Measurements and Analysis of a Denial of Service Attack by Resource Flood

Alba Moreno Hernández
National Polytechnic Institute.
Mexico City
alba_mhdez@hotmail.com

Eleazar Aguirre Anaya
National Polytechnic Institute
Mexico City
eaa000@gmail.com

Ruben Jonathan García Vargas
National Polytechnic Institute
Mexico City
ligthg@gmail.com

Ramon Galeana Huerta
National Polytechnic Institute
Mexico City
calewiro@gmail.com

***Abstract*—Nowadays, a lot of web servers are object of great threats and vulnerabilities [7] that compromise their security. One of the possible threats that Web servers face is a Denial of service (DoS). This attack endangers the availability of a service, and is effective when there are no protection mechanisms that allow its mitigation. This paper presents measurements made to a denial of service attack that exhausts the capacity to release resources from an Apache Web server with the Linux operating system. For this, a test network with common security controls was used. In this scenario two mechanisms for mitigating the effects of DoS attack were implemented and measured in order to reduce recovery time of service.**

Key words—DoS-Attacks, Hardening, Network, Security, TCP/IP.

I. INTRODUCTION

The Web servers are designed to answer multiple simultaneous client requests efficiently in a minimum time. Also, the Web servers have a configuration in their hosting sites to indicate the maximum simultaneous connections to accept. When the limit is reached, the Web server does not respond to the client requests and denies the service to the website users.

The limit of connections is set due to the possible slow down of the web server when a great amount of users are using it simultaneously.

When a web server's client suddenly stops to answer to the server's packets, the web server waits until a set time has elapsed and then retransmits the packets to the client. After that, the server waits for the client´s response the double of time it did before. If the web server does not get any answer from the client, it keeps the retransmission–wait process until a set time has elapsed. The set time is called connection timeout and all the connections that reach this state are closed immediately.

Some types of Denial of Service Attacks (DoS) and Distributed Denial of Service Attacks (DDoS) are based on the timeout configuration to keep a server allocating resources for connections that will not answer. This behavior also avoids that new clients can connect to a service. A resource flood attack to a web server is produced after the attacker has established a connection (three way handshake) satisfactorily and tries to collapse an application (Website, Web services, etc.) by consuming the capability to release resources.

An attacker connection keeps a resource busy until it reaches the set time to close a no responding connection and sends a timeout error before releasing the resource. After synchronization process, the attacker sends a HTTP request to the web server and stops answering to such request. The web server is forced to wait for the timeout to expiry.

While the web server waits the response of the attacker, it receives new connection requests, decreasing the number of available connections to use. The attack is accomplished when the number of connections waiting response reaches the maximum connections allowed to the service, causing that the web server denies new connections until a resource is released by the timeout error. These attacks are difficult to detect because they do not require wide bandwidth, but they need a high number of connection requests, which are difficult to identify between attackers requests and legitimate requests.

This type of attacks affects all web servers without exception. Actually, the most distributed web server is Apache, which until January 2012 covered 64% of the market [8]. Studies show that the Apache Web server and the Linux operating system are the most attacked until 2010 [9].

The present work shows the behavior of resource flood attacks and presents configurations in Apache web server and Linux kernel that allow the faster recovery from the attack's effects. This paper is organized as follows. Section II present some proposal and show how to join two ways to help the web server to minimize the effects of the attack. Section III shows a test schema to measure the attack. In section IV the effects of the attack are shown. Section V presents an implementation of the protection mechanisms. In Section VI the results of the implementation are presented and finally in Section VII the conclusions of this work are shown.

II. PROTECTION MECHANISMS AGAINST RESOURCES FLOOD DENIAL OF SERVICE ATTACK

Existing proposals to detect the resource flood attacks use a Hidden Semi-Markov Model (HSMM) [10] which requires a large number of legitimate request sequences to train the HSMM. The disadvantage of this method is that if during the

training period an attack is executed it would avoid its later detection.

Another proposal analyzes browsing order of pages or correlation between browsing time and page information size [11]. The disadvantage in this kind of methods consists in the high number false negative and false positive.

There are other proposals to prevent the DoS Resource Flood attacks, which modify the TCP/IP protocol stack [3]. Another suggests the adjustment of the web server configuration parameters [1] [12]. Both proposals aim to harden the web server's behavior and eliminate connections that do not answer fast. However, the implementation of these methods must be considered carefully due the fact that the heterogenic network systems can contain slow packet transmission sections or clients that could be disconnected by these aggressive configurations.

Modify the TCP/IP protocol stack and adjust the Apache configuration parameters are the most practical manners to reduce the recovery time of the web server after a resource flood DoS attack. This paper joins these two proposals to mitigate the effects of the attack. This paper also measures the attack´s behavior before and after the implementation of mitigation methods and allows characterize it. The main goal of this section is to describe the parameters to modify in the kernel files for TCP/IP protocol stack and Apache web server's configuration files. In section V the implementation of these parameters in Linux and apache are shown.

A. TCP/IP protocol stack

Modifying the behavior of the TCP/IP protocol stack is possible to get a reliable operating system against this attack. This is achieved by configuring the next behavior parameters [4]:

1. Disable IP Source Routing. Avoid the IP address spoofing.
2. Enforce sanity checking, also called ingress filtering or egress filtering. The point is to drop a packet if the source and destination IP addresses in the IP header do not make sense when considered in light of the physical interface on which it arrived.
3. Log and drop "Martian" packets. A "Martian" packet is one for which the host does not have a route back to the source IP address.
4. Increase resilience under heavy TCP load. There are five major steps to making a system more resilient under heavy, possibly malicious, TCP load:

- Use TCP SYN Cookies. With TCP SYN Cookies, the kernel does not really allocate the TCP buffers unless the server's ACK/SYN packet gets an ACK back, meaning that it was a legitimate request.
- Reduce the allowed number of HALF_OPEN TCP circuits. Further requests are refused, a Denial of Service, but at least the server has not run out of memory.
- Reduce the amount of time an opening TCP circuit can stay in the HALF_OPEN state. The server is made less patient if the TCP circuit is not fully established quickly, it is dropped and the client, if legitimate but very slow, must start again.
- Reduce the amount of time a closing TCP circuit can stay in the TIME_WAIT state.

B. Web server parameters

The general parameters considered in the web server configuration to avoid the DoS resource flood attack are listed below [5] [12]:

Timeout: Defines the time in seconds that the web server will wait to receive and send a request during the communication. Specifically, defines how long the server waits to receive GET request, how long to wait for a TCP packet in a POST or PUT request, and how long waits between an ACK and other. Usually the default value is 300 seconds [2].

KeepAlive: Enables HTTP persistent connections. By default this value is set ON.

MaxKeepAliveRequest: Establishes the maximum number of requests to allow during a persistent connection.. This value is 100 by default [2].

MaxKeepAliveTimeout: Amount of time the server will wait for subsequence requests on a persistent connection.

Startserver: Allows establishing how many processes will be created at start-up.

MinSpareServers and MaxSpareServers: The web server can adapt dynamically to receive load keeping an appropriated number of instances free based on the traffic. The web server checks the number of instances waiting for requests and removes some if their number is bigger than MaxSpareServers or creates new ones if the number is lower than MinSpareserver.

MaxClients: Defines the maximum number of threads that the web server will create to attend requests.

MaxRequestPerChild: Indicates the number of requests that each thread attends before recycling.

The TCP/IP configuration is located in a configuration file called sysctl.conf in the directory /etc/sysctl.conf.

III. TEST SCENARIO

This section provides a typical network topology, with security controls, in which the effect of the DoS resource flood attack in a web server was measured. The use of network devices like switch, firewall and router was proposed for the test scenario to give a common security implementation.

The construction of the test scenario is shown in figure 1. The firewall gets the requests from Internet and sends them to the web server inside a DMZ. This topology was used due to the simplicity of the required method to measure the attack.

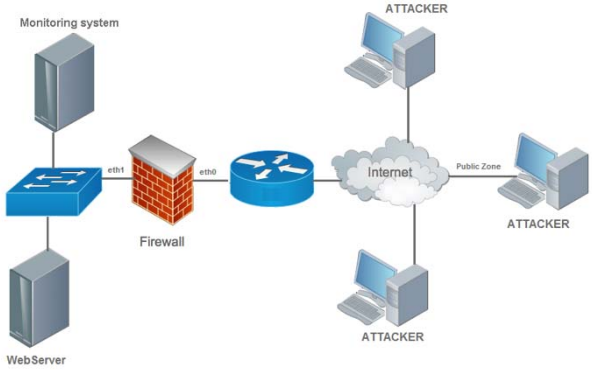

Fig. 1. Topology of test network

The figure 1 shows two zones; one public and one demilitarized. The web server and monitoring system are inside the demilitarized zone.

The web server, the monitoring system and the firewall are located in the DMZ connected by switch. The monitoring system captures web server traffic through a switch port in mirror mode. The firewall separates both zones, filters the packets to the web server and only accepts the ones that are sent to port 80. The router connects the firewall to Internet where three attackers are situated.

TABLE I. FIREWALL POLICIES

SourceZone	Source Port	Destination Zone	Destination Port	Protocol	Action
Public Zone	Any	Private Zone	80	TCP	accept
Any	Any	Any	Any	ICMP	deny
Any	Any	Any	Any	TCP	deny
Any	Any	Any	Any	UDP	deny

The firewall used IPTABLES. The firewall policies for the interconnection of the two zones are shown in Table 1. The default configuration for the policy of the firewall is ACCEPT.

TABLE II. .DEVICES CHARACTERISTICS

Device	S.O	Processor	RAM	NIC
Firewall	Ubuntu 11.04	Intel Core 2 Duo (3 GHz)	3 GB	eth0 (Realtek 10/100 Mbps PCI) eth1 (Realtek 10/100 Mbps PCI)
Web server	Ubuntu 11.04	Intel Core 2 Duo (2.2 GHz)	1 GB	eth0 (Realtek 10/100 Mbps PCI)
Monitoring System	Ubuntu 11.04	Intel Core 2 Duo (2.2 GHz)	1 GB	eth0 (Realtek 10/100 Mbps PCI)
Attackers	Backtrack 4	AMD Athlon X2 Dual–Core	3 GB	eth0 (Realtek 10/100 Mbps PCI)

The hardware characteristics of the systems used for this topology are listed in the table 2

The firewall has two NICs installed with the next characteristics

- eth0: Connected to public zone with network address 177.169.0.1/30 and 100 Mbps.
- eth1: Connected to Demilitarized zone with network address 201.15.7.1/24 and 100 Mbps.

The web server has the network address 177.169.0.2 and had installed Apache 2.2

The network devices used are:

- Switch Catalyst 2960 in the Demilitarized Zone
- Router Cisco 3800 Series for Public Zone

The attack was executed from the Public Zone to the Demilitarized Zone using the public address of the Firewall as shown in table 2. The tool used for the attack was slowhttptest.

In the next section are shown the results and consequences of the attack in the web server.

IV. ATTACK DESCRIPTION AND EFFECTS

The resource flood attack was executed from the public zone to the demilitarized zone, taking as a reference the characteristics in table 2.

The attackers were located in the public zone. The attack was executed by sending 1000 connection requests followed by GET requests to the Web service. The time elapsed in the attack depended on the number of attackers.

The execution of the attack was done with the use of the tool slowhttptest [6], installed in Backtrack 4, that allows testing the effect of a possible attack to a web service.

The tool constantly performs synchronization requests to check for available connections. If after 3 seconds, the server does not answer, the tool considers that the service is busy and repeats the process until the number of connections specified is reached or 240 seconds have elapsed.

The command used for the attack is shown below:

```
slowhttptest -c 1000 -H -g -o httptest1 -i 10 -r 200 -t GET -u
http://177.169.0.1 l -x 24 -p 3
```

A first test was executed with only one adversary to measure the effects of the attack in the web server and thus characterize it. The attack started when the tool sent multiple connection requests and started to download the web site. The tool leaved the server waiting for an instruction and kept busy the resource for the connection. Meanwhile the server was waiting, the tool continued to open new connections. In this case the tool tried opening 200 connections per second, and when the web server did not accept new connections, waited until a connection was released and took control over the resource again.

The attack finished when the tool had opened and closed the indicated number of connections, in this case 1000. The tool was configured to send 1000 connections due to the limitations in the attacking machine. The default configuration of Apache web server accepts 150 clients spread over up 10 threads. One attacker requested almost 7 times the server capacity, keeping the server busy to other clients

Before and while the attack's execution, the CPU usage and network traffic of the web server were tracked. Figure 2 shows that before the attack has started there was no abnormal activity in the server.

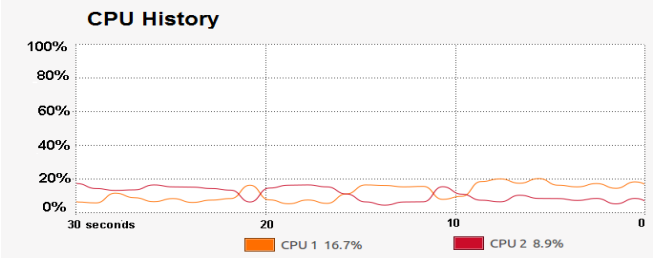

Fig. 2. Initial CPU usage

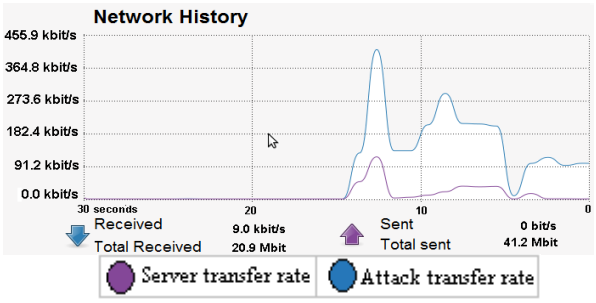

Fig. 3. Initial Network usage

Figure 3 was taken 15 seconds after the attack had started and shows that the attack began with a transfer rate of 410.35 Kbit/s. Before the attack started, the used RAM percentage was 33.75. When the attack began, the RAM usage increased in 1.85 percent.

While the attack was executed, the network traffic varies over time and increases CPU consumption as shown in Figure 4. This happened because the tool consumed all available connections in a short period of time. When the web server closed a connection, the tool opened a new one to try to consume the available resource again, shown in Figure 5.

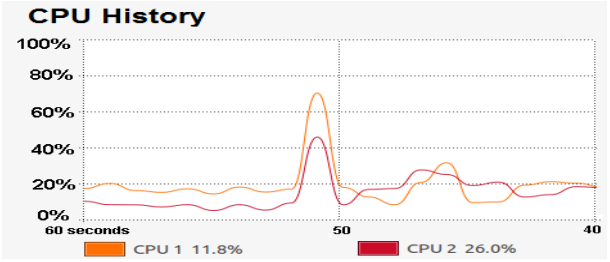

Fig. 4. Intermediate CPU usage

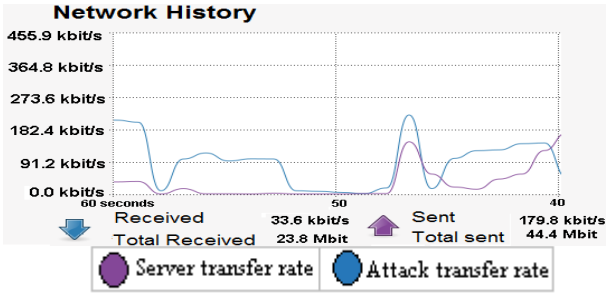

Fig. 5. Intermediate stage of the attack

15 to 20 seconds were needed to close a no responding connection created by the tool. Figure 6 shows the flow of a connection until the server closed it.

No.	Time	Source	Destination	Protocol	Info
679	0.714656	201.15.7.2	177.169.0.2	TCP	44255 > http [SYN] Seq=0 Win=5840 Len=0 MSS
680	0.714681	177.169.0.2	201.15.7.2	TCP	http > 44255 [SYN, ACK] Seq=0 Ack=1 Win=144
682	0.715518	201.15.7.2	177.169.0.2	TCP	44255 > http [ACK] Seq=1 Ack=1 Win=5888 Len
686	0.720781	201.15.7.2	177.169.0.2	HTTP	Continuation or non-HTTP traffic
688	0.720817	177.169.0.2	201.15.7.2	TCP	http > 44255 [ACK] Seq=1 Ack=392 Win=15552
3333	10.033645	201.15.7.2	177.169.0.2	HTTP	Continuation or non-HTTP traffic
3538	10.036668	177.169.0.2	201.15.7.2	TCP	http > 44255 [ACK] Seq=1 Ack=407 Win=15552
5119	17.612772	177.169.0.2	201.15.7.2	HTTP	HTTP/1.1 400 Bad Request (text/html)
5120	17.612820	177.169.0.2	201.15.7.2	TCP	http > 44255 [FIN, ACK] Seq=509 Ack=407 Win
5126	17.612991	201.15.7.2	177.169.0.2	TCP	44255 > http [ACK] Seq=407 Ack=509 Win=6912
5129	17.613060	201.15.7.2	177.169.0.2	TCP	44255 > http [FIN, ACK] Seq=407 Ack=510 Win
5131	17.613106	177.169.0.2	201.15.7.2	TCP	http > 44255 [ACK] Seq=510 Ack=408 Win=1555

Fig. 6. Connection stream

The behavior of open-close connections continued until the tool connected and disconnected 1000 times. In this case, the tool took 190 seconds to open and close 1000 connections. Once the attack has finished, the server returns to its previous state as shown in Figure 7 and Figure 8.

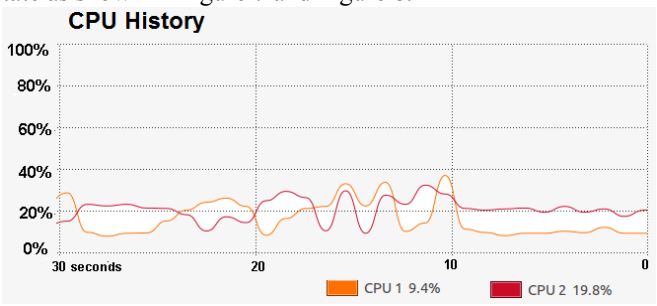

Fig. 7. Final CPU usage

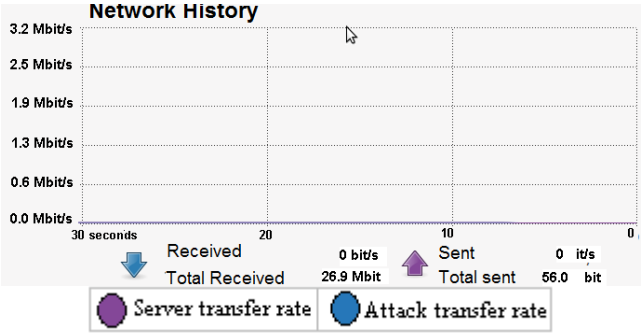

Fig. 8. Final stage of the attack

While the attack was executed, two clients tried to connect to the web server unsuccessfully due to the available connections competition created by the attack. The clients could connect to the server 15 seconds after the attack. A second attack was executed with three adversaries in the public zone, taking as a reference the characteristics in table 2. The tool was executed with the same parameters of the first test. The memory and bandwidth consumption stayed the same as first test. This behavior occurs because the tool does not send more connections when it detects that the web server does not answer to new requests. In this test, the web server remained more time unavailable because the number of attacking connections increased three times.

The second attack showed that behavior in the web server stayed the same regardless the number of adversaries.

Next section presents how to prevent and reduce the effects of this attack. The proposal is to implement aggressive configurations to the TCP/IP protocol and web server behavior.

V. Implementation Of Mechanisms Against Dos Resource Flood Attacks

Due to attacks like the presented before, it is necessary to implement measures to allow mitigating their effects. Below is presented the implementation of mechanisms of section II in TCP/IP protocol stack over a Linux operating system and the use of Apache 2 as the web server.

C. Apache Web server

Following the mechanism presented in section II, the modifications must be done in the apache configuration file, which is located in /etc/apache2 and it is called apache.conf. (For Ubuntu 11.04)

The configured parameters are listed below:

```
Timeout 3
KeepAlive           On
MaxKeepAliveRequests    100
KeepAliveTimeout        3
StartServers        3
MinSpareServers 3
MaxSpareServers 4
MaxClients          150
MaxRequestsPerChild     100
```

With previous configuration, the web server closes unanswered connections faster, allowing releasing resources in a small period of time.

D. TCP/IP Protocol Stack

The TCP/IP protocol stack configuration is located in a file called sysctl.conf. The configured parameters are:

```
# The following is suitable for dedicated web server, mail, ftp server etc.
# BOOLEAN Values:
# a)0(zero) - disabled / no / false
# b) Non zero - enabled / yes / true
# ------------------------------------
# Controls IP packet forwarding
net.ipv4.ip_forward TCP/IP protocol stack TCP/IP protocol stack 0
# Accept packets with SRR option? No
net.ipv4.conf.all.accept_source_route = 0
net.ipv4.conf.all.forwarding = 0
net.ipv4.conf.all.mc_forwarding = 0
# Enable source validation by reversed path, as specified in RFC1812
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1
# Controls source route verification
net.ipv4.conf.default.rp_filter = 1
# Do not accept source routing
net.ipv4.conf.default.accept_source_route = 0
# Log packets with impossible addresses to kernel log?yes
net.ipv4.conf.all.log_martians = 1
net.ipv4.conf.default.accept_source_route = 0
# Prevent against the common 'syn flood attack'
net.ipv4.tcp_max_syn_blacklog= 1280

net.ipv4.tcp_syncookies = 1
```

With this configuration, the TCP/IP protocol stack of the web server is hardened, and many kinds of network attacks are avoided because the packets are ignored or filtered before being uploaded to the applications [3], these results are shown in the next section.

VI. Implementation Results

After the configurations implemented to the TCP/IP protocol stack and the Apache web server, an attack test was executed to measure the new behavior in the web server. The executed attack had identical parameters to the first measured. However, the tool took only 29 seconds to open and close 1000 connections. The attack´s full CPU flow is shown in the Figure 9 and Figure 10 shows the network usage rate.

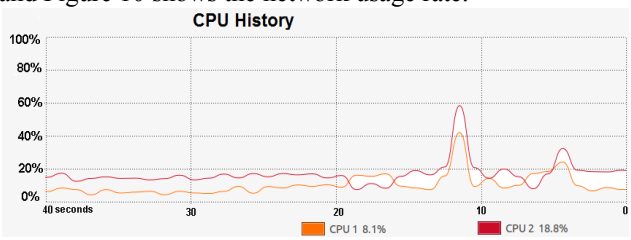

Fig. 9. CPU behavior until 35 seconds after the attack started.

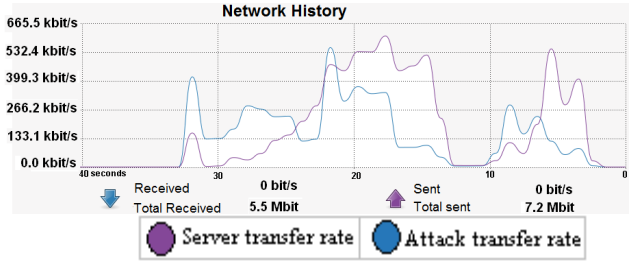

Fig. 10. Network behavior until 35 seconds after the attack started.

Once the attack has finished opening and closing 1000 connections, the server returned to the previous state, before the attack, as shown in Figure 11 and Figure 12.

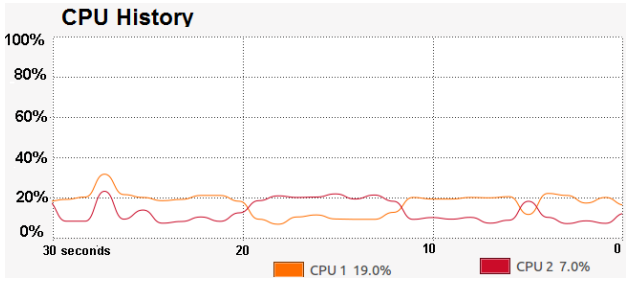

Fig. 11. CPU behavior after the attack.

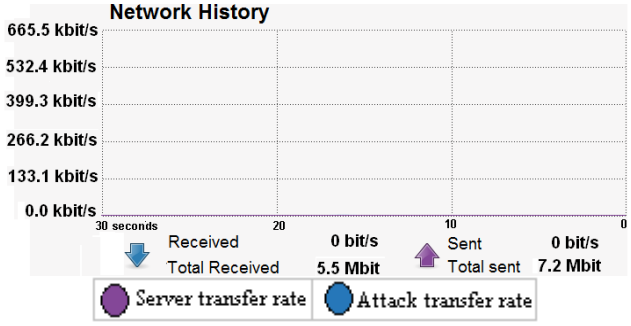

Fig. 12. Resources behavior after the attack.

During the attack, the clients did not notice any change in the web server.

VII. Conclusions

The denial of service resource flood attacks are easy to execute due to the simplicity of existing tools, and reach their objective easily in scenarios that lack of security controls and mechanisms endangering the availability of the services. When security configurations for TCP/IP protocol stack and web server are implemented, the web server tolerance for this attacks increases significantly, in this case the test showed that the web server closed 1000 attacking connections 6.5 times faster after the configuration was done.

The use of these methods must be carefully planned due to the fact that the clients that do not respond fast enough could have their connections closed. Therefore an analysis of the exact values to configure the service is needed; this includes contemplating what kind of users does the service have, and where they are located. This work enabled to characterize the attack and in future work present new security controls to detect and mitigate this attack.

References

[1] A.Nadalin, C. Kaler, R. Monzillo., "Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)", 2006.

[2] Apache Group, "TimeoutDirective", Apache.org http://httpd.apache.org/docs/2.2/mod/core.html#timeout

[3] TCP protocol - Linux man page, http://linux.die.net/man/7/tcp

[4] Cromwell B, "TCP/IP Stack Hardening", linux-sec, April 2004

[5] M. Jose, "Apache Tunning", March 2012.

[6] Shekyan Sergey, "Application Layer DoS attack simulator", 2012, http://code.google.com/p/slowhttptest/

[7] S. Suriadi, A. Clark, and D. Schmidt, "Validating denial of service vulnerabilities in web services", in Network and System Security, International Conference on Network and System Security. IEEE Computer Society, 2010

[8] Netcraf LTD, "January 2012 Web server Survey", http://news.netcraft.com/archives/2012/01/03/january-2012-web-server-survey.html, 2012

[9] Almeida Marcelo, "Defacements Statistics 2008 - 2009 - 2010", http://www.zone-h.org/news/id/4735

[10] Lu Wei-Zhou, "An HTTP Flooding Detection Method Based on Browser Behavior", IEEE

[11] Yatagai Takeshi, "Detection of HTTP-GET flood Attack Based on Analysis of Page Access Behavior", IEEE 2007

Apache Group, "Apache HTTP server - Security tips", Apache.org, http://httpd.apache.org/docs/trunk/misc/security tips.html.