching software solutions to help me evaluate the prototype.

Decide on the evaluation thresholds – what is a useful part to test? Why?

Supervisor's Comments:

Look at evaluation strategies such as black box, white box, and so on.

Look at defining your evaluation properly. Look at the key variables for the experiment and how you will time the server's response.

Appendix 4: DoS Vulnerability in *httpdx Web Server 1.4*

```
# httpdx Web Server 1.4 'Host Header' Remote Format String Denial of
Service PoC
#
# Coded by Pankaj Kohli
# http://www.pank4j.com
#
# httpdx web server 1.4 is vulnerable to a remote format string
# vulnerability through the Host header.
# The vulnerability lies in httpd_src/http.cpp in h_readrequest() :
# snprintf(temp[1],MAX,client->host);
#

use LWP;

(($target = $ARGV[0]) && ($port = $ARGV[1])) || die "Usage: $0 <target>
<port> \n";

my $ua = new LWP::UserAgent;
print "Connecting to $target on port $port\n";
my $request = new HTTP::Request('GET', "http://" . $target . ":" . $port);
print "Sending evil header \n";
my $host_header =  "%s"x32;
$request->header('Host', $host_header); my $response = $ua-
>request($request);

if ($response->is_success) { print "DoS Failed \n" }
else { print "DoS Successful \n" }

# milw0rm.com [2009-09-14]
```

Appendix 5: Log Files from Live DDoS Attack

```
78.163.62.149 - - [06/May/2009:09:46:35 +0100] "GET / HTTP/1.1" 200 16955 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:35 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:38 +0100] "GET / HTTP/1.1" 200 16955 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:39 +0100] "GET / HTTP/1.1" 200 16955 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:41 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:41 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:41 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:42 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:42 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:43 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:43 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:53 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:56 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:56 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:56 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:57 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:46:57 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:04 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:08 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:09 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru;
rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
```

78.163.62.149 - - [06/May/2009:09:47:12 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:16 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:16 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:17 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:17 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:18 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:18 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:21 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:21 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:26 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:26 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:27 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:27 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:30 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:32 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:36 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:41 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:41 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:41 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:50 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:53 +0100] "GET / HTTP/1.1" 500 358 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"
78.163.62.149 - - [06/May/2009:09:47:54 +0100] "GET / HTTP/1.1" 301 237 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; ru; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1"

Appendix 6: Source Code for Apache Module

```
/* These includes let us access the HTTPd API */
#include "httpd.h"
#include "http_config.h"
#include "http_core.h"
#include "http_log.h"
#include "http_protocol.h"

#include <time.h>

/* Some variable defines that can be changed */
#define REQUESTS_PER_MINUTE 3
#define CAPTCHA_LOCATION    "http://uni.considerit.co.uk/Captcha.php"
#define ALLOWED_FILE        "/usr/local/apache/allowed.hosts"

/* Declare module name to httpd core (Tutorial 2 of
http://threebit.net/tutorials/apache2_modules/tut2/tutorial2.html */
module AP_MODULE_DECLARE_DATA mitigate_module;

/* Linked list for managing client connection data. */
struct client
{
  int            num, done;
  time_t         begin_time;
  char          *name;
  struct client *prev, *next;
};

struct client *clients;
struct client *last;
apr_table_t *allowed;
time_t last_req = 0;

/* Add a client. */
void client_add(const char *hostname)
{
  /* If we have an empty list.. */
  if(!last)
  {
    last = clients = malloc(sizeof(*last));
    last->prev = NULL;
  }

  /* Otherwise: */
  else
  {
    last->next = malloc(sizeof(*last->next));
    last->next->prev = last;
    last = last->next;
  }

  /* Populate new entry. */
  last->done       = 0;
  last->num        = 1;
  last->begin_time = time(NULL);
  last->next       = NULL;
  last->name       = malloc(strlen(hostname));
  strcpy(last->name, hostname);
}

/* Check to see if a hostname exists within the latest clients.
```

```
 * If so, return that client. */
struct client *client_exists(const char *hostname)
{
  struct client *tmp;

  /* Iterate over every client in the list. */
  for(tmp = clients; tmp; tmp = tmp->next)
    if(strcmp(tmp->name, hostname) == 0)
      break;

  return tmp;
}

/* Purge clients that haven't exceeded the limit. */
void client_purge()
{
  struct client *tmp;
  time_t current = time(NULL);

  for(tmp = clients; tmp; tmp = tmp->next)
  {
    /* Remove client if it has been in the list for more than 59 seconds.
*/
    if(current - tmp->begin_time > 59)
    {
      struct client *to_free;

      /* Free its hostname. */
      free(tmp->name);

      /* Remove reference to client. */
      to_free = tmp;
      tmp = tmp->prev;
      tmp->next = to_free->next;

      /* Free client. */
      free(to_free);
    }
  }
}

/* Handler for HTTP requests. */
static int mitigate_handler(request_rec *r)
{
  struct client *cl;
  const char *hostname;
  time_t      current;

  hostname = ap_get_remote_host(r->connection, r->per_dir_config,
REMOTE_NAME, NULL);

  /* Skip if client is already in allow list. */
  if(apr_table_get(allowed, hostname))
    return DECLINED;

  /* Check if we already have the hostname in the client list. */
  if(cl = client_exists(hostname))
  {
    /* Increment request amount and redirect if applicable. */
    if(!cl->done)
    {
      if(++cl->num == REQUESTS_PER_MINUTE)
      {
```

```
        r->content_type = "text/html";
        apr_table_set(r->headers_out, "Location", CAPTCHA_LOCATION);
        apr_table_set(allowed, hostname, "1");
        return HTTP_TEMPORARY_REDIRECT;
      }
    }
  }

  /* Otherwise, add the client in. */
  else
    client_add(hostname);

  /* Purge old requests every minute (or latest request). */
  current = time(NULL);
  if(current - last_req > 59)
    client_purge();
  last_req = current;

  return DECLINED;
}

/* Callback function to declare what other functions
 * should be called for request processing. */
static void mitigate_hooks(apr_pool_t *p)
{
  FILE *in;
  char line[256];

  /* Create table of allowed hosts. */
  allowed = apr_table_make(p, 0);

  /* Open file and scan in hosts. */
  in = fopen(ALLOWED_FILE, "r");
  if(in)
  {
    while(fgets(line, sizeof line, in))
      apr_table_set(allowed, line, "1");
    fclose(in);
  }

  ap_hook_post_read_request(mitigate_handler, NULL, NULL, APR_HOOK_FIRST);
}

/* Module information. */
module AP_MODULE_DECLARE_DATA mitigate_module =
{
  STANDARD20_MODULE_STUFF,
  NULL,           /* Per-directory configuration creator. */
  NULL,           /* Directory configuration merger.      */
  NULL,           /* Server configuration creator.        */
  NULL,           /* Server configuration merger.         */
  NULL,           /* Command table.                       */
  mitigate_hooks, /* Hook installer.                      */
};
```

Appendix 7: Source Code for Captcha.php Module

```
<?php

  session_start();

  // include the public library from reCAPTCHA
  require_once('reCAPTCHAlib.php');

  // Get a key from http://reCAPTCHA.net/api/getkey
  $publickey  = '6Lf3GAkAAAAAALlC4PzWSaOQzPhjgu-6EGyirjbx';

  // allowed attempts
  $attempts   = 3;
  $privatekey = '6Lf3GAkAAAAAAD6skRsUIqfOVHp9QT8Rjh13YCQB';

  if(!isset($_SESSION['reCAPTCHA_attempts']))
  {
    $_SESSION['reCAPTCHA_attempts'] = 0;
  }

  $resp  = null;
  $error = null;

  if(isset($_POST['reCAPTCHA_response_field']))
  {
    $resp = reCAPTCHA_check_answer($privatekey,
            $_SERVER['REMOTE_ADDR'],
            $_POST['reCAPTCHA_challenge_field'],
            $_POST['reCAPTCHA_response_field']);

    if($resp->is_valid)
    {
      echo 'success!';
    }
    else
    {
      $error = $resp->error;
      $_SESSION['reCAPTCHA_attempts']++;

      if($_SESSION['reCAPTCHA_attempts'] == 3)
      {
        $ip = $_SERVER['REMOTE_ADDR'];
        file_put_contents('.htaccess', "deny from $ip\n", FILE_APPEND);
        exit;
      }
  }
}
?>
<html>
  <body>
    <form action="" method="post">
<?php echo reCAPTCHA_get_html($publickey, $error) ?>
    <br/>
    <input type="submit" value="submit" />
    </form>
  </body>
</html>
```

Appendix 8: Protocols and Response Codes

HTTP Status Codes Obtained from

http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

On 22 May 2009 at 10:51

Informational 1xx

This class of status code indicates a provisional response, consisting only of the Status-Line and optional headers, and is terminated by an empty line. There are no required headers for this class of status code. Since HTTP/1.0 did not define any 1xx status codes, servers MUST NOT send a 1xx response to an HTTP/1.0 client except under experimental conditions.

100 Continue

The client should continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server.

101 Switching Protocols

The server understands and is willing to comply with the client's request, via the Upgrade message header field, for a change in the application protocol being used on this connection.

Successful 2xx

200 OK

The request has succeeded. The information returned with the response is dependent on the method used in the request.

201 Created

The request has been fulfilled and resulted in a new resource being created. The newly created resource can be referenced by the URI(s) returned in the entity of the response, with the most specific URI for the resource given by a Location header field.

202 Accepted

The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place. There is no facility for re-sending a status code from an asynchronous operation such as this.

Other 2xx codes exist but are not relevant…etc…

Redirection 3xx

300 Multiple Choices

The requested resource corresponds to any one of a set of representations, each with its own specific location, and agent- driven negotiation information (section 12) is being provided so that the user (or user agent) can select a preferred representation and redirect its request to that location.

301 Moved Permanently

The requested resource has been assigned a new permanent URI and any future references to this resource SHOULD use one of the returned URIs. Clients with link editing capabilities ought to automatically re-link references to the Request-URI to one or more of the new references returned by the server, where possible. This response is cacheable unless indicated otherwise.

[This is a common response due to the fact .htaccess file is used with Wordpress. In addition, the client who was under attack received assistance with their website from a Search Engine Optimisation company who used 301 redirects for the main pages.]

307 Temporary Redirect (since HTTP/1.1)

The requested resource resides temporarily under a different URI. Since the redirection MAY be altered on occasion, the client SHOULD continue to use the Request-URI for future requests. This response is only cacheable if indicated by a Cache-Control or Expires header field.

The temporary URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s) , since many pre-HTTP/1.1 user agents do not understand the 307 status. Therefore, the note SHOULD contain the information necessary for a user to repeat the original request on the new URI.

If the 307 status code is received in response to a request other than GET or HEAD, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

Other 3xx codes exist but are not relevant…etc…

Server Error 5xx

500 Internal Server Error

The server encountered an unexpected condition which prevented it from fulfilling the request.