DDoS DEFENSE AGAINST BOTNETS IN THE MOBILE CLOUD

David Jensen

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

May 2014

APPROVED:

Ram Dantu, Major Professor
Hassan Takabi, Committee Member
Cornelia Caragea, Committee Member
Barrett Bryant, Chair of the Department of Computer Science and Engineering
Costas Tsatsoulis, Dean of the College of Engineering
Mark Wardell, Dean of the Toulouse Graduate School

Jensen, David. DDoS Defense Against Botnets in the Mobile Cloud. Master of Science (Computer Science), May 2014, 73 pp., reference list, 71 titles.

Mobile phone advancements and ubiquitous internet connectivity are resulting in ever expanding possibilities in the application of smart phones. Users of mobile phones are now capable of hosting server applications from their personal devices. Whether providing services individually or in an ad hoc network setting the devices are currently not configured for defending against distributed denial of service (DDoS) attacks. These attacks, often launched from a botnet, have existed in the space of personal computing for decades but recently have begun showing up on mobile devices. Research is done first into the required steps to develop a potential botnet on the Android platform. This includes testing for the amount of malicious traffic an Android phone would be capable of generating for a DDoS attack.

On the other end of the spectrum is the need of mobile devices running networked applications to develop security against DDoS attacks. For this mobile, phones are setup, with web servers running Apache to simulate users running internet connected applications for either local ad hoc networks or serving to the internet. Testing is done for the viability of using commonly available modules developed for Apache and intended for servers as well as finding baseline capabilities of mobiles to handle higher traffic volumes. Given the unique challenge of the limited resources a mobile phone can dedicate to Apache when compared to a dedicated hosting server a new method was needed. A proposed defense algorithm is developed for mitigating DDoS attacks against the mobile server that takes into account the limited resources available on the mobile device. The algorithm is tested against TCP socket flooding for

effectiveness and shown to perform better than the common Apache module installations on a mobile device.

ACKNOWLEDGEMENTS

First I would like to thank Dr Dantu. Completing my thesis has been the most challenging academic accomplishment I have ever undertaken and I would not have had that opportunity without him. Also he ensured I always provided my best work into this and pushed for nothing but the best of success. I also want to thank everyone from the lab for serving as a source of inspiration and understanding while I completed my thesis. In particular from the lab I thank Logan for being there with technical assistance. As well as Fazeen and Michael for serving to read and help me craft my thesis.

Finally I thank my parents. I would not have accomplished anything without their never-ending support and encouragement of this endeavor.

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

Background

Denial of service (DoS) is a type of attack designed to prevent legitimate traffic from accessing a network resource. The term DoS refers to a single system launching the attack toward its intended target. Distributed denial of service (DDoS) refers to the case of multiple systems simultaneously carrying out the denial of service attack. In order to disrupt the operation of the targeted system the DDoS attacks focus on exhausting the resources in either the network or the actual server of the intended victim. DDoS attacks are traced back to 1999 with the tools Trin00 and Tribe Flood Network (TFN).[1] The attacks have become increasingly sophisticated and effective since those first tools.[2]

The two main vectors for DDoS attacks are either at the network level or the application level. Historically DDoS attacks would be focused on the network level attacking through methods such as malformed packets and spoofed IP addresses to exhaust bandwidth and routing resources. Recently the trend has been to see the DDoS attacks incorporate or rely exclusively in attacking on the application layer.[3][4] The popularity of application level attacks can be traced to their respective effectiveness in overwhelming one particular aspect of the victim's resources.

Mobile cloud computing is defined by one or more mobile phones forming an ad-hoc network that can provide web service to outside connections or intra network communications.[5] The mobile usage focused on the purpose of DDoS defense in a single mobile running a web server. This web server can be representative of a user wishing to have

content with them at all times that can be displayed to all nearby local connection or to serve out to the internet.

Another possibility from a mobile ad hoc network is the sharing of resources.[6] A mobile when needing to complete computationally intensive operations would be able to seek and enlist the aid of other nearby mobiles over WiFi connections to offload some of the required processing. The nearby mobiles that are idle would be able to share resources to complete small amounts of work for the requesting mobile. This example requires the assisting mobiles do not offer too much power for assistance as to not drain their own batteries or occupy resources to the deficit of the device operators' intended usage.

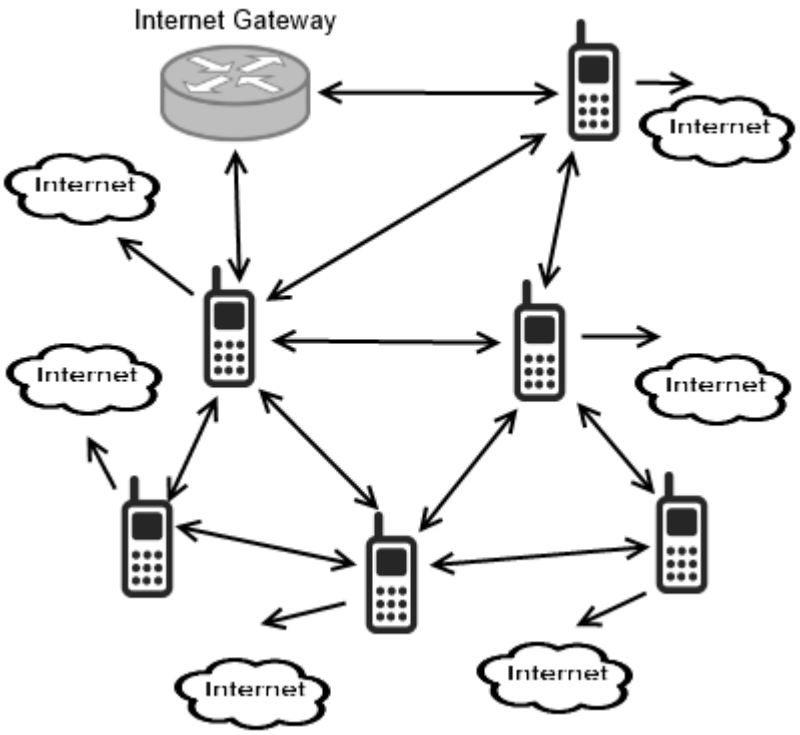

Figure 1. Sample ad hoc mobile cloud.

Example map of an ad hoc mobile cloud network. The mobiles are interconnected into an ad hoc mesh network through wireless or sensor communications. The ad hoc network has

access to an internet gateway to communicate out from the network. Additionally each mobile maintains its connectivity to the internet. This is the structure unique to ad hoc network by using mobile to have constant internet connectivity as well as intra network communications.

Finally mobile clouds can be leveraged by governments and military for various uses. Military and organizations alike that desire to host private networks can make use of a mesh network comprised of a mobile cloud. Here a force could setup the mobile cloud and handle communications and sensor reporting for all the mobiles in a given area with a mobile cloud passing the information.[7] This would be beneficial in comparison to mobile devices transferring information through traditional internet backbones as it would eliminate the traditional internet links that could be intercepted. A government may also wish to use a mobile cloud setting in cases of public safety needs. A mobile cloud can be setup around a disaster area to alert any mobiles entering into the cloud area of the danger.[8] Also the mobile cloud could track the movements of first responders and report location data to each other in the ad hoc network.

Using a mobile phone as a web server also comes with advantages for potential users. Primary the mobile web server is an excellent starting point to develop the future in mobile ad hoc networks. Users can host web pages on their phones at all times that are being served to the users around them at any given point in the day. One potential expansion of that is users mobile web servers can give information about the user that is completely user controlled. Then over the WiFi ad hoc network business could gain access to user supplied information on shopping preferences. A framework is established in the coffee shop to use mobile ad hoc networking to automatically place orders for users that enter the business and have properly

configured their own mobile servers to broadcast order information out. The other main advantage to hosting web servers on mobile phones is the ability to run those pages independent of the internet. Pages can be served by a user in an area that has no connectivity such as after disasters cause power loss. Users would be able to host information sharing on their mobiles to others in the area through an ad hoc network despite no access to internet gateways.

DDoS Attack Types

Network Layer

The main focus in network layer DDoS attacks is to consume all of the available bandwidth of the victim's network. One such method to accomplish this is to a data flooding attack where the attacker sends massive amounts of data that must be processed by the victim allowing for the occupying of all the available bandwidth on the victims network. [9] Another way to accomplish expending all the available bandwidth is a UDP Flood attack. This is sending a high number of UDP packets to the victim across random ports. The packets have spoofed IP addresses so the victim returning unreachable destination packets will not be returned to the attackers, thus occupying no additional resources for the attack. The sheer number of UDP packets that can be sent to the victim will accomplish the task of occupying all bandwidth.[10]

TCP SYN floods are a commonly successful network layer attack.[11] This attack utilizes an exploit in the TCP handshake process where a server is required to allocate a data structure on the stack to handle the incoming connection. [12] The attacker executes the SYN flood by sending the victim server a TCP SYN request with a spoofed nonexistent source IP address. The victim server will receive the SYN request and allocate its data structure on the stack to

represent the connection request. It then sends a SYN-ACK packet to confirm the connection to be started. Because the attacker is using spoofed source IP addresses the SYN-ACK packet will not be responded to with the expected ACK as the attacker never intends to establish the connection.[13] The received connection request will then sit on the victim memory stack until it reaches its timeout limit and drops the connection data. During the delay from generating the connection request data the timeout and attacker is able to generate such requests to the point that there is no longer room on the stack to receive any additional requests. Without the ability to receive any additional TCP connections all legitimate traffic is dropped from the victim until the TCP flood has ended.

While becoming less common the smurf attack is an attack accomplished through an ICMP flood capable of disabling the target. The attack uses an ICMP echo request to abuse a broadcast address. The attack will spoof an ICMP echo request with the source IP address as that of the intended victim. The request is send to an intermediary broadcast network address. Upon receiving this request this intermediary broadcast will send all of its network members the echo request. Each of these network members then send their echo replies back to the source IP which is given as the intended victim. The high number of echo replies then will overwhelm the bandwidth available to the victim.

Application Layer

As opposed to network layer attacks focused on occupying all the bandwidth application layer attacks target the exhausting of a victims resources (some examples being sockets, CPU, memory, or I/O). A main advantage of application layer attacks is they are harder to detect as often attack traffic is indistinguishable from normal HTTP traffic. Though different in focus the

reflection attacks on the application layer serve much of the same function as the network layer attacks. A common method is to use DNS servers as the reflection point.[14] The attacker will create DNS requests with source IP address as the intended victim. The DNS server responds to queries with its response messages which are much larger in size to the received queries. These large messages are directed toward the target where it's unable to handle the flood of responses.

More specific to the application layer are attacks that use the HTTP for specific functions to achieve the DDoS.[15] First is the session flood. This can also be known as the excessive VERB attack. The attack is designed to generate connection requests from the attacker in such numbers as to overwhelm and prevent access from legitimate users. The attacker has no need to spoof IP address and will simply send as many valid GET or POST requests as the systems can usually carried out with a botnet. A variation of that attack is the excessive VERB single session in which the attacker makes many valid requests within its single HTTP session. This can succeed when the victim has a security system capable of constraining the number of sessions.

An attack that relies less on the number of attacking systems needed is the slowloris attack.[16] This is where the attacker will send HTTP requests without the complete set of header or body content. The victim server will open these connections while the attacker slowly sends the rest of the information. Exploiting the server keeping these connections open will allow the attacker to continue opening more and more connections under this method eventually occupying all of the server's available connection sockets. With no available sockets legitimate connection requests will fail to establish with the server.

A variation on the slowloris attack is the slow POST attack. Here the attacker will send a POST request with the correctly specified body field length. The attacker then sends the content at a managed and slow rate. The victim server will keep these connections alive while receiving the content so slowly the attacker is able to overwhelm the amount of available connection sockets much like the slowloris attack. This attack has been shown successful against IIS and Apache servers.[17]

Depending on the victim's application a successful route to DDoS can be achieved in exploiting specific application vulnerabilities. These attacks will succeed by overloading the victim CPU. An example would be attackers learning of SQL injection vulnerability and flooding the victim with injections to lock up the CPU and bring the whole server down.

Known Defense Methods

Defending against DDoS attacks is a task that requires the wide scope to encompass as many known attack types as possible while remaining capable of mitigating attacks not yet developed with known signatures.[18] The ideal defense strategy would be to stop a DDoS attack closest to the attacking source to prevent the wasting of resources not only at the victim but on the routing path to the victims system.[19] The challenge faced is that detection of a DDoS attack is accomplished much more successfully at the destination point of the attack. The two main classifications for describing current defense strategies will be divided on where they take place in terms of the defense mechanism location.

Source and Hybrid

Source and hybrid based defenses will describe methods of protecting and detecting DDoS attacks that occur closer to that attackers system or will encompass components from multiple locations in the route for the attack.[20][21]

One method designed to combat network layer attacks is ingress filtering for source edge routers. [22][23] The proposed method is to have the edge routers determine if packets that originate in their own system have their source IP address set to a valid range within the systems IP address limits. If the packets do not then the edge router will deny the packet and not forward it along its intended path. The main limitations of this filtering will be that attacks originating from attackers that do not spoof their source IP address will not be stopped.[24] And this system could also harm mobile phone users as the current structure of IPv4 has mobiles receiving internal private IP addresses based on the network providers routing. Therefore as a mobile phone moves across the network its IP address will stay associated to its home agent while changing the edge router it would send traffic through.

Another proposal for stopping DDoS attacks close to the source of the attackers is D-WARD.[25] This method monitors traffic flow both in and out of the edge router. The totality of the traffic is considered to be a flow. This flow is analyzed for patterns that would represent typical or atypical behavior in a network. Flows are stored based on the originating IP within the edge routers network, holding statistics on TCP, UDP, and ICMP traffic. An example being the monitoring of TCP to detect number of packets in versus the number of packets out. After a set interval the flow is compared to the data of a typical TCP flow to detect for abnormalities. A limitation of D-Ward as a solution is first in storing of the flows statistics. Due to memory

constraints not all source IP address can have flow statistics saved indefinitely. A hash table of recent connections and higher use connections is thus implemented to track as many IP address as the system can handle. The other main limitation is a matter of incentive for networks to implement a D-WARD system. The system may be effective in filtering out DDoS attacks originating in the network but that tangible benefit is not as easily seen as the intended victims of DDoS attacks that are being stopped by D-WARD are not required to be running D-Ward themselves.

A similar method that instead will focus on the bandwidth of traffic flowing both in and out of edge routers is found in MULTOPS and TOPS. [26][27] MULTOPS is designed to monitor the bandwidth looking for an imbalance of activity flowing out of the network compared to return traffic over its specified IP addresses. The MULTOPS router will store packet bandwidth rates for each of its IP addresses and drop packets when an imbalance that may represent a DDoS attack is detected. A limitation of this is that the memory used to store bandwidth rates for each IP address can be attacked itself and quickly exhausted of available space. Knowing this the TOPS solution was proposed that instead of a dynamically build structure for storing IP addresses traffic a fixed table will be generated to represent all internal IP addresses. With the benefit of a fixed table the TOPS method will not be vulnerable to DDoS attacks. The drawback though still shared with the MULTOPS system though is that legitimate network traffic is capable of existing at an imbalance of bandwidth to and from an IP. An example would be a stream of multimedia content will largely be one directional and has the potential to set off the abnormality filter while having no place in a DDoS attack.

No single source of defense is capable of stopping DDoS attacks, so current methods being proposed are focused around the hybrid model with different components of the defense located in separate locations along the internet and networks.[28] Basically the setup could be that the detection of a DDoS attack would occur at the destination location which would then implement filtering closer to the attackers systems.[29]

At the application level a hybrid approach is the Speak-up.[30] The novel approach here is not to filter or limit the bandwidth on attackers. But instead have the destination detect attacks as they are in progress then send requests out to all incoming connections to increase their respective upload rates. DDoS attacking systems will be operating at maximum upload rates already so the only connection that will attempt to increase are the legitimate users.[31] These legitimate users are then able to occupy a higher percentage of the resources and complete their tasks largely unaffected by the DDoS. The drawback to this method is it is only fit to be deployed on systems large enough to handle the higher traffic loads. A limited resource server would only serve to harm itself faster.

An application specific defense commonly undertaken is to determine if a user is a human or if it is a bot. One proposal is to have browsing statistics designed to store the typical human user interaction with a web server.[32] One key component is detecting bots based on the speed at which they can request additional pages. Typical browsing data will show humans to redirect at expected time periods. While an aggressive bot will instead generate traffic at its maximum rate by minimizing time between requests those bots can then be filtered against. Another proposed feature is adding links and text fields that are small enough to be invisible to a human browser. Then any interaction with said links will signify that user as a bot.

Another hybrid based example is a trust based model. The proposal is to monitor a user's connection to the server and use browsing statistics to generate trust levels.[33] An Entropy is used to serve as a measure of randomness to help represent the interaction of human user. Then when a DDoS attack is detected the method will continue forwarding packets associated to users with high trust values and low trust values will immediately begin being dropped. [34] Unknown trust values will be evaluated and if upon receiving an entropy value that deviates too far from predefined expectation that session is dropped and source is identified with low trust.

Destination

These methods will refer to the defense against DDoS attacks taking place at the destination of the attacks, being either the victim server or its edge router. These methods in comparison to source based defenses will much more accurately be capable of detecting attacks as they occur.[35]

The first solution to defense occurring at the destination is that of enlisting IP traceback methods.[36] This is specifically to trace the true sources of attacker IP addresses and help overcome the attacker's usage of spoofed IP address.[37] Solving this problem from the destination is accomplished by determining which connections are attackers then recursively test upstream links to locate where the attacker is originating from.[38] The limitation with finding sources through this method is the extensive overheads required as the ISP would also need to cooperate in moving up the upstream connections.

Another attempt is to keep logs of connection usage when the system is not under a DDoS attack. The method will then when under attack check previous logs in order to filter

traffic according to the past behavior.[39] Only allowing previously established connection in the log will effectively filter out an attack and not compromise use of the correct users with a previous visit though it will not assist legitimate users on their first connection attempts.

Application layer attack defenses at the destination include a proposed defense against amplification attacks.[40] The first step is to disable open recursion on name servers from external sources. The proposal is then to keep a database that will track DNS messages.[41] The table will track all the DNS requests going out and then upon receiving a DNS reply can cross check that there exists and request before forwarding along the reply to its destination. The limitation is that the router must maintain the database and the overhead associated to the additional step of logging all DNS messages.

DDoS-Shield is a proposed tool to protect at the destination through monitoring HTTP session activity profiles.[42] A value is assigned to each session that in contrast to most methods is not a binary value but a continuous value. Then the shield will serve as a rate limiter giving bandwidth and connectivity assignments based on the connections value.

Summary

The further purpose of the paper is to focus on the possibilities of DDoS attacks in the environment of mobile devices. Chapter two provides background on how a botnet capable of DDoS attacks could be constructed on the android phone platform and what its attack capabilities would be. This will show the design and structure of a potential botnet and its feasibility in attack output per bot.

The third chapter will then focus on how resilient a mobile is against DDoS given the current common tools available to Apache servers. Readily available modules will be examined

for their potential success operating on a mobile device. Then a proposal of a mobile specific method is created to serve as a better method of defense against DDoS attacks that is specifically created for use on a mobile device.

CHAPTER 2

MOBILE BOTNETS

With the growth in amount of active mobile phones running the Android platform there are security risks that while known and defended against on PC are viable for implementation on Android devices. [43] Botnet is described as a collection of infected Android devices that can receive and execute commands from the botmaster, who will have sole control over the botnet. The main focus is in designing for the ability of the botnet to show capability in launching a denial of service attack. The botnet was tested for the amount of bandwidth capable of occupying as well as the volume of connections that can be occupied.

The purpose of building a mobile botnet is in having the full capabilities in a traditional botnet at the hands of the botnet spread across many mobile in a platform that is not ideally ready to have defenses against being infected.[44][45] Worms, Trojans, and other popular propagation methods are at this point well known enough on the pc that antivirus software or operating systems themselves are able to detect and disable most.[46] Currently the mobile platforms are not as strongly secured for the average user. This means that designing a botnet for mobile phones can easily be spread and left to run on the mobile with a lowered chance of detection from the infected users operating system.[47] Android uses permissions to alert users to what services of the mobile an app is requesting access too. The downfall is that these permission categories are often much too vague for a user to make a proper informed decision. In granting an app permission to use internet connections a user has unwittingly enabled an app to run a bot in both command and attacks over that internet connection.

The mobile botnets end result is to generate traffic flows sufficient for successful DDoS attacks against a chosen victim. A mobile cloud can further be used to advantage in the creation of a botnet as connected devices would be able to propagate the infecting code as well as pass command messages within the ad hoc network saving further chances of detection from making calls out to the internet for instructions.

Background

Botnets have existed on the PC arena for over a decade and are evolving constantly as the defense mechanisms respond to each newly found botnet design.[48] The project focused less on the distribution of a botnet and instead work on the command and control and execution aspects. There are two main methods for command and control of a botnet, push and pull.[49] Push means the botmaster will issue commands and have these commands sent to bots and spread throughout the botnet by forwarding the commands to each bot.[50] Pull uses a method where the botmaster will setup the command and have all the bots set to look for the command on schedules and execute from there. Most early botnets relied on pull structure often using IRC as the method of posting commands. Security professionals have been able to severely limit the viability of this method and have ushered in many more peer to peer style push botnets. The main limitations to pull command and control botnets on PC was the ability of defenders to quickly detect the botnet and begin receiving the commands or potentially injecting their own commands into the system.[51]

The presence of botnets on mobile devices is a recent advance with some examples of potential consequences of the spread of an effective botnet. The main method of mobile botnet so far created has been that of using SMS to develop a push style command and control.

The advantages of using SMS are laid out. First the passing of commands by SMS greatly reduces likelihood of detection as SMS traffic[52] is not frequently monitored. Additionally the usage of push control through SMS allows devices not currently on network to still receive commands when they return to coverage areas. The Android platform allows for the reading of SMS messages and deleting the contents before users are even alerted to the arrival of messages reducing chances users will discover the bot on their device.

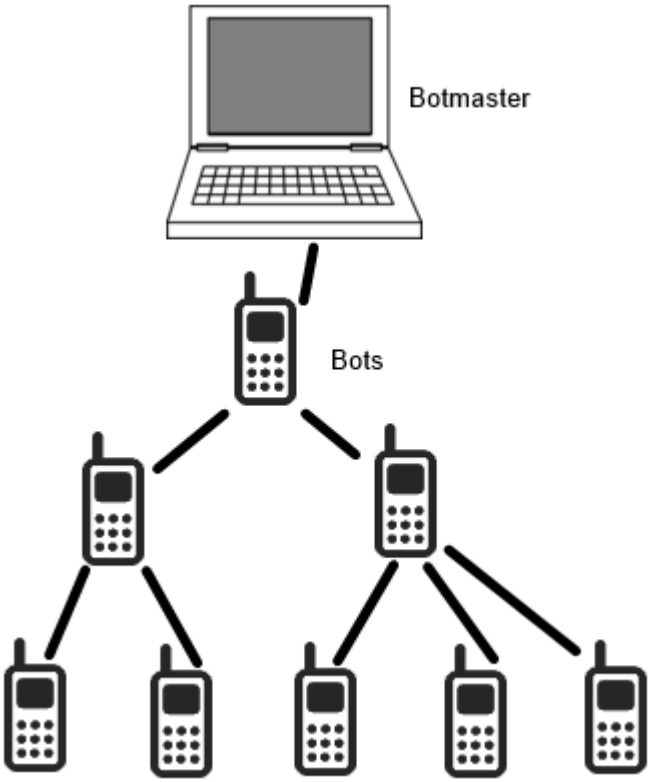

Figure 2. SMS push botnet structure.

One approach has been to implement command and control elements into SMS while also building other command and control elements into the botnet to form a hybrid.[53] This brings in the use of Bluetooth picked for its availability on most smartphones. The Bluetooth can assist in short range command forwarding. Though the design must be careful to enable and disable the Bluetooth to only run when the botnet needs it, as extended running time of

Bluetooth will create noticeable battery loss for the smartphone.[54] The final component is to use HTTP channels to deliver information from the bots back to a server under the botmasters control. HTTP in this botnet is not used to deliver commands or spread the botnet in any way. The topology of the botnet is structured as a tree with the use of selected bots to serve as cluster heads responsible for forwarding commands on down the tree.

Another approach to building a mobile botnet was to exploit the availability of unsecured WiFi access points encountered in most cities. [55] WiFi is chosen exclusively over phones data connections as it provides a much stealthier channel. In order to use open WiFi connections the botnet is designed to handle all its traffic over port 80 as HTTP traffic. This will prevent the occasions where an access point has restricted opening traffic through any other ports. In order to function then the bots will join open access points when possible and immediately attempt to retrieve commands from the botmaster by HTTP to the botmasters controlled server. The topology is centralized control where no bot relies on another to relay commands. This build showed not only the capability of using open WiFi but demonstrates the potential DDOS results in the botnet issuing SYN requests over WiFi connections as potentially capable of flooding sites if scaled up across multiple geographic locations.

One of the most important design features in developing a botnet is to keep the command and control as stealthy as possible. A potential method for this was developed to use Google's cloud to device messaging (C2DM) as the basis for command and control.[56] C2DM is a service provided to have Google servers forward commands to installed apps on mobile devices. This push command will be sent to available devices and delayed and resent to devices that are not immediately available when the command is first issued. The botnet is able to take

advantage of the high uptime availability of Google's servers to help overcome the single point of failure introduced by having all commands in C2DM. C2DM was deprecated by Google on 6/26/2012 and replaced by Google Cloud Messaging (GCM). GCM alleviates some of the difficulties the C2DM botnet faced in terms of messaging limits allowing for the botnet to not have to be broken down into smaller botnets. A major advantage in this design is the ability to deliver a payload with the commands compared to SMS botnets where commands were limited to just delivering a message.

One of the main difficulties with centralized command and control is the danger in having a single point of failure to issue commands from. To combat this SURL Flux is designed to seek commands from a set of sources.[57] This is accomplished by each bot generating URLs that are potentially under the control of the botmaster that will be holding a special jpeg file that has the commands encoded and appended to it. This generating of URL at the bot level is an advancement made forward from Andbot that requires getting the URL by accessing a blog server.

The recent developments of mobile botnets are showing a growth of sophistication in the methods used for command and control. This focus of discovering the potential in varied structures is allowing the developers of botnets to select methods best suited for the actual construction of their botnets. One aspect of the botnet not focused on is the propagation. The main method currently for propagation is to use repackaging or Trojan applications. [58] This is chosen especially on the Android for its ease of delivery and exploit the trust of users in their application downloading decisions. Furthering the advantage of the attacker is that there exists

no universal or centralized method of security for mobile devices. Given the CPU and battery needs a robust antivirus solution does not exist that can trap and detect mobile botnets.

Proposed Botnet

The design for the botnet exists in two main pieces. First is the code that carries out attacks from the compromised mobile devices. This was designed to be as modular as possible in order to function as the execution aspect regardless of how the botnet is structured or spread across devices. The main focus here was for generating site traffic for the purpose of denial of service.[59] This was accomplished through one of the known methods of denial of service traffic.

The second piece of the botnet design was the creation of a sample application that demonstrates the setup in infecting a device and receiving and handling of instructions from the botmaster that are then handed off to the attack piece.

For the purpose of the project this application piece focused on the exploitation of Twitter. To begin an app was created that can offer users Twitter services and designed as a simpler user experience compared to the official app. The choice is to set the app apart and generate downloads and installs from users that trust and utilize the application for its stated purpose. Application is structured around a main view and activity calling the model for functions and a separate service that will be setup to run from the main activity and then start on boot from phone.[60] With the integration of Twitter for hiding commands the botnet will now be referred to as ATbot for android twitter botnet.

By using Twitter for legitimate reasons and the front of the application users will be granting the app authorization to Twitter. For the purposes of ATbot the harvesting and malicious use of the authorization tokens being passed to infect devices was not explored. The benefit of authorized devices is for the command and control. The botmaster will be able to simply setup and public Twitter account and post command instructions to that account. Infected devices on schedule will then pull down such tweets and follow the command from there. Having infected devices authorized on Twitter will hide the command pulls inside of otherwise legitimate network traffic. Detection of ATbot from receiving instructions is greatly reduced as Twitter traffic will frequently be white listed from security applications and not monitored. [61][62]

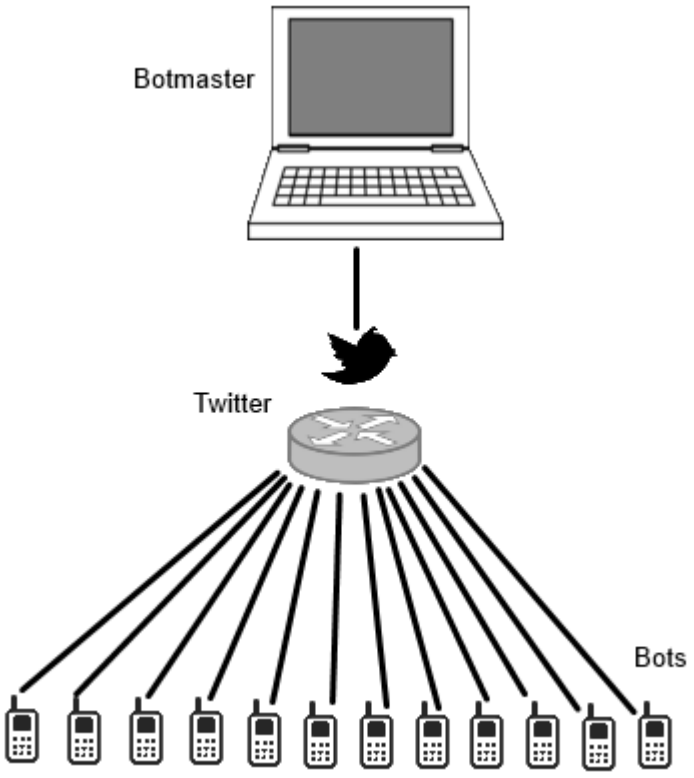

Figure 3. ATbot structure.

ATbot was designed to interpret and execute a set number of commands. There was also a requirement of basic functionality that serves as the false front of the app that users are interacting with that ultimately leads to Twitter authorization. The building of the front app will be ignored as anything that will grant Twitter authorization codes will be considered sufficient to help propagate the botnet.

Function structure of ATbot

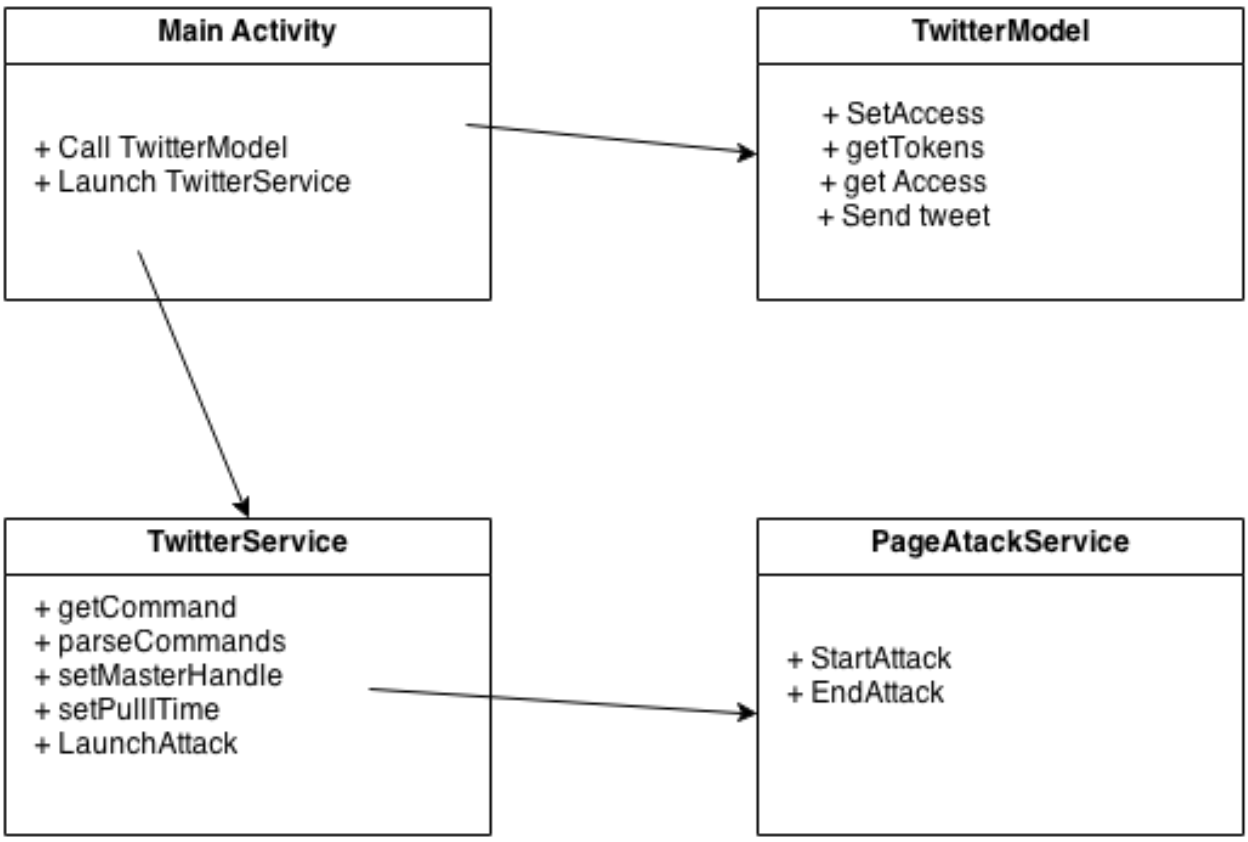

Figure 4. ATbot design.

ATbot was setup to work as modular as possible to fit into any false front application that is designed. Therefore The MainActivity calling the TwitterModel will only need to differentiate the app specific tweet needs such as posting a high score to the users timeline. TwitterService works as a background service and will be called on device boot or opening of

the app. The MainActivity checks for proper access tokens before launching though to ensure the service will be capable of retrieving the commands.

TwitterService is setup so that it is passed a main botmaster account and time in which to check on its startup. The functions exist such that the service will fetch and reset its botmaster Twitter handle. The design is such that a with multiple applications capable of building in ATbot there needs to be a single Twitter account on first access for non updated release packages that can point to the current botmaster account for current commands. A handle may be capable of being detected on the side of Twitter if the botnet has grown to a size such that thousands of accounts checking a single user would alarm Twitter to an account acting outside of expected bounds. Having set the first master account the app checks as merely a handle forwarder should prevent a simultaneous surge of timeline checks as the apps only will view the once on their granting of access tokens. The other functions that must be capable of are setting the time in which to check for commands and resetting to future botmaster accounts. Likely operation of the account would be to set the pull time and issue a new botmaster account in every tweet the bot pulls down. Finally TwitterService will need to be capable of calling another background service to launch an actual DDoS attack.

The PageAttackService is as simple as a background service that is also designed to function independent of the false front application an infected user downloaded. Launching the attack service will require only a where, when, and how long. It is capable of receiving the site to attack and given a time and for how long the attack is planned to last.

One key aspect of the pulling of tweets from the botmaster is that the instructions must be encoded and make use of short instruction sets. Encoding will assist in hiding the exact

nature of the botmasters tweets lowering the change of discovery of the function of the botnet.[63] Instruction sets must be kept as small as possible as a main limitation of Twitter is the 140 character maximum.

Capability of Android Based Bots

The proposal of ATbot relies on the infected android phones being capable of generating traffic sufficient to harm the function of a site and thus function as a DDoS attack. Designing of code that can properly execute an attack from android requires working around the built in limitations and security preventions. The main limitation is that java for android will not allow the spoofing of source on IP headers. Without this capability the use of a network layer attack is not examined any further. The focus of the attack code was to work on the application layer by generating page requests in sufficient number to overload a servers capacity on either bandwidth or connection limitations.

The attack always occurs with an intent service to allow background operation as a user would not allow the operation of a botnet on their phone if they are capable of discovering its operation through something as trivial as a foreground icon or alerts. Also of note is that attacks should not be used at too frequent of a rate such that a user could be capable of noticing a large uptick in their data usage and discover the botnets presence in solving for potentially higher bills.

To determine the capabilities of individual phones as bots in a DDoS the test is setup to pull down a target webpage for a set number of iterations. The time was recorded for total time to complete the test. This allowed to generate and expected number of page requests the

phone is capable of generating per second off of a typical web server. The first test was to check if the phone can operate successfully from both a 3g and WiFi connection. The attack was directed to run in a short burst at www.cse.unt.edu. These tests were not designed to actually impair the performance of the targeted site so results did not produce timeout or forbidden http responses.

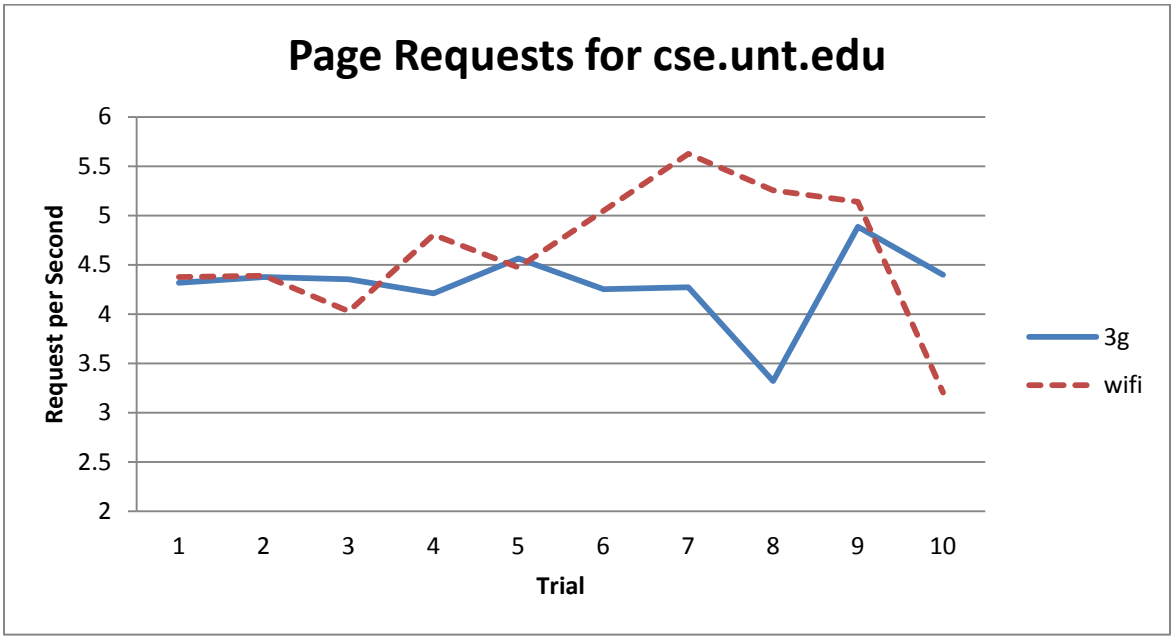

Figure 5. Bot page requests shown in both 3q and wifi connections for number of requests each bot is capable of generating per second.

The results of 10 trial runs for pulling page requests from the site show more a slightly greater amount of pages per second from WiFi. The averages being 4.3 pages per second on 3g and 4.6 pages per second on WiFi. While not insignificant, may show improvement for operation on the potential large scale of a botnet >1000 bots. The data supports that the android phones will be capable of producing traffic for a DDoS on either WiFi or mobile data connection. This will allow the attack code to run when designed without concern for the user

being on WiFi or not. The only limitation to the attack would be the phone is off or in an area without access to WiFi or data.

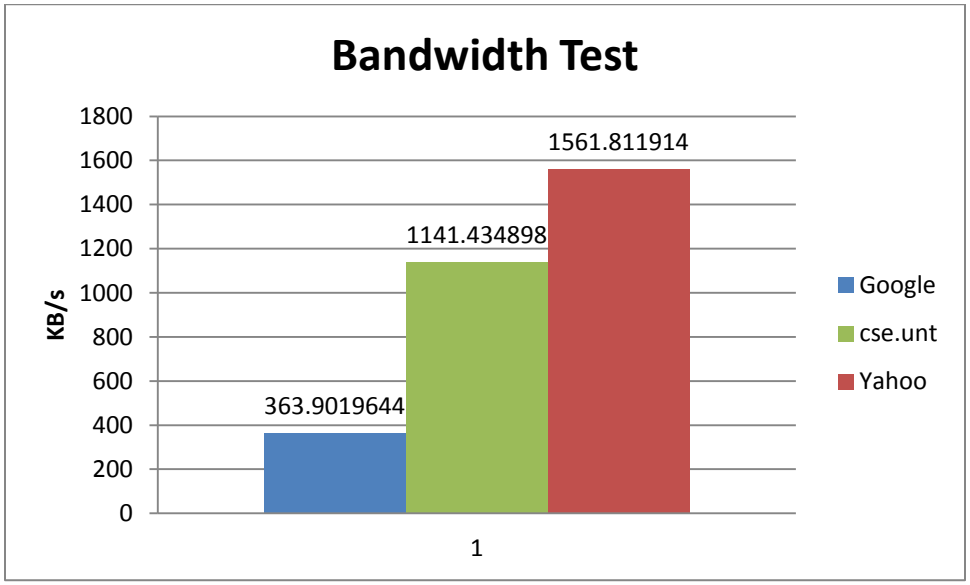

Figure 6. Bot testing for the total amount of bandwidth used in KB/s.

The bandwidth test conducted the page requests at the maximum rate capable over WiFi to determine the potential maximum traffic a single mobile can produce. The results are reflective of the amount of content a page serves and is inversely related to the page request yield in connection numbers. With unchanged attack of page requests the specific site to be targeted may thus show resource exhaustion in a different manner depending on its content volume. The other feature for a botmaster to be aware of is that in attacking a site with more content on the page being requested there will exist greater potential data usage for the phone user to take notice of the botnets presence.

Summary

The conclusion is that creating a botnet on the Android platform is not only possible but can result in a botnet that is capable of generating the traffic to launch DDoS attacks and

succeed in affecting a web server. With a strong propagation method the mobile botnet can spread to enough devices to be viewed as a true threat to server operators and must be accounted for in their defenses.

CHAPTER 3

MOBILE SERVER DEFENSE AGAINST DDOS

With smartphones constantly becoming more ubiquitous one new potential area to use them is in the hosting of personal web servers. Phone connectivity has shown that in 2013 the average smartphone generated 529MB per month.[64] As the common user is experiencing a more stable network and reliable data connections the ability to host their own servers is rapidly becoming a possibility. A use for a personal web server can be in always having a personal web page or blog that the user can update on the go. Or users can hold their presentation information on the phone and in instances without a full audience on the internet a local network can still allow connections to view all of their information.

As for why a mobile cloud and servers need to be examined for defense against a DDoS attack, the reasons depend on the intended operation for the mobiles. In the most general example malicious agents could be attacking mobile servers for no direct benefits but find gain by disrupting a users network from standard operation. If the operation for example was in ecommerce this would disrupt a business even though it provides no direct benefit to the attacking agent.

In the example of military based ad hoc network a DDoS attack could be directed at nodes in the network from an outsider interested in breaking the communications. For instance a military ad hoc network may have resources with mobiles reporting status and location. A DDoS attack could then overwhelm a selected resource and prevent other mobiles in the ad hoc network from being able to communicate. This may lead to a decision of operations being sent

out to investigate the now non reporting node that would lead to wasting of resources or danger to the military.

Problem Definition

The purpose of testing an apache server for success and failure of performance against DDoS attacks is to discover and overcome the unique challenge of a mobile phone compared to a traditional server environment. Experimenting is done to discover the best defense against DDoS while maintaining any changes to default Apache to have as small a footprint as possible as the capability of the mobile to serve a page is limited to begin with and adding cumbersome tools will degrade the performance too greatly for legitimate users. The goal is for the mobile web server to detect and reduce the effectiveness on any incoming DDoS attacks that are launched against it. Ultimately the aim is to design a DDoS defense mechanism that requires little additional resources beyond the basic installation of apache while providing substantial improvements in defending against DDoS attacks in comparison to the standard installation of apache.

More powerful web services could be provided by linking multiple mobile phones together to form an ad hoc network. For the purpose of examining the effects of DDoS attacks against a mobile web server ad hoc networks are not analyzed. They are not looked at as the complexity of maintaining the network and resource and connection sharing will change the approach to defending against DDoS attacks.

Testing Environment

With the mobile phone web hosting still being in its earliest stages the web servers are not optimized and secured such as one would find in looking to setup a traditional server. For

the purpose of the testing the web server is setup to have basic security measures in place by virtue of demonstrating a simple web server run from updated publicly released resources. Namely the server is to run on apache and the webpage is off of wordpress code. The testing will thus be done entirely on mobile web server performance under small scale DDoS attacks.

In order to demonstrate easily available resources Apache and wordpress for the experiment a LAMP stack needed to be setup on mobile devices. With a Linux distribution running on top of android for the Motorola android device and the developer release of Ubuntu touch on the Samsung the installation of the AMP stack is the same straightforward process and server could undertake through shell access. The phones are run with root access being granted to the Linux shell in order to make all necessary file system modifications. While not ideal to require root access for a smartphone owner to setup the web server the test scenario was not focused on the ease of end user web server installation but on the configuring of said server.

After the basic installation of the LAMP stack and verification of a connection the server is then given the default installation of a wordpress site. No theme or content changes are given, the site is installed just to provide a small amount of content to transfer above the apache default index.html page. The wordpress installation also ensures the proper LAMP setup as it will fail to function without the required sql and php downloads.

Testing of the DDoS defenses will be accomplished by creating an attack against the mobile web server and monitoring the performance during this attack. The type of attack used can be characterized as a sustained TCP flooding attack. This attack is designed to exhaust server memory resources to generate failed connections and broken data transfers. This attack

is accomplished by having machines open sockets on the mobile server as fast as they can back to back. Those sockets once opened will not be used further from the machines but no close connection messages will ever be sent. Forcing the mobile server to keep opening new connections and maintaining memory devoted to the sockets that are no longer going to have any messages come in through them.

Examining Modules

The most advanced module that can be attached to Apache2 for defense against DDoS attacks is mod_evasive. This mod is accepted at being successful in helping Apache servers mitigate DDoS attacks, but for the mobile setting it is not deemed an acceptable tool. The drawbacks that limit its effectiveness on a mobile web server start with the footprint left from its operation. Mod_evasive works by creating a running hash table on the IP addressing making requests to the server per child thread that is spawned from Apache. Essentially the trouble here is that in creating and watching for repeated page requests on the same child mod_evaisve is efficient in denying access to the IP address making those requests at the expense of processing the connections for each individual child. The mobile does not have the spare resources to constantly maintain the checks at each child level. While configurable another limitation is that the checks are designed to run as a set limit on the number of requests per defined time period. With the already limited processing power and bandwidth available to the mobiles the configuration for mod_evasive would have to be extremely narrow windows to catch the attacks. The setting would need to be in the nature of a few requests per second to represent a DDoS attacker. That figure runs too dangerously close to a legitimate user making a burst of refresh requests and accidentally blacklisting themselves. This kind of

chance at false positives is unacceptable to the needs of a web server. Mod_evasive is really left to its best usage on a server that can support the bandwidth needed to configure the settings such that the chance of confusing a legitimate user from malicious bot activity approaches zero. The other huge drawback in mod_evasive on the mobile is that even when triggering for a malicious user the connection requests are still able to queue before being replied to with 403 forbidden http messages. For these reasons no testing scenarios are executed to work on mod_evasive on the mobiles as the installation of it would be impractical to the goals of reasonable DDoS defense.

The next examined module is ModSecurity. This module at its core is designed for to see HTTP traffic and then allow for analyzing and logging for events the user is watching.[65] The benefits of using ModSecurity are found in its foundation of hooking into Apache at specific points and reading the information going through the server. ModSecurity hooks into Apache at the five locations: request headers, request body, response headers, response body, and logging. The module is also built around the concept of using rules to detect conditions and take specific action when those conditions are found at any of the hook in points. The flexibility offered in writing custom rules at a predefined point in the connection process can allow for powerful usage of the module to achieve specific goals. One consistent area of success with ModSecuirty is defending against XSS or SQL injections as the rules can read the requests and properly strip the malicious attempts before passing them along. The final main advantage is the ability to utilize persistent storage from the rules. This can be a key feature in monitoring the IP addresses of malicious requests to filter and deny.

Despite the flexibility and potential successful usage of ModSecurity, drawbacks were found that prevent it from being a good tool for defending against DDoS attacks on a mobile web server. The first limitation is that the module is not designed to function as a successful defense application out of the box. Users must either create or download preexisting rules and configure them. The danger here is that the process of creating rules can be quite complex to get correct and leaves a high change that the user will create rules that either do not fully accomplish the stated goals or worse may leave the server vulnerable in some new unplanned way. Even in downloading preexisting rules from trusted sources the configuration must be looked at carefully as no two servers will have the exact same needs and improperly configuring downloaded rules can also create unintended consequences. In both cases incorrectly setup rules can result in blocking service to legitimate users by generating too many false positives.

The main limitation on implementing ModSecurity for the mobile web server though is in the amount of resources required to run it. One of the first features is that on the request body and headers ModSecurity will buffer the data until the whole request is finished. This is helpful in analyzing the incoming traffic but will place a RAM burden on the server to keep all its requests buffered until complete. In the case of slow incoming connections on the mobile these buffered requests can quickly add up to occupy too much memory for proper server function. Additionally the mod will parse all of the information it is pulling in order to implement the data with the rules sets. This parsing, especially on complex information such as XML, will place extra strain on the CPU as well as holding the parsed data will again add to the needs of RAM. Finally the logging process depending on how it is configured has the potential to become very I/O expensive if too much information is being logged.

In order to determine if the resource usage was acceptable in the context of providing DDoS defense without sacrificing operation ModSecurity was setup for the first test. Here apache is setup on the mobile and ModSecurity is installed and configured with the OWASP Core Rule Set.[66] The rule set is designed to be a starting foundation for web server security with the module and needs little configuration to see results. The rule set states that it will assist in defense against HTTP level DOS attacks in both flooding and slowloris variety. For the testing the server is flooded connections over HTTP port 80 that are then left dormant to occupy connection space while more connection requests are sent as soon as the previous connection has been established.

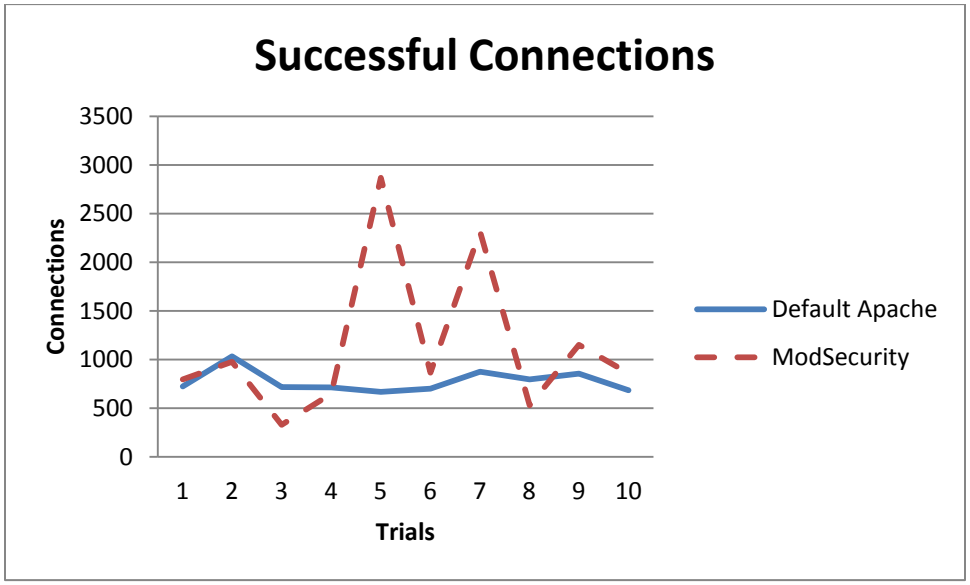

Figure 7. Mobile server connections for ModSecurity shown as total successful connections contrasting base install of Apache to the enabling of ModSecurity.

Here is the example of 10 trials run on the mobile server. First is with the default Apache setup and running Apache2 with no changes. And then run again with ModSecurity enabled with the OWASP Core Rule Set. This shows that the ModSecurity setup was generally able to process more successful connections while being enabled.

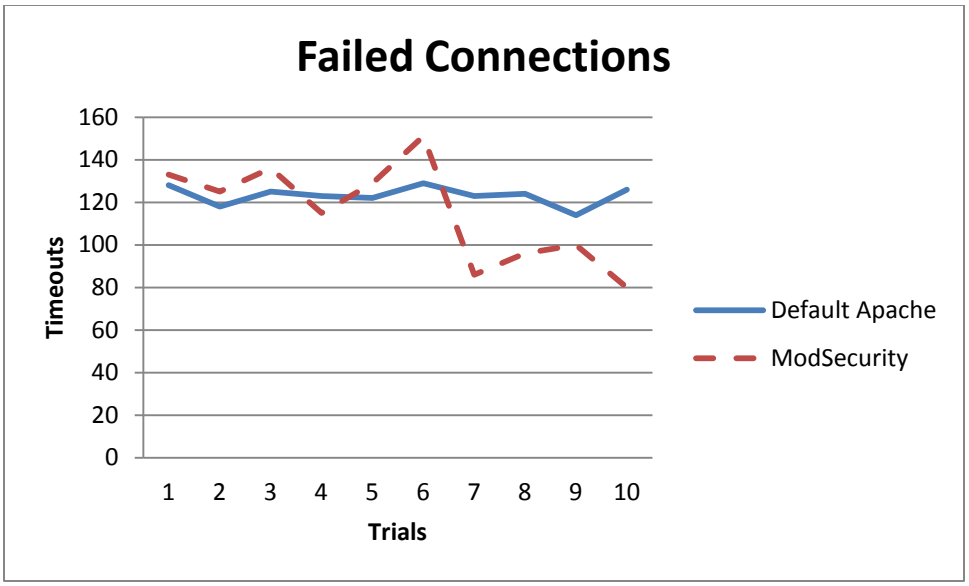

Figure 8. Mobile server connections for ModSecurity shown as total failed connections contrasting base install of Apache to the enabling of ModSecurity.

Here is the other side of the experiment in looking at the failed connections returned to the machines in timeouts. Ideally here the ModSecurity would achieve a more consistent lower value as the timeout represent connections that are not able to establish. Non established connections are a goal of DDoS in that users will be unable to access the targeted website. Ultimately what is being seen is that while ModSecurity is capable of generating more connections it is doing so at the expense of losing potential connections. Having not been able to achieve a reliable reduction in timeouts the server would still be vulnerable to DDoS as much as the default Apache. It is for this reason ModSecurity is not examined any further.

The final module to look at is ReqTimeout. This is examined for its ability to maintain operation for new connections while under a flood attack as well as detect and mitigate from slow attacks. This is a key distinction from the other two modules as they were not designed around monitoring for slow attacks which are rising from the ability to affect on network but also at application layer. [67] CPanel in fact specifically recommends ReqTimeout for defending

slow DDoS attacks. [68] ReqTimeout comes as a disabled module by default in Apache2, allowing end users very easy enabling and configuring for DDoS defense needs. ReqTimeout works by keeping track of all the servers established connections and monitoring for how long the connection has been open without either completing or receiving any further requests. The delay on how long a connection will be left open is configurable as is a minimum transfer rate.

Examination of ReqTimeout

ReqTimeout is examined further for its viability in defending against DDoS attacks primarily for its smaller footprint on resource usage especially in comparison to benefits offered. The test scenario is setup first on the Motorola razr and repeated for the Samsung nexus 4. Motorola runs a 1.2GHz CPU and 1GB RAM, and the Samsung 1.5GHz with 2GB RAM. The testing is to show connections that are successful versus the unsuccessful for the web server in the states of default Apache, then with ReqTimeout enabled to timeouts of 20 seconds, 10 seconds, and 5 seconds. Testing is not completed below 5 seconds as given the nature of a mobile connection lowering the timeout window too far would begin to drop legitimate users when signal strength is not the best on both the server and requesting users ends of the connection. The testing is again done to open as many connections as possible then no further transfer is requested. This is used as a hybrid simulation in flooding and slow attack as the connections are designed to flood and occupy all of the available connection sockets on the server while then requesting no further data. Actual slow requesting is not tested as the module is predefined t o handle that and would result in the same rate of flooding as only opening the connections. Not slow attacking also guarantees that the attacking machines are not expending any further resources on the connections once they open them. All testing is

completed in a local network setting to maximize the attacking users ability to generate requests.

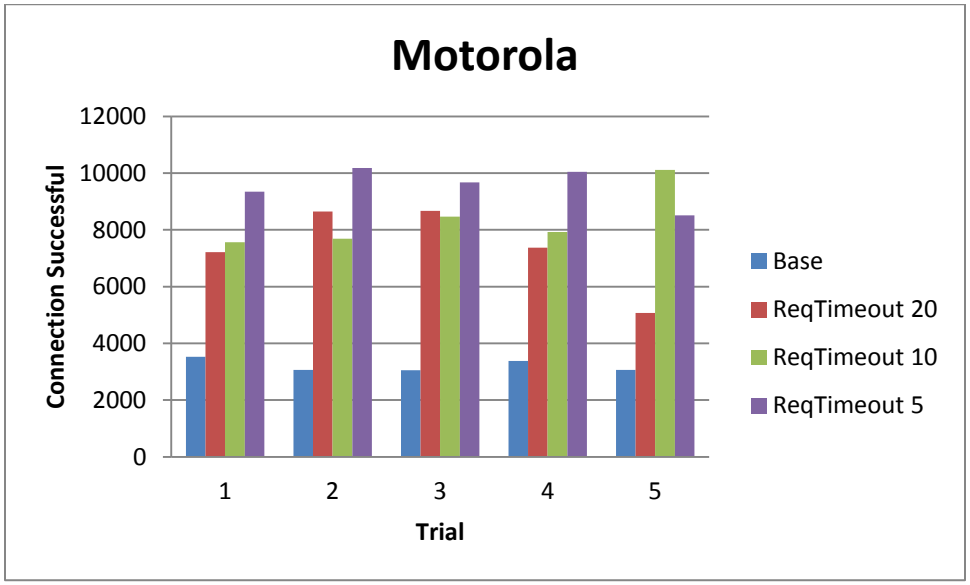

Figure 9. ReqTimeout total successful connections on Motorola.

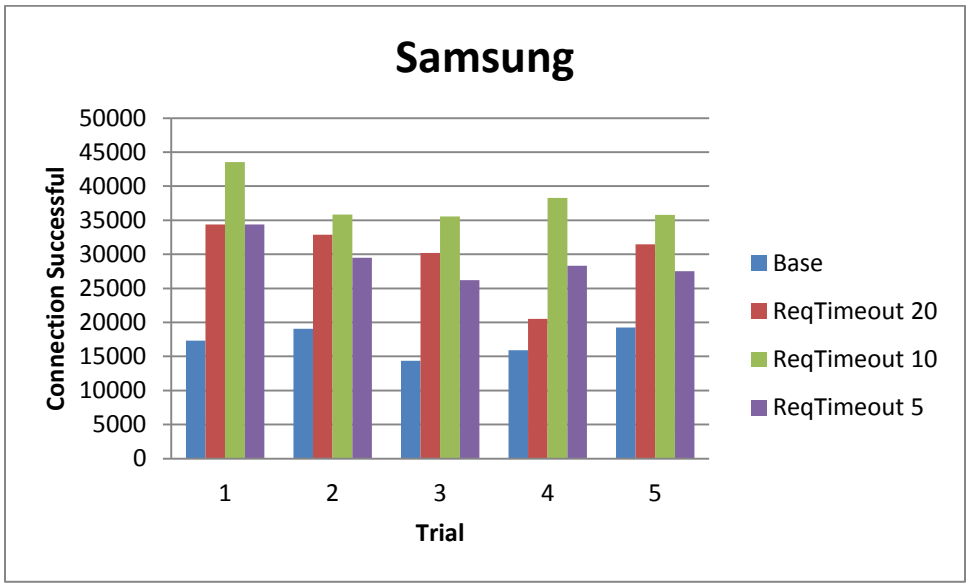

Figure 10. ReqTimeout total successful connections on Samsung.

These first tests for successful connections demonstrate the effects of ReqTimeout in granting the most possible connections. Both mobiles show a strong increase in effectiveness going from the base Apache install to having enabled ReqTimeout even at the timeout level of

20 seconds. The Motorola shows on average a 130% improvement while the Samsung 74%. The gains are not as significant for the Samsung and this is attributed to being a more powerful device, so by default it is more capable of handling and processing the connections.

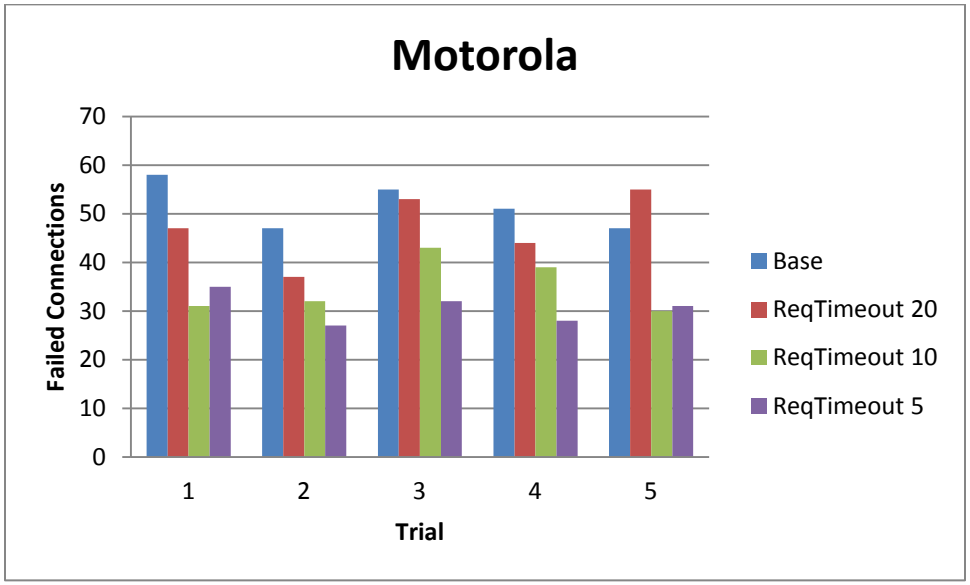

Figure 11. ReqTimeout failed connection total on Motorola.

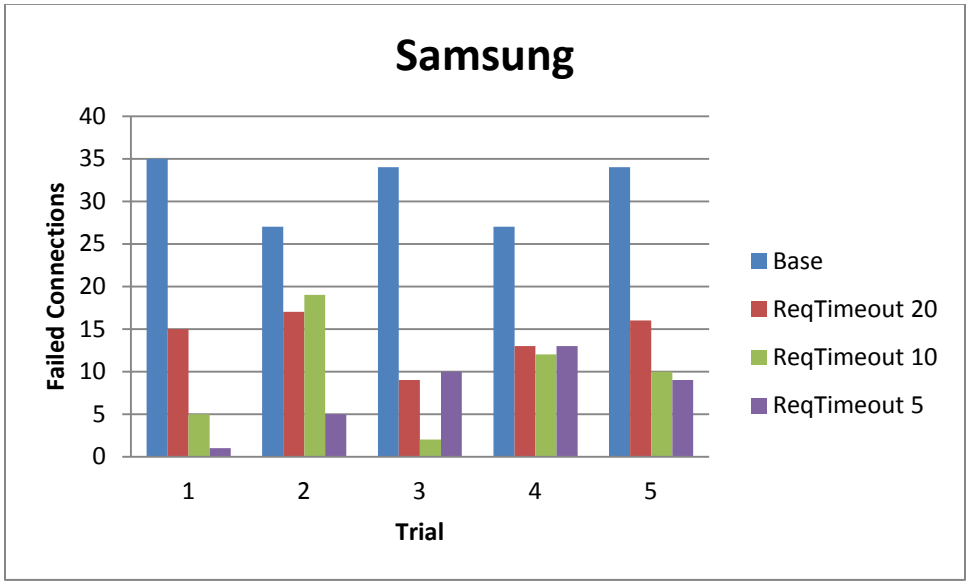

Figure 12. ReqTimeout failed connection total on Samsung.

These represent the failed connections during the same trial runs as the previous successful connections tests. Both mobiles show a decrease in the amount of connections that

timeout and result in failure. Here the improvements in moving from default Apache to a 20 second timeout are 8.5% for Motorola and 55% on the Samsung. The gains for the Motorola show a much more respectable 32% from default when the connection timeout is lowered to a 10 second threshold. The difference again can be attributed to the different phone specs. The Samsung is capable of serving more concurrent connections and thus shows a greater benefit in clearing non active connections at any point then the Motorola which is still suffering from the flood in keeping open 20 seconds connections. The average connections do well to show the advantage in enabling ReqTimeout.

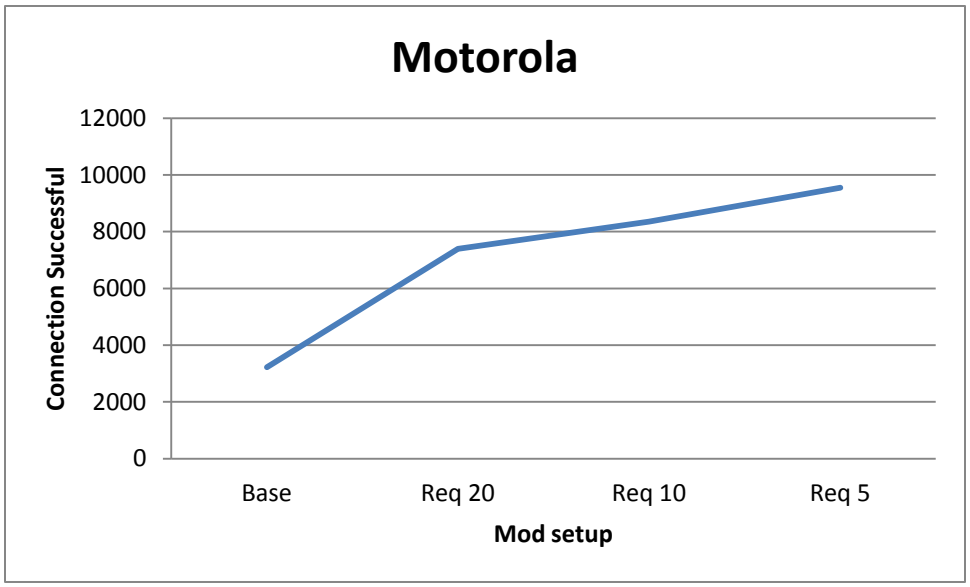

Figure 13. Motorola connection totals in average by timeout value.

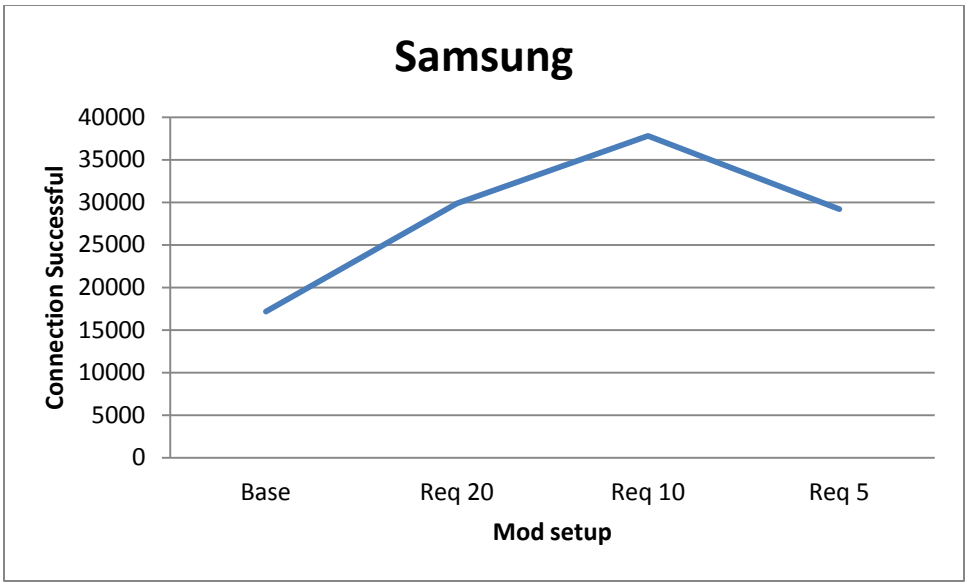

Figure 14. Samsung connection totals in average by timeout value.

The average number of successful connections is shown across each of the trials for the settings of the server. The most noticeable item here is the decline in successful connections on the Samsung when lowering the connection timeout from 10 to 5 seconds. This is a result of the mobile spending more resources clearing out the connections at 5 seconds when its capable of just generating the new connections it needs and ignoring them . The Motorola shows a steady improvement as on the lower spec phone the more sockets it can free up the better performance that will be achieved.

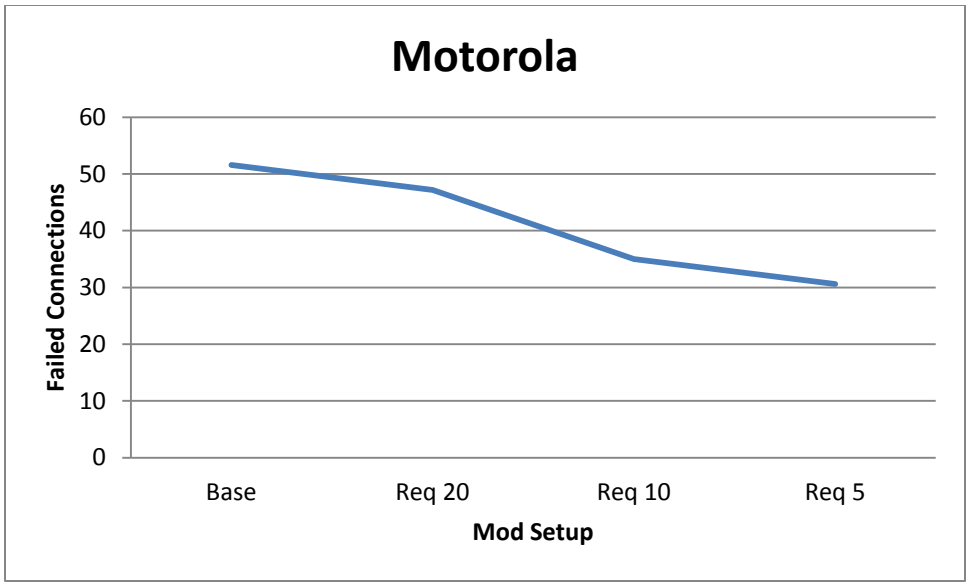

Figure 15. Motorola average failed connections by timeout value.

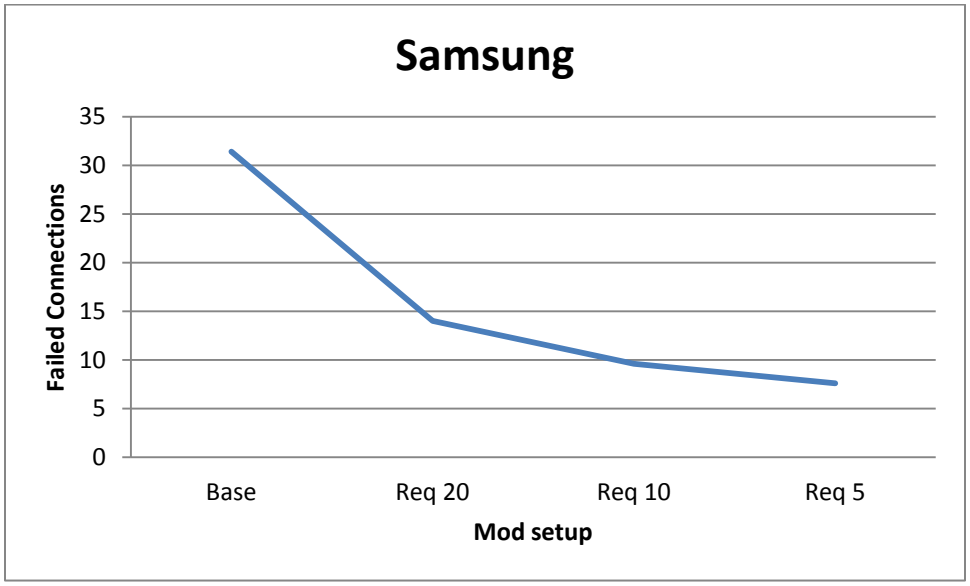

Figure 16. Samsung average failed connection by timeout value.

On the timeout side of things the mobiles both show steady improvements in the number of connections that fail as the timeout setting is lowered. There exists a similar pattern to successes that the Motorola doesn't realize its best gains until lowering the timeout down to 10. Here the Samsung is still showing improvements from 10 to 5 seconds unlike the successful connections. This is further proving that the resources to keep open for all incoming

connections are succeeding at the expense of the rate at which new connections are actually being established.

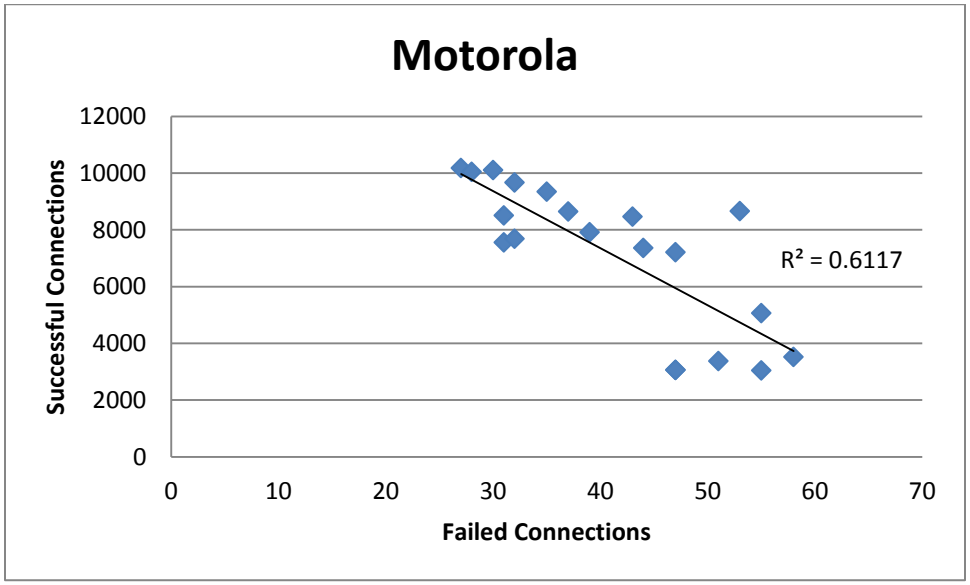

Figure 17. Motorola number of failed connections by number of successful connections.

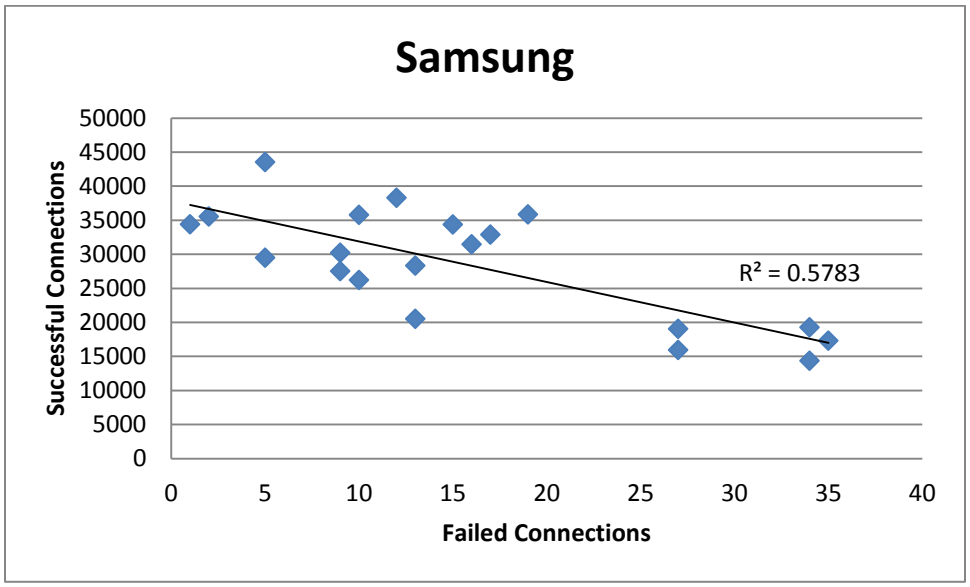

Figure 18. Samsung number of failed connections by number of successful connections.

Here the plots for each of the tests under changing ReqTimeout settings are shown as successful connection against timeouts on that trial. Both mobiles are displaying the relationship of increased number of timeouts is correlated with lowered value of successful

connections. What these do not take into account is cases where a connection is successful but will require more time to establish. Of note also is the Motorola operating even at lowered values of timeout is not able to establish any volume of connections without incurring over 20 failed attempts.

To check on the viability of the Samsung with increased traffic loads the tests are run again with double the machines creating connections.

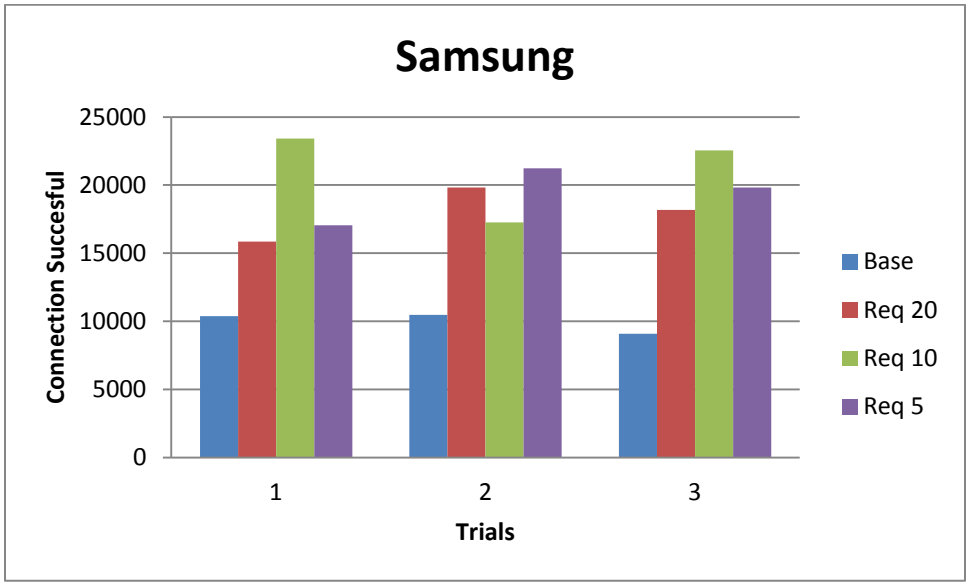

Figure 19. Samsung large attack successful connections.

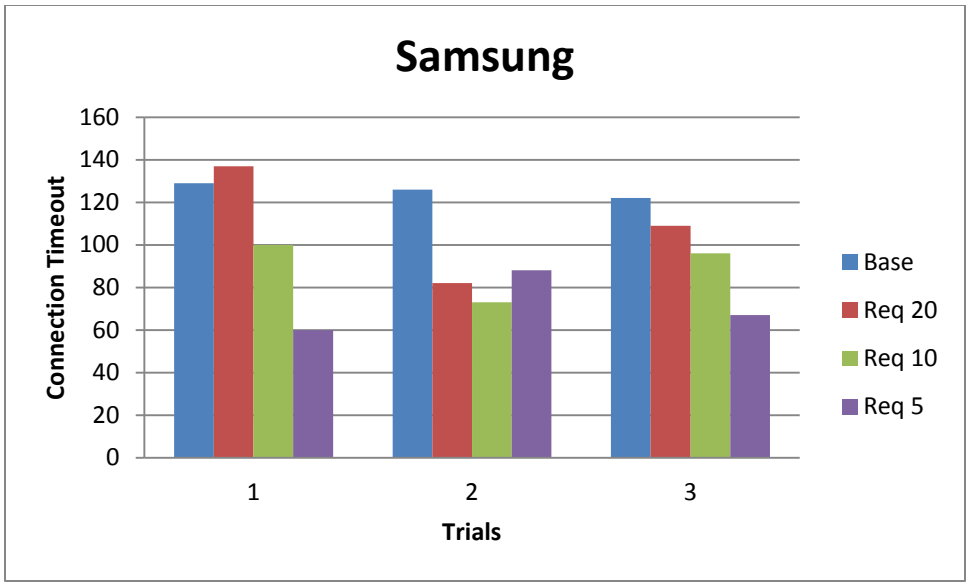

Figure 20. Samsung large attack failed connections.

With the increased attack volume the Samsung shows results more similar to the original tests run on the Motorola. This provides that the enabling of ReqTimeout is advantageous, the better benefit is found at lowering the timeout threshold to 10 seconds.

Summary

In conclusion the ReqTimeout module is easily installed and configured to establish a base level of defense against TCP flooding attacks in the case of generating the socket connections successful at over a 90% rate. A configured timeout value shows the potential to provide over 100% more successful connections in the same time frame as the apache server not running the ReqTimeout Module. The limitation though remains in this module suited only to drop TCP connections and has no ability to detect a DDoS attack and mitigate accordingly.

CHAPTER 4

MOBILE DDoS DEFENSE FOR HTTP SERVICE

The previous testing as all has been to demonstrate the server's performance in opening the greatest number of TCP sockets as possible in the fixed time. This testing was sufficient to show the scenario of network layer only connections but that does not do a sufficient job of demonstrating a more practical example of the servers' usage during a sample attack. To establish a better idea parameters going forward will always be for the legitimate user to request a page from the server. To establish a success the page must be transferred in full without a timeout in establishing the connection which would be similar to the TCP only checks. But also the page cannot have a failure during the sending of the page objects. Either of those will result in a failed connection for the testing. Also the transfer time for the pages will be looked at a significant slowing of page transfer performance can be a metric for successful attacking.

The next set of tests is run to show the results in having a combination test of running a 6 machine attack against the server while having a different 6 machines actually connect and requests the full webpage. Here the focus is on testing for the number of failed connections as well as the time required to complete a page. Two variations are run on the pages, one is a page with a high resolution image and the other is a page with multiple small text sections. Both are looked at for examples of if the content in the page could affect the speed at which the page is served.

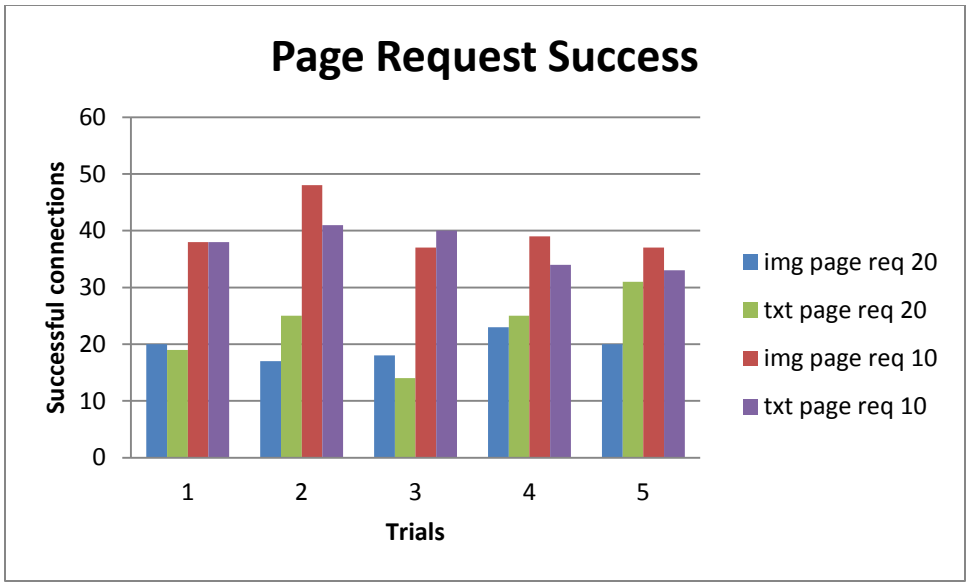

Figure 21. Page request connections successful for text and image page at 20 or 10 second timeout value.

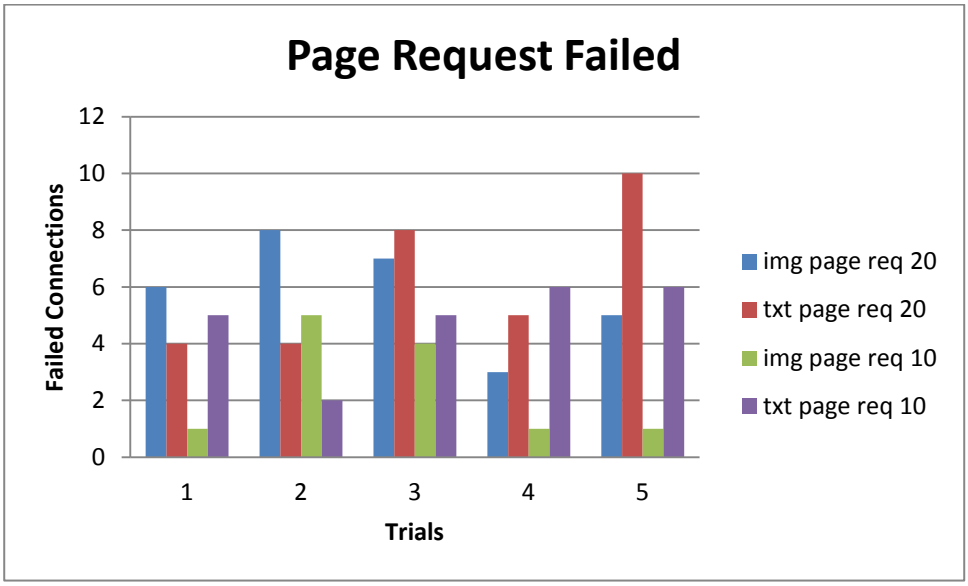

Figure 22. Page request failed for text and image page at 20 or 10 second timeout value.

These graphs repreesnt the successful and failed connection count on the page requets in achieveing the connection and transfering the complete page. Here the results show a distinct difference in result with the timeout being lowered. This is expected as finding sofar has shown under stress the mobile server performs better with the lowered timeout.

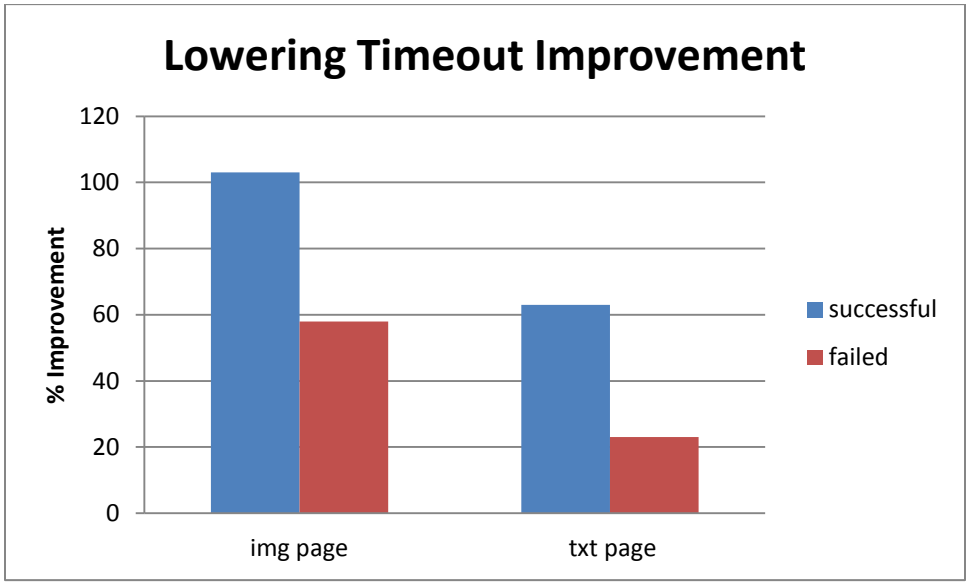

Figure 23. Page request timeout improvements from lowering the timeout level from 20 to 10 seconds.

This shows the percentage improvement achieving in lowering the timeout from 20 to 10 seconds. The image webpage performed better from the adjustment in both successful and failed connections. Both though show improvements that place value in the lowered timeout value while the server is under attack.

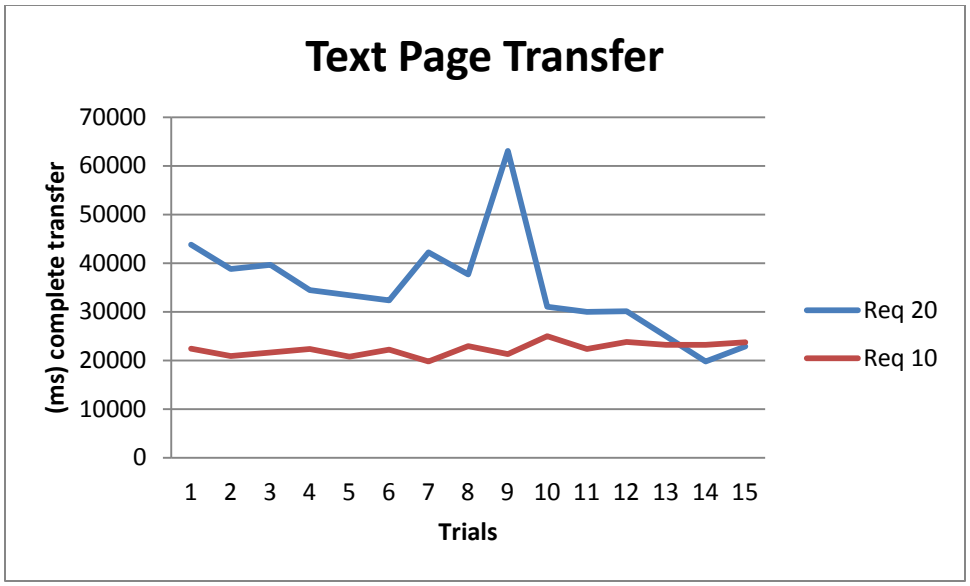

Figure 24. Text page delays to transfer a complete text based page at both 20 and 10 seconds for timeout value.

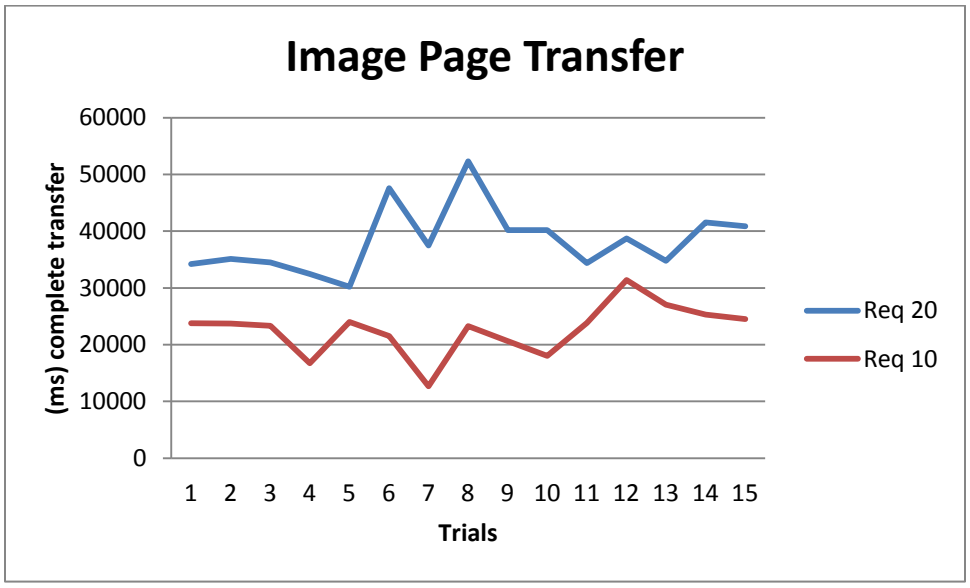

Figure 25. Image page delays to transfer a complete text based page at both 20 and 10 seconds for timeout value.

These show the average time to complete a page request and receive the whole page across the trials. The difference in times is significant when adjusting the timeout value down from 20 to 10 seconds. Also of note here is the volatility in the 20 second timeout lines. The page transfer times have a much greater range than is seen in the 10 second timeout lines. That

volatility is a signifier of the success of the attack creating unstable server connection and operations.

Figure 26. Page transfer time average across all trials for 20 and 10 second timeouts for both text and image pages.

Finally this is the representation of the average time in milliseconds to complete a successful page request while the web server is under attack. Using the timeout value of 10 seconds there is no real difference shown in the text page versus image page. The gains from changing the timeout value are 36% on text pages and 45% on image pages. This reiterates that the transfer while not under duress is stable but the image file transfer tends to struggle more while the server is under attack.

Mobile DDoS Defense Proposal

The mobile web server is currently not equipped well enough to handle any form of coordinated DDoS attacks. While it is found that the correct settings on the ReqTimeout module can reduce the effectiveness of a flooding attack or a slowloris, the server is still easily exhausted in serving static pages to many users simultaneously. Phone technology and

hardware is progressing constantly and can certainly alleviate some of the weakness as seen by the comparatively better results achieved on the Samsung with the higher end hardware specs. Ultimately though a better solution must be found to have any real chance of mitigating small or medium scale DDoS attacks. A large scale attack is not going to be defended against on the mobile as the flooding can quickly overwhelm the data connection rates alone for a mobile device.

The first major step in looking at how to better defend a mobile web server from DDoS attacks is understanding the switch to IPv6 occurring in the mobile phone networks. The internet topology for data connections over IPv4 required mobiles to route traffic through first its current tower and then to its home agent that would direct the traffic back to the mobile. With IPv6 mobiles are able to access internet resources directly from communicating with their current tower connection.[69] This benefits the servers in the first for achieving a static IP address that can be accessed from users without the constant changing of location requiring a new address to find the web content. As of 2009 Verizon has mandated that all its phones that are to use the LTE network must support IPv6. [70] In terms of what this means for defending against a DDoS attack against the mobile, the effect of having a static IP address means the testing results from modifying Apache will not be dependent on network provider routing.

The proposed defense strategy is designed to make use of a server end only solution to defense but make a better use of available resources. Knowing the limitations of the mobile itself the proposal is the have the phone monitor its number of incoming connections. If those connections in a minute ever exceed a set threshold then the server will recognize a potential DDoS attack and begin attempts to mitigate the attack. The first step is to monitor connection

count on a 5 minute interval. This is due to the presence of a potential attack is found quickly when exceeding the threshold and the defense will only recheck every 5 minutes as detecting the attack has finished too late will not have detrimental effect to the legitimate users the same as detecting the start too late would.

```
http_request.c

establish IP address table for incoming connections

capture incoming IP address

compare IP address to all addresses listed in IP table

if( table entry matches current incoming )

      if( count on matched IP > threshold )

            drop connection

      else increment count on matched IP

else insert IP address with count of 1 in next open table
location and

      increment next count

on exit of timer reset tables to blank


reqtimeout.c

if( timeout variable is false)

if( total count exceeds threshold  )

      reduce timeout

      set timeout variable to true

else increment connection counter

on exit of timer reset timeout variable to false and
reset total count
```

The defense mechanisms will enable when the attack is occurring. The first mechanism is the lowering of the timeout down from the always enabled 20 seconds. This is done as the testing data showed consistent gains in connection handling while the server was under stress by reducing that value. This is accomplished in the reqtimeout.c file, this is the file for where ReqTimout is found. Here the connection counts are monitored on a global basis. The timeout lowering is completed by creating each new thread after the threshold is exceeded by overwriting the structures ccfg timeout member to be the new value not found from the configuration file. This step occurs on the TCP layer in order to fully work against SYN-ACK and slowloris type attacks.

The next step is to begin populating a table that will store the incoming IP addresses along with the number of requests that IP address is associated with. This work is completed in the file http_request.c where incoming IP addresses are found in the struct conn_rec. The table size is not predefined as that will be a source of potential difference based on the capabilities of the particular mobile. The table though regardless must be of sufficient size to capture a DDoS so a recommended minimum would be of 50 entries. Knowing a mobile botnet of 29 phones can begin to affect a server the table must at least account for that case and have room for legitimate users sending requests during an attack. A running list also must be kept on the table to know the order updated on the tables. So when an incoming IP is to be logged it will first check if already exists and increment its counter. If it does not the table is checked for an empty cell and places the IP in with a counter of 1. Otherwise the update list will be checked for the last entry to have been incremented and the incoming IP will be inserted into that cell with a counter initialized to 1.

If a counter in the IP table on the mobile reaches the threshold then the mobile has recognized an attacking IP address and needs to directly stop connections from that source. This blocking will occur at the HTTP level and is accomplished by setting the ap_conn_keepalive_e variable to AP_CONN_CLOSE signaling apache to close that connection. Here a flood of page requests would be stopped from clogging bandwidth in their constant use of the server uploading content to the attacking connections.

At the completion of every 5 minute cycle the connection counters reset. This will allow for the mobile to properly monitor the incoming attack. The IP addresses sent to the network for filtering will still be filtered so there is no work being repeated in detecting malicious sources. Once the mobile detects the attack is no longer greater than the threshold of connections per second the local counters and IP table are all reset.

The threshold and timeout reduction values are not given as a set value. The results in testing ReqTimeout showed that the reduction to 10 seconds on the timeout offered a general defense level capable of mitigating from DDoS attacks with success. Those tests were done though in the static setting of starting and finishing with the same timeout value. Here some tests were done to lower the timeout and to 10, 8, and 6 seconds. The testing was all done at high level flooding of 5,000 connections with 3 machines requesting pages as legitimate users.

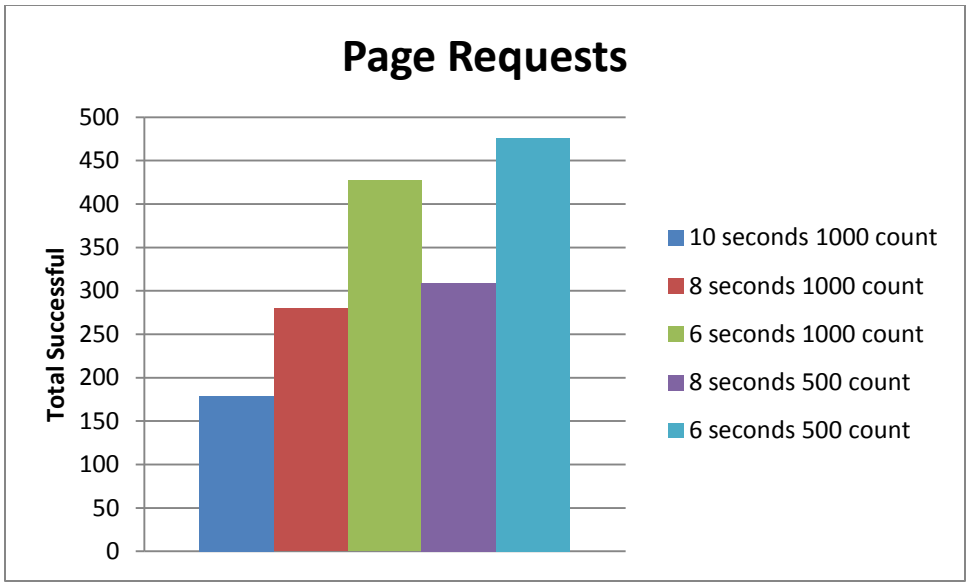

Figure 27. Dynamic timeout adjust levels showing successful page requests completed.

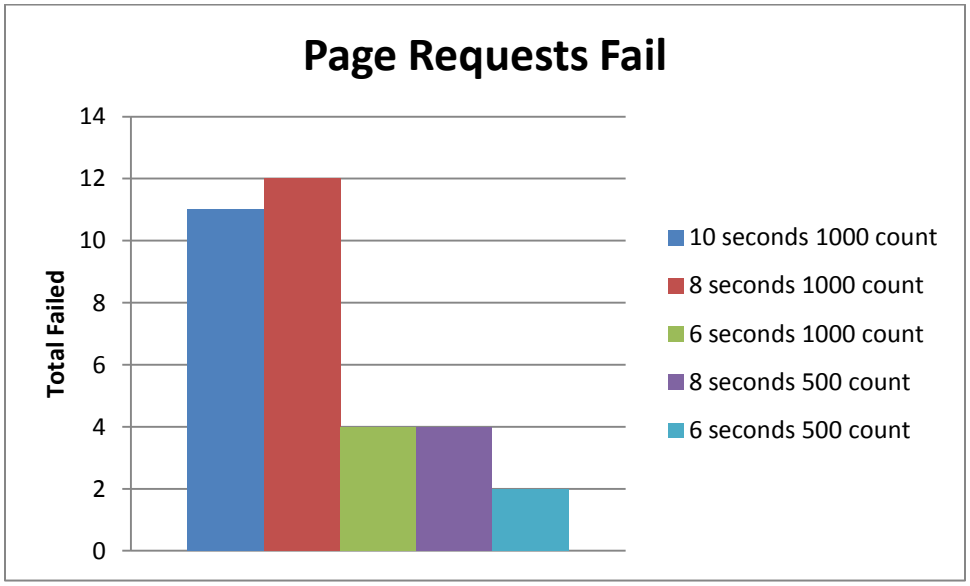

Figure 28. Dynamic timeout adjust levels showing failed page requests.

These results in modifying the timeout level and the number of connection before the timeout is lowered. This shows that the lowering of the timeout down below the previously successful level of 10 seconds shows substantial improvements. The failures also tend to benefit from lowering the timeout. What is happening is that by lowering the timeout below the 10 seconds the server is saying that knowing it is under attack it is acceptable to potentially

lose slow legitimate connections as the benefit is found in greatly increasing the legitimate connections successful while under attack. Having the dynamic adjustment difference is also looked at from lowering the total connection count before changing the timeout. Here similar jumps in success are seen at either the 8 or 6 second timeouts. This is as expected that the lowering of the count allows the server to begin combating the DDoS at an earlier time. The improvement on 8 and 6 second timeouts in lowering the connection counter threshold was 10% and 11% respectively. This means that lowering the timeout at an earlier time period in the attack should generate successes at similar rates no matter what the timeout is ultimately lowered to.

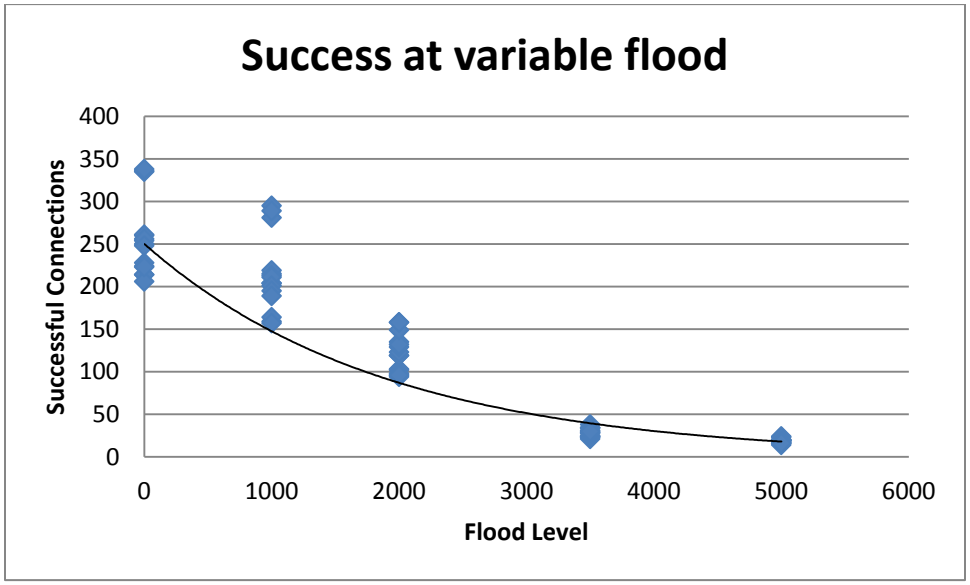

Figure 29. Controlled flood level effects on successful connection running proposed dynamic algorithm.

This is the results of running TCP flooding against the algorithm setup to operate at the 500 connection count threshold and 6 second timeout adjustment. This shows the exponential decline in successful connections seen as the volume of attack flood is increased. The server and algorithm show fairly steep decline in the performance approaching 3000 attack

connections. The decline in the rate of lowered successful connections seen as the flood level increases is attributed to the threads becoming increasingly saturated by the open sockets and the further attempts to open new sockets will begin to delay as the slots fill. The base level of 0 flood is nearly similar to the 1000 flood rate indicating the server can near optimally survive under the 1000 attack connections and show no effect on legitimate users.

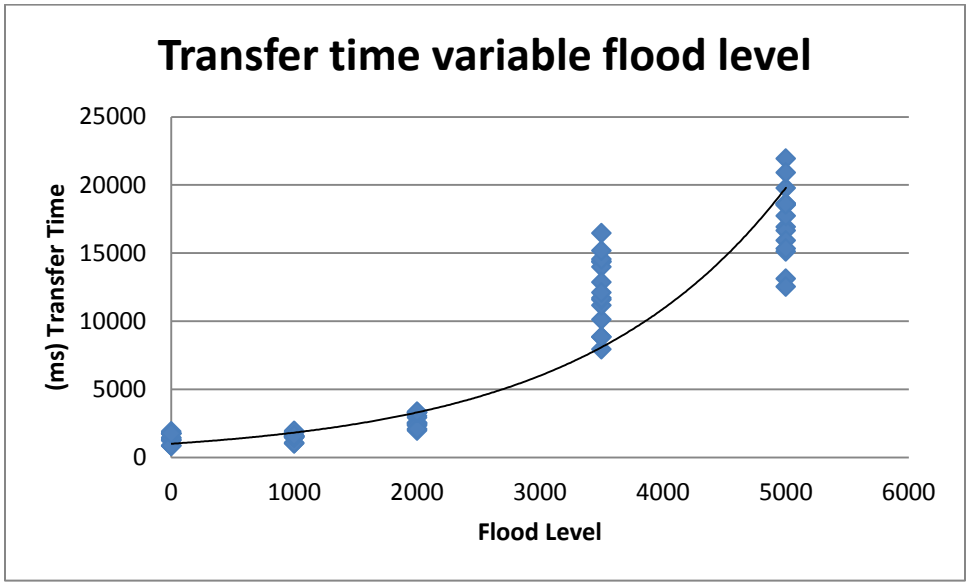

Figure 30. Time in ms to transfer complete page at different flood levels while running dynamic algorithm.

This is from the same data as the previous figure for testing the 500 count connection threshold and 6 second timeout adjustment at different levels of TCP flooding. The results here are displayed for the total time the legitimate page requests are taking to completely transfer all of the page data. The results are demonstrating an exponential increase in time take for successful transfers. This is from the effects of the server being under a and more overwhelming level of flooding requiring additional resources and bandwidth, this begins to severely limit resources given to the actual transfer of web page objects to the established

connections. The delays grow very slowly under the lower level flood volumes and being to increase rapidly as the flooding affects the server.

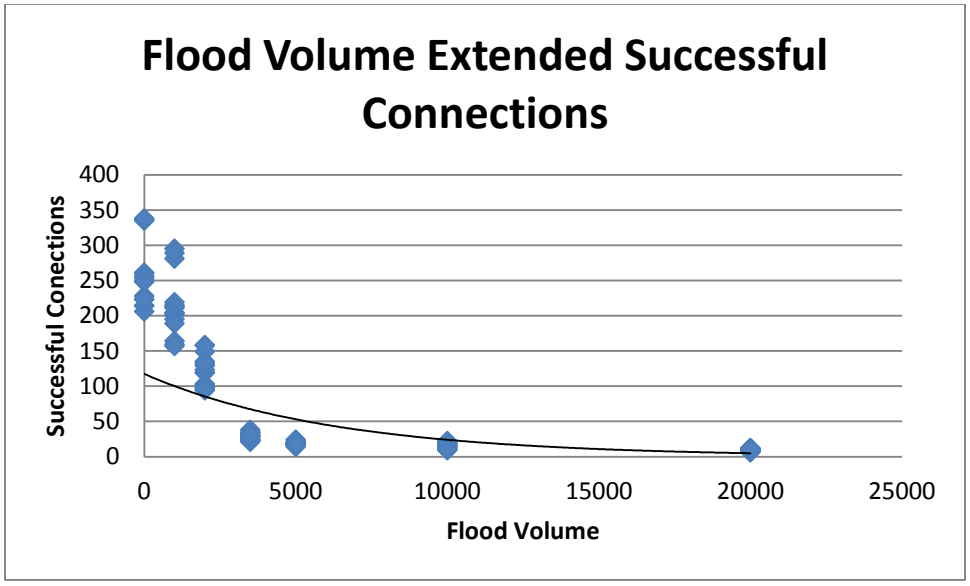

Figure 31. Extended levels of flood volume effect on connections.

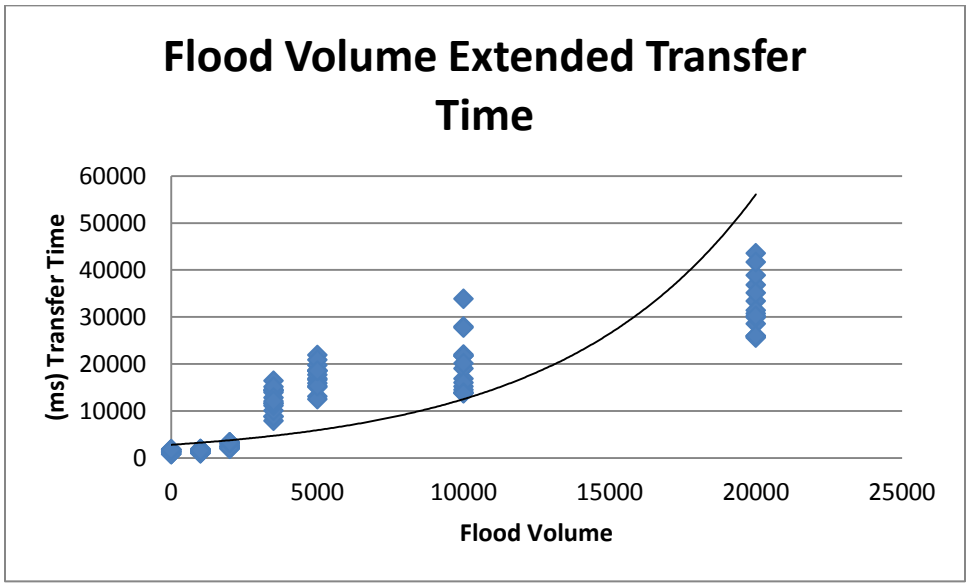

Figure 32. Extended flood level effect on transfer time.

These graphs show the effect on the trend when the flood level of the TCP attack is extended out to greater volume than the previously tested controlled rate floods. Again the

results represent on a per machine basis the amount of connections achieved as successful first and then shown as the total time to transfer the page for each iteration. The additional flooding levels bring an interesting not to compare when viewing the trend lines. First that the Successful connections graph displays the similar fashion but is much flatter to begin representing a better picture to the weak effects of lowest level flooding be nearly handled completely by the server without effect. The extended graph for transfer times paints the same picture as the trend is still showing sharp increase. Although here the transfer times being to decline in the rate of increase. This is attributed to the flood volumes approaching a saturation level quickly that then will show more consistent effects on transfer speed for the more rare connections that are established and capable of completing the transfer. On both extended graphs the data points change sides of the trend near to 4000 connections on the flood. This is an ideal base level for flooding to represent an attack on this mobile server. Additional units over that volume will assist in exhausting resources but less would be invalid in creating a true attack environment.

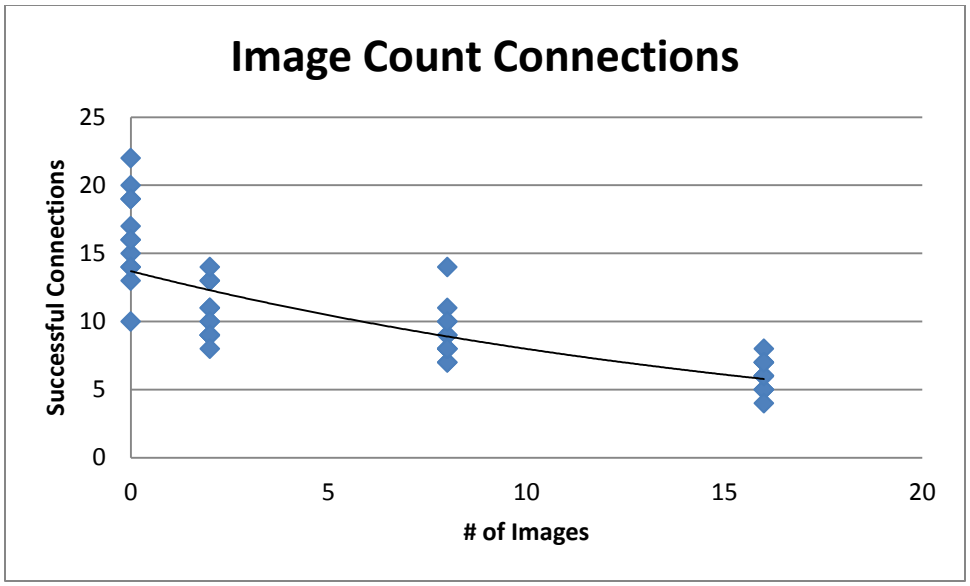

Figure 33. Successful connections for changing page content.

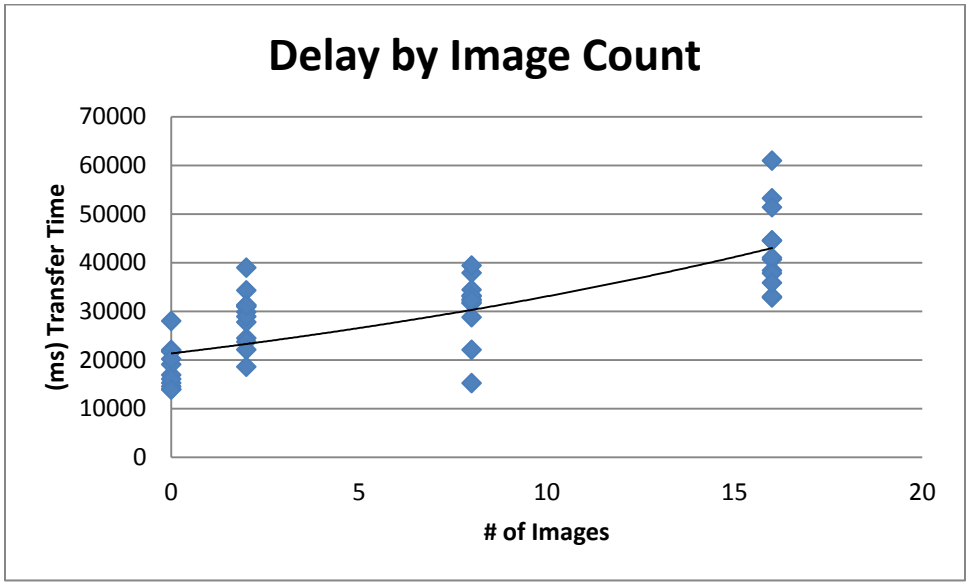

Figure 34. Transfer time at different page content.

These figures demonstrate the relationship found from controlling the flood level at the over 1,000 TCP connections while requesting the page content. The page was structured using image files to load as separate objects. The number of images is then tested showing the successful connections of how many total pages were pulled down. Each image was

approximately 350 KB. This has a fairly linear and steady decline in connection count as the number of images was increased. Looking at the transfer times from these tests also show a more linear increase in times. This all means that as the attack is occurring the connections are transferring at roughly the same rates once established independent of the number of objects. The successful connections though lowers as the delays are higher but also from the longer transfer time introducing greater chance of that connection failing.

Multi Layer Defense

The next step in creating the best DDoS defense algorithm is to adapt to communicate through multiple levels of networking together. So instead of leaving the connection count monitors completely separate from the TCP and HTTP layers a single connection counter is introduced at the HTTP level. This counter will then be capable of also signaling the TCP layer dynamic timeout adjust to lower if an attack level is discovered. This is accomplished through adding a global variable ccounts into the header file for http_request.c . Then on each incoming HTTP request on the apache server the variable ccounts will be incremented by one. In the reqtimeout.c file then the counter can be seen as http_request.h is an imported file by default. In the same check for total tcp connections to exceed the threshold will be OR checked with if the ccounts of HTTP requests has exceeded its threshold. These thresholds can be set independently to allow a smaller number of HTTP requests to trigger the timeout lowering.

The multi layer approach is designed and added to better assist the apache server in defending against a greater number of attack types. The best example of a new attack defended by this addition is the slowloris attack. Slowloris is an application layer attack that

uses TCP sockets being kept alive to exhaust server resources. The slowloris attack would not previously been efficiently detected as the TCP counter wouldn't be reaching threshold to lower timeouts and free up sockets. And an HTTP counter that may have triggered did not have a defense mechanism attached that was designed for clearing out sockets that are associated to malicious connections.

In sharing information from the application layer and the TCP layer the apache server will stand a better chance to not only detect a DDoS attack in progress but then to properly launch all of the defense mechanisms in place to mitigate the attack more successfully.

Summary

The proposed defense solution will be effective in mitigating DDoS attacks on the application layer. The network layer attacks could still prove effective against a mobile phone in high enough volume as the defense strategy against those requires a firewall or better control over the additional routers before the server to defend.

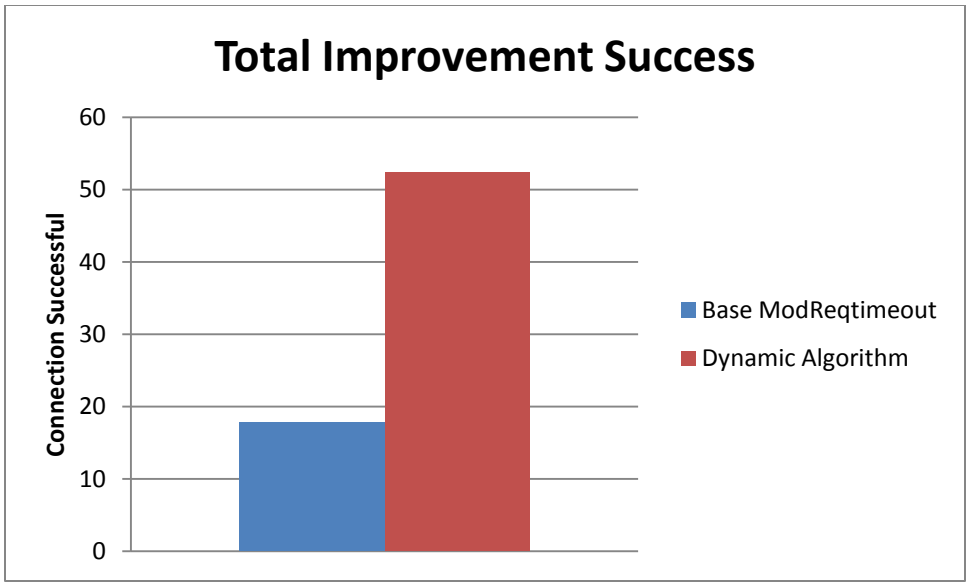

Figure 35. Total successful connections averaged compared from new dynamic algorithm to basic timeout setup.

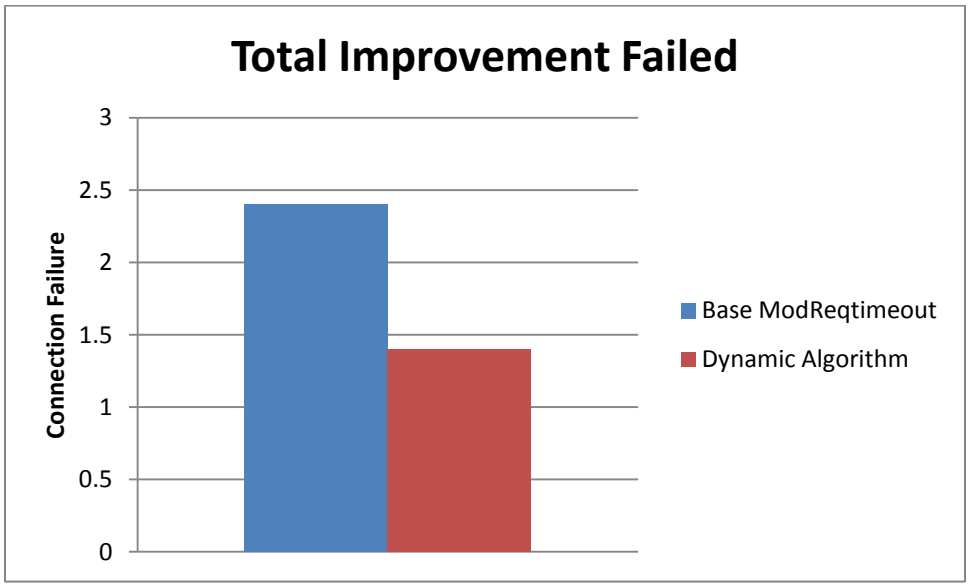

Figure 36. Total failed connections average compared from dynamic algorithm to basic timeout setup.

The final results of testing the dynamically adjusting timeout algorithm to defend better shows significant differences when compared to the server when originally established with just the ReqTimeout module installed and setup to run with 20 second timeouts.

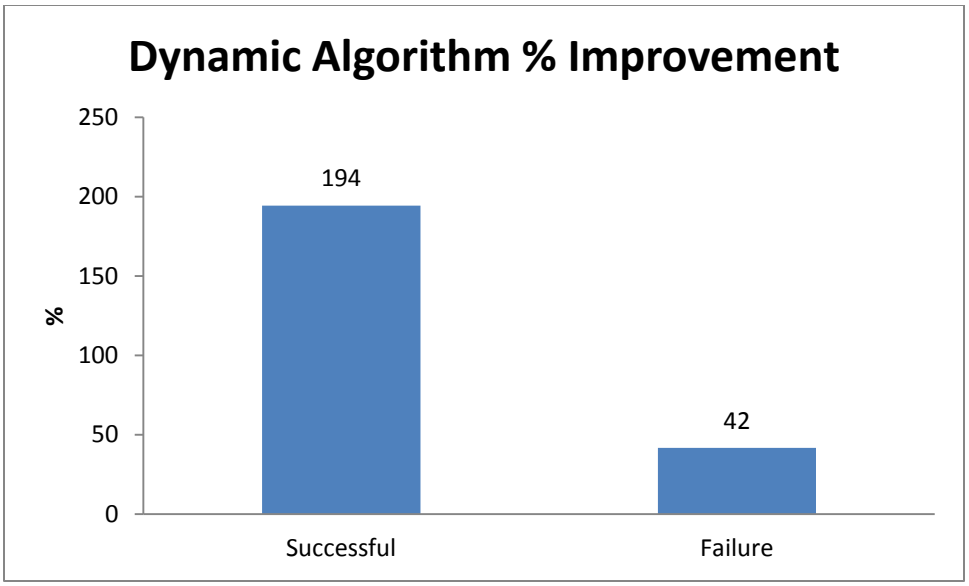

Figure 37. Improvement percentages for using dynamic algorithm.

Shown as improvement by percentage we see that the successful connections have a 194% greater rate when the dynamic algorithm as added while the failures are 42% lower. These tests were done by setting the algorithm to pass the TCP threshold at 500 connections and lower the timeout value to 6 seconds when the limit was reached. Both the dynamic and base were tested with a TCP flood level set to 5,000 connection attempts.

In the end the defense of a mobile web server from a DDoS attack is going to require a multifaceted approach. There is no single solution that will out of the box protect a user. The addition of a multi layer approach will assist in mitigating attacks on both the TCP level in opening sockets and application layer attacks. Ultimately as the hardware capabilities of phones advance and users begin to experiment running their own web servers the tools must evolve to be capable of detecting and mitigating from the known types of DDoS attacks that could affect the server. Ideally these defense tools could also be used to help detect and identify the presence of infected mobiles serving as bot and stamp out the issue closer to the source.

CHAPTER 5

CONCLUSION

The proper configuration and enabling of the ReqTimeout module showed certain promise to assist a mobile web server mitigate DDoS attacks. However even though these connections are successful and completing does not mean that service is ideal under the attack. The improved rates under timeout of 10 seconds for completing a page request are just over 20,000 milliseconds. 20 seconds would be unacceptable delays for a user to access most websites. A single connection requesting the page repeatedly with no additional flooding connections averages around 5 seconds to complete the transfer in comparison.

Knowing that bot can be created on an Android device that is capable of generating just over 4 page requests per second to well established servers. The capabilities of botnet can quickly overwhelm the defenses available to a web server on a similar mobile. Even under the most ideal setup of the Samsung mobile with 10 second timeout enabled the amount of average connections handled was 37795. That represents connections handled sent from 6 machines for 5 minutes. So the server in that case was handling 126 requests per second. Given the rate a bot was tested at 4.3 requests per second, the equivalent flooding rate would need a botnet of just over 29 mobiles to being to affect the server.

Distributing the botnet to have control over 30 phones does not represent a challenge or concern for the ultimate goal of disabling a web server through the use of DDoS. The mobile server with no ReqTimeout demonstrated over 4 times more failure in handling its reduced capacity for connections. The successful connections on the default Samsung represent 57 per minute. Which would be susceptible to being replicated by a botnet of just 13 mobiles. Finally

the unsecured Motorola only demonstrated itself to handle just under 11 connections per second. That would merely need the coordination of 3 phones to begin exhausting resources and succeeding as an attack.

Finally in order to affect the mobile web servers ability to serve pages in a reasonable time we know the testing showed a flood of 6 machines was enough to slow the page request completion times to speeds that would be unacceptable to users. With the limiting factor of the upload processing a mobile botnet would also be able to accomplish this task in less than 10 phones.

The dynamic algorithm proposed resulted in improvements on connections handled from the default configurations. The benefit to using the dynamic timeout value is that it can offer better performance for both during attacks and when the server is not under attack. An optimally selected static timeout value on the other hand was forcing compromise in either losing slower connections during normal operation or to not be tuned to best drop malicious connections during an attack.

Future Work

The defense algorithm is successful currently against smaller scale DDoS attacks and is designed to work against TCP or HTTP layer attacks. Improvements could be looked for in changing the behavior of the defense mechanisms by monitoring the rate of incoming connections with multiple levels of connections resulting in different mechanisms activating instead of the current method of clearing a single threshold value and activating all the defenses. One such case could be to keep a separate IP table that keeps attacker IP addresses

after an attack has completed. This table could then be used as a blacklist at an early stage of DDoS detection to mitigate a repeat attacker.

Further testing can also be looked at in establishing different modifications on the DDoS attack flood being sent to the mobile. An example being more application layer attacks that would be focused solely on crashing the server without being mitigated as easily from the TCP layer defense adjustments.

The mobile were not examined for better performance in the instance of a genuine large volume of users all legitimately trying to access pages. In such a case the mobile server would be expected to quickly struggle to provide content to all of the users. A quality of service mechanism would be ideal to kick in under such conditions. The mobile server would need to first detect the connections were in fact not a coordinated attack. The method best to fit in with the timeout adjustments would be to have a second lower HTTP threshold level that adjusts upload rates down to all connections when the threshold is passed. And then if the original is passed the timeout adjustments are made. Simply the lower threshold would be just traffic detection while the second remains the flag for a DDoS attack in progress.

The main limitation in modifying the algorithm comes from the inherent limitations of running the web server on the mobile phone. The mobiles are currently only capable of processing so many threads which is what allows the flooding of connections to quickly occupy all of the threads. Therefore the defense strategies must focus on detecting and removing the malicious connections as fast as they can be discovered. Users running a mobile web server will not have reliable access to a configurable firewall or edge router so all the defense mechanisms must be kept on the local server. This means caution must also be taken that the defenses

added to the mobile are not so robust that the computational needs would exceed the expected gains from defending a DDoS attack. Phone hardware is advancing continually though not enough to rely on that to be the sole solution. A more native basis for running servers on mobiles will also assist in improving performance under attack. Late 2014 is to see the first commercial release of Ubuntu phones running a Linux as the main operating system. This will allow the server better service as the Linux isn't being placed onto Android devices.[71]

Along with monitoring the computational needs of more advanced DDoS defense mechanisms the mobiles can be tested for server performance with any amount of added defense for the affect on battery consumption. The mobile web servers advantages of providing access in ad hoc settings at the users location will result in scenarios where the user is not able to constant power supply available to the mobile. Not only can adding extra processing to the server require potentially more battery usage. A DDoS attack on the mobile server could be successful even against well tuned defense algorithms if the connections flood for long enough duration to run the mobile out of battery completely taking down the server. Monitoring defense algorithms under longer attacks for battery usage could result in the need for unique mechanisms to prevent a mobile from running itself out of power.

The final path for work to look into is a properly configured ad-hoc network with multiple mobiles capable of all serving the same web content. Testing would need to be done on balancing where outside incoming connections would go between the mobile servers as well as the mobiles ability to communicate with each other and how that will ultimately change the needs of the defense algorithm to make use of additional servers without leaving each server as an independent operation.

The goal of the research was to test for the viability of using a mobile phone to serve as a personal web server that would then be capable of still offering some level of acceptable performance to a legitimate user when under a DDoS attack. Running a defense against DDoS was also designed to require the least amount of resources in order to be successful. The resulting usage of the ReqTimeout module and dynamic adjusting algorithm alongside a temporary IP address blacklist accomplish these goals. Securing mobiles against DDoS attacks going forward thus can be accomplished through creation of specific defense mechanisms that while similar to common defenses will account for the unique platform of a mobile phone.

REFERENCE LIST

1. Criscuolo, P. J. (2000). Distributed Denial of Service: Trin00, Tribe Flood Network, Tribe Flood Network 2000, and Stacheldraht CIAC-2319 (No. CIAC-2319). CALIFORNIA UNIV LIVERMORE RADIATION LAB.

2. Mirkovic, J., & Reiher, P. (2004). A taxonomy of DDoS attack and DDoS defense mechanisms. ACM SIGCOMM Computer Communication Review,34(2), 39-53.

3. Dobbins, R., & Morales, C. (2011). Worldwide infrastructure security report.Arbor Networks, Ann Arbor, Michigan, USA, Tech. Rep, 7.

4. Prolexic Technologies, Retrieved from http://prolexic.com/index.php_knowledge-center/frequently-asked-questions/index.html

5. Satyanarayanan, M., Bahl, P., Caceres, R., & Davies, N. (2009). The case for vm-based cloudlets in mobile computing. Pervasive Computing, IEEE, 8(4), 14-23.

6. Khan, A., Othman, M., Madani, S., & Khan, S. (2013). A survey of mobile cloud computing application models.

7. Conit, Marco., Giodano, Silvia. (2014, January). Mobile Ad Hoc Networking: Milestones, Challenges, and New Research Directions. IEEE Communications Magazine (pp. 85-96). IEEE

8. Conti, M., & Giordano, S. (2014). Mobile ad hoc networking: milestones, challenges, and new research directions. Communications Magazine, IEEE,52(1), 85-96.

9. Douligeris, C., & Mitrokotsa, A. (2004). DDoS attacks and defense mechanisms: classification and state-of-the-art. Computer Networks, 44(5), 643-666.

10. RioRey Taxonomy of DDoS Attacks, Retrieved from http://www.riorey.com/x-resources/2012/RioRey_Taxonomy_DDoS_Attacks_2012.eps

11. Geng, X., & Whinston, A. B. (2000). Defeating distributed denial of service attacks. IT Professional, 2(4), 36-42.

12. Peng, T., Leckie, C., & Ramamohanarao, K. (2007). Survey of network-based defense mechanisms countering the DoS and DDoS problems. ACM Computing Surveys (CSUR), 39(1), 3.

13. Wang, H., Zhang, D., & Shin, K. G. (2002, June). Detecting SYN flooding attacks. In INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE (Vol. 3, pp. 1530-1539). IEEE.

14. Security and Stability Advisory Committee. (2006). DNS distributed denial of service (DDoS) attacks. Advisory, ICANN, March.

15. Akbar, M. A., & Farooq, M. (2014). Securing SIP-based VoIP infrastructure against flooding attacks and Spam Over IP Telephony. Knowledge and Information Systems, 38(2), 491-510.

16. Slowloris HTTP DOS, Retrieved from http://ha.ckers.org/slowloris

17. Researchers to Demonstrate New Attack That Exploits HTTP, Retrieved from http://www.darkreading.com/vulnerability-management/167901026/security/attacks-breaches/228000532/index.html

18. Abliz, M. E. H. M. U. D. (2011). Internet denial of service attacks and defense mechanisms. University of Pittsburgh, Tech. Rep. TR-11-178.

19. Ranjan, S., Swaminathan, R., Uysal, M., & Knightly, E. W. (2006, April). DDoS-Resilient Scheduling to Counter Application Layer Attacks Under Imperfect Detection. In INFOCOM.

20. Mirkovic, J., Prier, G., & Reiher, P. (2003, April). Source-end DDoS defense. InNetwork Computing and Applications, 2003. NCA 2003. Second IEEE International Symposium on (pp. 171-178). IEEE.

21. Chen, R., Park, J. M., & Marchany, R. (2006, November). NISp1-05: RIM: Router Interface Marking for IP Traceback. In Global Telecommunications Conference, 2006. GLOBECOM'06. IEEE (pp. 1-5). IEEE.

22. IFerguson, P. (2000). Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing.

23. Huici, F., & Handley, M. (2007). An edge-to-edge filtering architecture against DoS. ACM SIGCOMM Computer Communication Review, 37(2), 39-50.

24. Burch, H., & Cheswick, B. (2000, December). Tracing Anonymous Packets to Their Approximate Source. In LISA (pp. 319-327).

25. Mirkovic, J., Prier, G., & Reiher, P. (2002, November). Attacking DDoS at the source. In Network Protocols, 2002. Proceedings. 10th IEEE International Conference on (pp. 312-321). IEEE.

26. Gil, T. M., & Poletto, M. (2001, August). MULTOPS: a data-structure for bandwidth attack detection. In USENIX Security Symposium.

27. Abdelsayed, S., Glimsholt, D., Leckie, C., Ryan, S., & Shami, S. (2003, December). An efficient filter for denial-of-service bandwidth attacks. In Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE (Vol. 3, pp. 1353-1357). IEEE.

28. Zargar, S., Joshi, J., & Tipper, D. (2013). A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks.

29. Argyraki, K., & Cheriton, D. R. (2009). Scalable network-layer defense against internet bandwidth-flooding attacks. IEEE/ACM Transactions on Networking (TON), 17(4), 1284-1297.

30. Walfish, M., Vutukuru, M., Balakrishnan, H., Karger, D., & Shenker, S. (2006, September). DDoS defense by offense. In ACM SIGCOMM Computer Communication Review (Vol. 36, No. 4, pp. 303-314). ACM.

31. Srivatsa, M., Iyengar, A., Yin, J., & Liu, L. (2008). Mitigating application-level denial of service attacks on Web servers: A client-transparent approach. ACM Transactions on the Web (TWEB), 2(3), 15.

32. Oikonomou, G., & Mirkovic, J. (2009, June). Modeling human behavior for defense against flash-crowd attacks. In Communications, 2009. ICC'09. IEEE International Conference on (pp. 1-6). IEEE.

33. Xie, Y., & Yu, S. Z. (2009). A large-scale hidden semi-Markov model for anomaly detection on user browsing behaviors. Networking, IEEE/ACM Transactions on, 17(1), 54-65.

34. Devi, S. R., & Yogesh, P. (2012). A hybrid approach to counter application layer DDoS attacks. International Journal on Cryptography and Information Security (IJCIS), 2(2).

35. Wu, Y. C., Tseng, H. R., Yang, W., & Jan, R. H. (2011). DDoS detection and traceback with decision tree and grey relational analysis. International Journal of Ad Hoc and Ubiquitous Computing, 7(2), 121-136.

36. John, A., & Sivakumar, T. (2009). Ddos: Survey of traceback methods.International Journal of Recent Trends in Engineering, 1(2), 241-245.

37. Cabrera, J. B., Lewis, L., Qin, X., Lee, W., Prasanth, R. K., Ravichandran, B., & Mehra, R. K. (2001). Proactive detection of distributed denial of service attacks using mib traffic variables-a feasibility study. In Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on (pp. 609-622). IEEE.

38. Yaar, A., Perrig, A., & Song, D. (2003, May). Pi: A path identification mechanism to defend against DDoS attacks. In Security and Privacy, 2003. Proceedings. 2003 Symposium on (pp. 93-107). IEEE.

39. Peng, T., Leckie, C., & Ramamohanarao, K. (2003, May). Protection from distributed denial of service attacks using history-based IP filtering. InCommunications, 2003. ICC'03. IEEE International Conference on (Vol. 1, pp. 482-486). IEEE.

40. Yu, J., Fang, C., Lu, L., & Li, Z. (2009). A lightweight mechanism to mitigate application layer DDoS attacks. In Scalable Information Systems (pp. 175-191). Springer Berlin Heidelberg.

41. Kambourakis, G., Moschos, T., Geneiatakis, D., & Gritzalis, S. (2008). Detecting DNS amplification attacks. In Critical Information Infrastructures Security (pp. 185-196). Springer Berlin Heidelberg.

42. Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A., & Knightly, E. (2009). DDoS-shield: DDoS-resilient scheduling to counter application layer attacks.IEEE/ACM Transactions on Networking (TON), 17(1), 26-39.

43. Yuanyuan Zeng , Kang G. Shin , Xin Hu, Design of SMS commanded-and-controlled and P2P-structured mobile
botnets, Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks, April
16-18, 2012, Tucson, Arizona, USA

44. Hund, R., Hamann, M., & Holz, T. (2008, December). Towards next-generation botnets. In Computer Network Defense, 2008. EC2ND 2008. European Conference on (pp. 33-40). IEEE.

45. Traynor, P., Lin, M., Ongtang, M., Rao, V., Jaeger, T., McDaniel, P., & La Porta, T. (2009, November). On cellular botnets: measuring the impact of malicious devices on a cellular network core. In Proceedings of the 16th ACM conference on Computer and communications security (pp. 223-234). ACM.

46. Jing, L., Yang, X., Kaveh, G., Hongmei, D., & Jingyuan, Z. (2009). Botnet: classification, attacks, detection, tracing, and preventive measures. EURASIP journal on wireless communications and networking, 2009.

47. Vogt, R., Aycock, J., & Jacobson Jr, M. J. (2007, February). Army of Botnets. In NDSS.

48. Flo, A.R., Josang, A: Consequences of Botnets Spreading to Mobile Devices. In: 14th IT Systems, Oslo(2009)

49. Cooke, E., Jahanian, F., & McPherson, D. (2005, July). The zombie roundup: Understanding, detecting, and disrupting botnets. In Proceedings of the USENIX SRUTI Workshop (Vol. 39, p. 44).

50. Starnberger, G., Kruegel, C., & Kirda, E. (2008, September). Overbot: a botnet protocol based on Kademlia. In Proceedings of the 4th international conference on Security and privacy in communication netowrks (p. 13). ACM.

51. Maymounkov, P., & Mazieres, D. (2002). Kademlia: A peer-to-peer information system based on the xor metric. In Peer-to-Peer Systems (pp. 53-65). Springer Berlin Heidelberg.

52. G. Geng, G. Xu, M. Zhang, Y. Yang, and G. Yang. An improved sms based heterogeneous mobile botnet model.
In Information and Automation (ICIA), 2011 IEEE International Conference on, pages 198--202. IEEE, 2011.

53. Pieterse, Heloise, and Martin Olivier. "Design of a hybrid command and control mobile botnet." Academic
Conferences and Publishing International Ltd, 2013.

54. Wang, P., Sparks, S., & Zou, C. C. (2010). An advanced hybrid peer-to-peer botnet. Dependable and Secure Computing, IEEE Transactions on, 7(2), 113-127.

55. Matthew Knysz, Xin Hu, Yuanyuan Zeng, Kang G. Shin, Open WiFi Networks: Lethal Weapons for Botnets?, The
31st Annual IEEE International Conference on Computer Communications: Mini Conference.

56. Zhao, Shuang, et al. "Cloud-based push-styled mobile botnets: a case study of exploiting the cloud to device
messaging service." Proceedings of the 28th Annual Computer Security Applications Conference. ACM, 2012.

57. Shuai, Wang, et al. "S-URL Flux: A Novel C&C Protocol for Mobile Botnets."Trustworthy Computing and Services.
Springer Berlin Heidelberg, 2013. 412-419.

58. Meisam Eslahi, Rosli Salleh, Nor Badrul Anuar, MoBots: A New Generation of Botnets on Mobile Devices and
Networks, 2012 International Symposium on Computer Applications and Industrial Electronics (ISCAIE 2012),
December 3-4, 2012, Kota Kinabalu Malaysia.

59. Khaled M Elleithy, Drazen Blagovic, Wang Cheng, Paul Sideleau, Denial of Service Attack Techniques: Analysis,
Implementation and Comparison. Systemics, Cybernetics and Informatics, Volume 3- Number 1.

60. Kartaltepe, E. J., Morales, J. A., Xu, S., & Sandhu, R. (2010, January). Social network-based botnet command-and-control: emerging threats and countermeasures. In Applied Cryptography and Network Security (pp. 511-528). Springer Berlin Heidelberg.

61. G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic.
In Proc. of NDSS, 2008.

62. Guofei Gu , Roberto Perdisci , Junjie Zhang , Wenke Lee, BotMiner: clustering analysis of network traffic for
protocol- and structure-independent botnet detection, Proceedings of the 17th conference on Security
symposium, p.139-154, July 28-August 01, 2008, San Jose, CA

63. Cui Xiang, Fang Binxing, Yin Lihua, Liu Xiaoyi, Zang Tianning, Andbot: Towards advanced mobile botnets,
Proceedings of the 4thMA

64. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013–2018 - Cisco. (n.d.). Retrieved from http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html

65. Ristic, I. (2010). ModSecurity Handbook. Feisty Duck.

66. Retrieved from
http://www.owasp.org/index.php/Category:OWASP_ModSecurity_Core_Rule_Set_Project

67. Aiello, M., Papaleo, G., & Cambiaso, E. (2014, January). SlowReq: A Weapon for Cyberwarfare Operations. Characteristics, Limits, Performance, Remediations. In International Joint Conference SOCO'13-CISIS'13-ICEUTE'13(pp. 537-546). Springer International Publishing.

68. How to Mitigate Slowloris Attack. (n.d.). Retrieved March 3, 2014, from docs.cpanel.net/twiki/bin/view/EasyApache/Apache/SlowlorisAttacks

69. Mobile IPv6 Tutorial. (n.d.). Retrieved March 3, 2014, from http://www.usipv6.com/ppt/MobileIPv6_tutorial_SanDiegok.pdf

70. Verizon Mandates IPv6 Support for Next-Gen Cell Phones. (n.d.). Retrieved from http://www.circleid.com/posts/20090609_verizon_mandates_ipv6_support_for_next_gen_cell_phones/

71. Wallen, J. (2014, Feb 21). The ubuntu phone is official: Let the madness begin!. Retrieved from http://www.techrepublic.com/article/the-ubuntu-phone-is-official-let-the-madness-begin/