ISSN:0975-9646

Modular System for Mitigating Flood Attacks

Devisha Srivastava

***Abstract*— Denial-of-Service (DoS) flooding attacks have become a serious threat to the reliability of the Internet. Web servers face all kinds of users; some of them engage malicious activities to degrade or completely block network services, such as flooding attacks. As a result, lots of resource and bandwidth on web sites might be wasted. While many approaches exist to filter network-level attacks, the application level attacks are harder to detect at the firewall. Filtering at this level can be computationally expensive and difficult to scale, while still producing false positives that block legitimate users.**

This paper presents a web application level approach to mitigate DoS attacks. The proposed method is to build a security gateway module with reverse proxy support that provides attack surface reduction enhancements against the HTTP Flood Attacks at the web application level. Web-based anomaly detection with reverse HTTP proxy which intercepts traffic and protects web application by providing users with a CAPTCHA to verify legitimate requests is used.

***Keywords*— DoS, HTTP Flood Attack, CAPTCHA, False Positives**

I. INTRODUCTION

Denial of Service is an attack which makes an information or data unavailable to its intended hosts. In a Denial-of-Service (DoS) flooding attack, attackers aim to disrupt and prevent legitimate communications by congesting links in a network. The first series of high-profile and well-publicized DoS flooding attacks occurred in 2000, in which attackers disrupted the services of various popular websites including Yahoo, eBay and CNN [4]. Today, DoS flooding attacks have become a serious threat to the reliability of the Internet. In order to accomplish a DoS state on systems, flood attacks aim to push limits of system usage to the out of boundaries determined by the normal usage scenarios. There may be a flood attack between the considered normal network traffic and the considered abnormal network traffic.

The main reason of flood attacks is the vulnerability in the protocol. Before starting a discussion on the web application level approach to the HTTP flood attacks, it is important to clarify whether the attack is a HTTP flood attack or not. To consider an attack attempt as a HTTP flood attack, a TCP packet which carries a HTTP request payload should be interpreted by the web service. Attack surface for HTTP flood attacks always begins with the web service and its backend infrastructure. A HTTP flood attack attempt, which cannot make it to the web service, is just a TCP DoS attack that saturates the network traffic.

This paper presents an approach to mitigate the impact of false positives, describes a prototype implementation of the system, and provides a discussion of its applicability to real-world applications. The objective is to build a security gateway module with reverse proxy support that provides attack surface reduction enhancements against the HTTP flood attacks at the web application level. The approach proposed composes a web-based anomaly detection system with a reverse HTTP proxy which intercepts traffic and protects web application by allowing only legitimate requests. It is a good security practice to implement additional precautions to every mitigation level including the web application level and this proxy server proves as an additional layer of defense by protecting the web server.

Web-based applications represent a serious security exposure. These applications are directly accessible through firewalls by design, and, in addition, they are often developed in a hurry by programmers who focus more on functionality and appearance than security. As a result, web-based applications have recently become the primary target of attempts to compromise networks [1]. Reverse proxies can hide the existence and characteristics of an origin server or servers. Application firewall features can protect against common web-based attacks. Without a reverse proxy, removing malware or initiating takedowns, for example, can become difficult. In the case of secure websites, a web server may not perform SSL encryption itself, but instead offloads the task to a reverse proxy that may be equipped with SSL acceleration hardware. A reverse proxy can distribute the load from incoming requests to several servers, with each server serving its own application area. In the case of using reverse proxy in the neighborhood of web servers, the reverse proxy may have to rewrite the URL in each incoming request in order to match the relevant internal location of the requested resource.

The situation of false positives is ameliorated by the approach of anomaly detection systems that characterize the “normal" use of a web-based application. These systems are able to block web requests that do not fit the established parameters of “normality" [2,3]. Unfortunately, anomaly detection systems are prone to false positives due to over-simplified modeling techniques or insufficient training. Therefore, if the anomaly score of a web request is used as the basis to deny access to a web site, a false positive may cause the denial of a legitimate request. This situation is the result of an “all-or-nothing" approach, in which all requests that are identified as anomalous are automatically considered malicious.

To mitigate the problem of false positives generated by anomalous but benign requests, a novel solution based on anomaly-based reverse proxy is proposed.

The proposed security module incorporates reverse proxy support. It implements security in web applications by meeting three important goals:

(1) Accurate attack detection,

(2) Effective response (dropping or rerouting) to reduce the flood, and

(3) Precise identification of legitimate traffic and its safe delivery to the victim.

The modular system detects IP addresses of the abnormally excessive requests according to a previously defined rule, reduces attack surface, return these requests with a low resource used response, block detected IP addresses by using other components at the other mitigation levels. It features reCAPTCHA support, slow down or restrict access for automated tools (HTTP flood, brute force tools, etc.), save your server & backend infrastructure resources under an attack, implicit deny of application access for DoS/DDoS attacks etc.

II. Background and Related Work

A. *What is a Denial of Service?*

Denial of Service attack is performed solely with the intention to deny the legitimate users to access services. Since DoS attack is usually performed by means of bots, automated software. These bots send a large number of fake requests to the server which exceeds server buffer capacity which results in DoS attack [5].

The defense against DoS flooding attacks is significantly complicated by the fact that the Internet lacks accountability at the network layer: it is very difficult, if not impossible, for the receiver of an IP packet to associate the packet with its real sender, as the sender is free to craft any part of the packet [4].

Although various other captivating approaches have been suggested (e.g. [6], [7], [8], [9], [10]), the current state is that DoS attacks are not fully mitigated.

B. *CAPTCHA*

DoS attack is usually performed by means of bots, automated software. These bots send a large number of fake requests to the server which exceeds server buffer capacity which results in DoS attack. DoS attacks have become more powerful in the last several years as the level of attack automation has increased.

One approach to prevent this problem is CAPTCHA verification. The CAPTCHA submitted by user is verified before allowing the access to credentials page. The CAPTCHA would consist of variety of patterns that would be distinct in nature and are randomly generated during each visit to the web page. Most of the current web sites use a common methodology to generate all its CAPTCHAs. The bots usually take advantage of this approach since bots are able to decipher those CAPTCHAs. A set of distinct CAPTCHA patterns prevents bots to decipher it and consequently helps to reduce the generation of illicit traffic [5].

CAPTCHA is an acronym for "**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part" is a type of challenge-response test used in computing to determine whether or not the user is human.

A new concept called reCAPTCHA is introduced by Google is a free service to protect your website from spam and abuse. reCAPTCHA uses an advanced risk analysis engine and adaptive CAPTCHAs to keep automated software from engaging in abusive activities on your site. It does this while letting your valid users pass through with ease. reCAPTCHA offers more than just spam protection. Every time our CAPTCHAs are solved, that human effort helps digitize text, annotate images, and build machine learning datasets. This in turn helps preserve books, improve maps, and solve hard AI problems.

The CAPTCHA tests must be:

- Easy for humans to pass.
- Easy for a tester machine to generate and grade.
- Hard for a software robot to pass. The only automaton that should be able to pass a CAPTCHA is the one generating the CAPTCHA.
- Having Accessibility option for the visually impaired.

III. Proposed System

The arriving HTTP requests first encounter a reverse proxy. Here they are filtered based on a set of rules. A decision engine uses statistical anomaly detection to generate rules for filtering traffic and sending suspicious traffic to the security module which results in presenting the end user with a CAPTCHA to verify they are a legitimate user. Experiments performed on a scalable implementation demonstrate effective mitigation of attacks launched using a real-world DoS attack tool.

The security module uses a novel approach to mitigate flooding based DoS attacks by incorporating connection limits that are used to identify and block malicious traffic, logging of flooder IPs and alerts that are triggered when a connection limit is exceeded.

The rule created for anomaly detection is well defined as 10 requests per second and is successful to mitigate false positives. Our proposed system thereby ameliorates the problem of false positives and secures the application.

The data flow diagram described below shows the functioning of the proposed system as HTTP requests are made to the server.

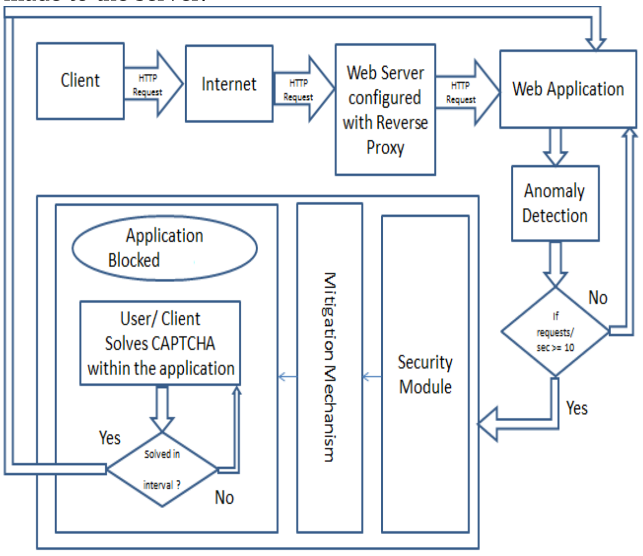


Fig. 1 Data Flow Diagram of the proposed system

A. *Approaches for Security Module to Prevent Flooding*

There are two approaches defined for the security module to prevent flooding attacks more efficiently.

1) Regulating Request: The basic approach is that if more requests are made during a certain period then we identify those IP addresses violating the predefined rules. If the rate of requests per second is more than 10 then action is required to prevent the IP address of the request. These IP addresses are redirected to a low resource consuming page having a CAPTCHA question. If the user is able to correctly answer the question within the allotted time of 20 seconds then they are able to access the web application.

2) Access Deny for each request: This is an approach that denies access to all incoming requests. It treats each request as malicious. The users are directed to a CAPTCHA page where they have to correctly solve the question within the allotted time. If solved, they are proved to be legitimate users; else, the application remains blocked until the CAPTCHA is solved and the IP address gets blacklisted.

B. *Benefits of Proposed System*

The proposed method to detect flooder IP addresses and mitigate HTTP flood attacks at the web application level has following features:

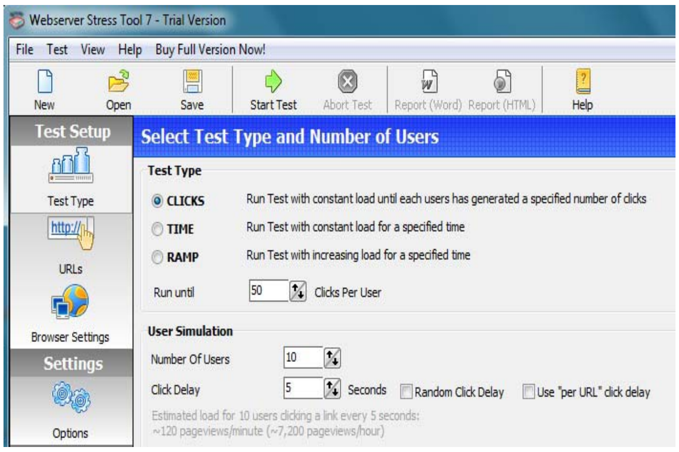
- All pages of the Web application invoked by the use of a common script.
- Each IP address is recorded micro-seconds of his time to request information be recorded.
- When requests are made from the same IP address information comparison.
- White exceptions for IP addresses in the list creation.
- Counter value created in the IP address and time information recorded with some of the micro seconds.
- Termination process for determining a timeout value.
- If a user does not want to wait for real this time, the CAPTCHA to be asked.
- Blacklist file containing details of the IP addresses which are blacklisted and cannot access the network.
- Excluded List file contain details of the IP addresses which are excluded from ever accessing the network.

This method also has the following benefits for utilizing CAPTCHA:

1. Simple to implement.
2. Easily identify between human generated traffic and robot generated traffic
3. Server resources are marginally consumed. Since CAPTCHA verification process takes less time as compared to verify the username and password.
4. Even a simple modification in the CAPTCHA makes it difficult for a bot to decipher a CAPTCHA.
5. Access Deny for each request approach which has CAPTCHA question for all users secures the application.

IV. Performance Evaluation

In order to test the feasibility of the system we evaluate our performance using Web Server Stress Tool 7. This software enables us to create a DoS attack on a selected URL. We assign the number of clicks per user so as to generate a sufficient number of clicks needed for attack. Once the web application without included security module is attacked we can see that the attack takes place causing loss of bandwidth and resources.


Webserver Stress Tool 7 - Trial Version
File Test View Help Buy Full Version Now!
New Open Save Start Test Abort Test Report (Word) Report (HTML) Help
Test Setup
Test Type
URLs
Browser Settings
Settings
Options

Select Test Type and Number of Users

Test Type
- (●) CLICKS — Run Test with constant load until each users has generated a specified number of clicks
- ( ) TIME — Run Test with constant load for a specified time
- ( ) RAMP — Run Test with increasing load for a specified time

Run until 50 Clicks Per User

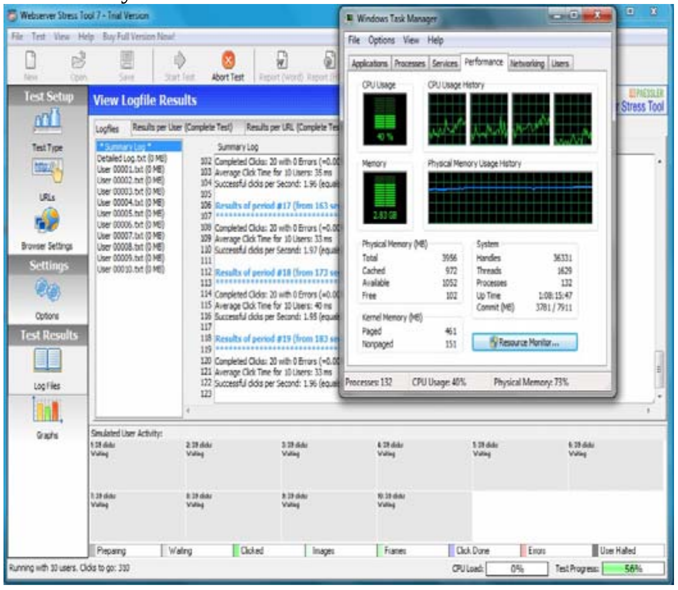
User Simulation
Number Of Users 10
Click Delay 5 Seconds ☐ Random Click Delay ☐ Use "per URL" click delay

Estimated load for 10 users clicking a link every 5 seconds:
~120 pageviews/minute (~7,200 pageviews/hour)

Fig. 2: Testing performance using Web Server Stress Tool 7

Now we include our security module in the application and again start the test. It is evaluated that the surface attack is reduced and no loss of resources or bandwidth is witnessed. The system's performance is not affected and thus this test concludes the importance of our module in detecting and mitigating flood attacks.

Firstly, the application is tested without any security module present. It is observed that when DoS attack is emulated on the application, the CPU and Memory usage is higher than usual. The resources are wasted and performance of the system is impaired. This can be controlled by utilizing our security module which successfully saves loss of resources and bandwidth.


Webserver Stress Tool 7 - Trial Version
File Test View Help Buy Full Version Now!
New Open Save Start Test Abort Test Report (Word)
Test Setup
Test Type
URLs
Browser Settings
Settings
Options
Test Results
Log Files
Graphs
Stress Tool
Windows Task Manager
File Options View Help
Applications Processes Services Performance Networking Users
CPU Usage
CPU Usage History
Memory
Physical Memory Usage History
Resource Monitor...

View Logfile Results

Logfiles | Results per User (Complete Test) | Results per URL (Complete Te

```
* Summary Log *
Detailed Log.txt (0 MB)
User 00001.txt (0 MB)
User 00002.txt (0 MB)
User 00003.txt (0 MB)
User 00004.txt (0 MB)
User 00005.txt (0 MB)
User 00006.txt (0 MB)
User 00007.txt (0 MB)
User 00008.txt (0 MB)
User 00009.txt (0 MB)
User 00010.txt (0 MB)
```

```
Summary Log
102 Completed Clicks: 20 with 0 Errors (=0.00
103 Average Click Time for 10 Users: 35 ms
104 Successful clicks per Second: 1.96 (equa
105
106 Results of period #17 (from 163 se
107 ********************************
108 Completed Clicks: 20 with 0 Errors (=0.00
109 Average Click Time for 10 Users: 33 ms
110 Successful clicks per Second: 1.97 (equa
111
112 Results of period #18 (from 173 se
113 ********************************
114 Completed Clicks: 20 with 0 Errors (=0.00
115 Average Click Time for 10 Users: 40 ms
116 Successful clicks per Second: 1.95 (equa
117
118 Results of period #19 (from 183 se
119 ********************************
120 Completed Clicks: 20 with 0 Errors (=0.00
121 Average Click Time for 10 Users: 33 ms
122 Successful clicks per Second: 1.96 (equa
123
```

CPU Usage 40 % Memory 2.83 GB

Physical Memory (MB)		System	
Total	3956	Handles	36331
Cached	972	Threads	1629
Available	1052	Processes	132
Free	102	Up Time	1:08:15:47
		Commit (MB)	3781 / 7911

Kernel Memory (MB)	
Paged	461
Nonpaged	151

Processes: 132 CPU Usage: 40% Physical Memory: 73%

Simulated User Activity:

```
1: 19 clicks   2: 19 clicks   3: 19 clicks   4: 19 clicks   5: 19 clicks   6: 19 clicks
Waiting        Waiting        Waiting        Waiting        Waiting        Waiting
7: 19 clicks   8: 19 clicks   9: 19 clicks   10: 19 clicks
Waiting        Waiting        Waiting        Waiting
```

Preparing | Waiting | Clicked | Images | Frames | Click Done | Errors | User Halted

Running with 10 users. Clicks to go: 310   CPU Load: 0%   Test Progress: 58%

Fig. 3: Testing Memory and CPU usage of application under Dos Attack

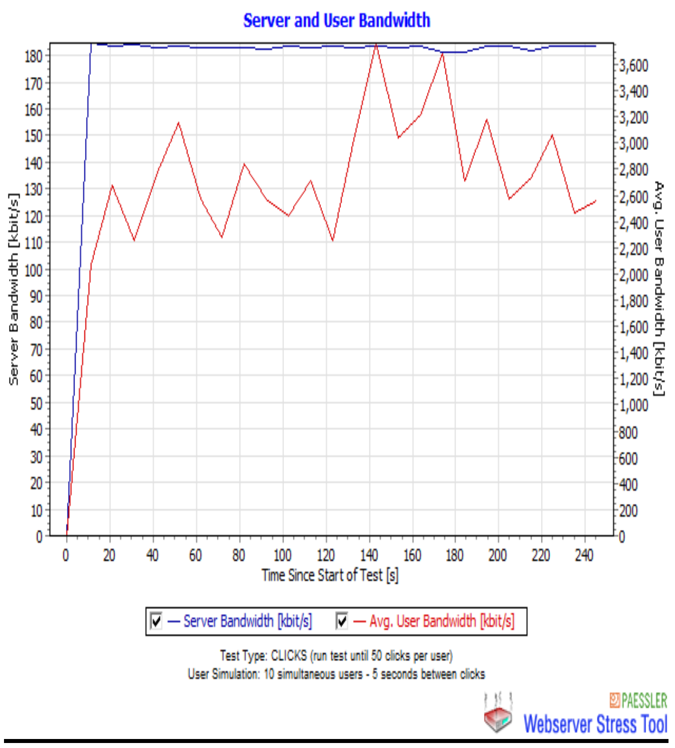


Fig. 4 The Kbits/sec of DoS traffic that reached the Web Application without security module

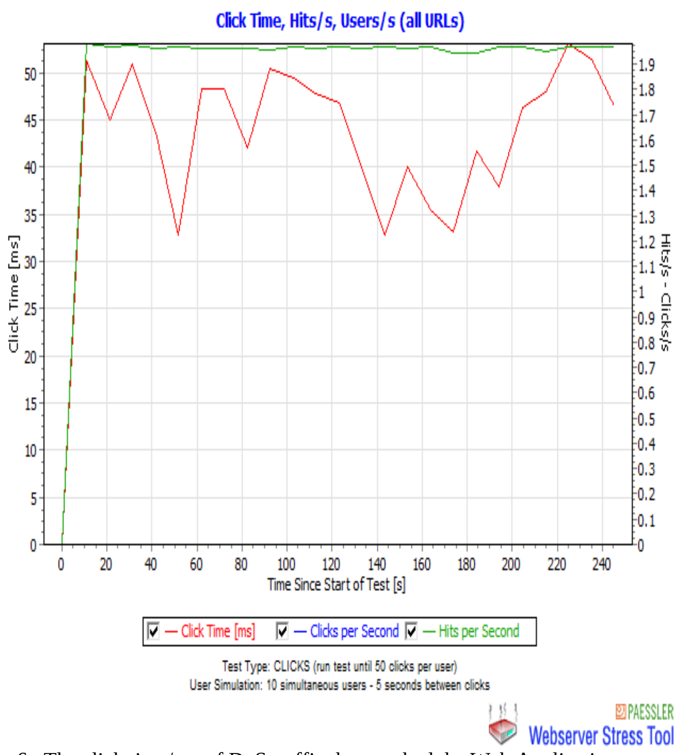


Fig. 6: The click time/sec of DoS traffic that reached the Web Application without security module

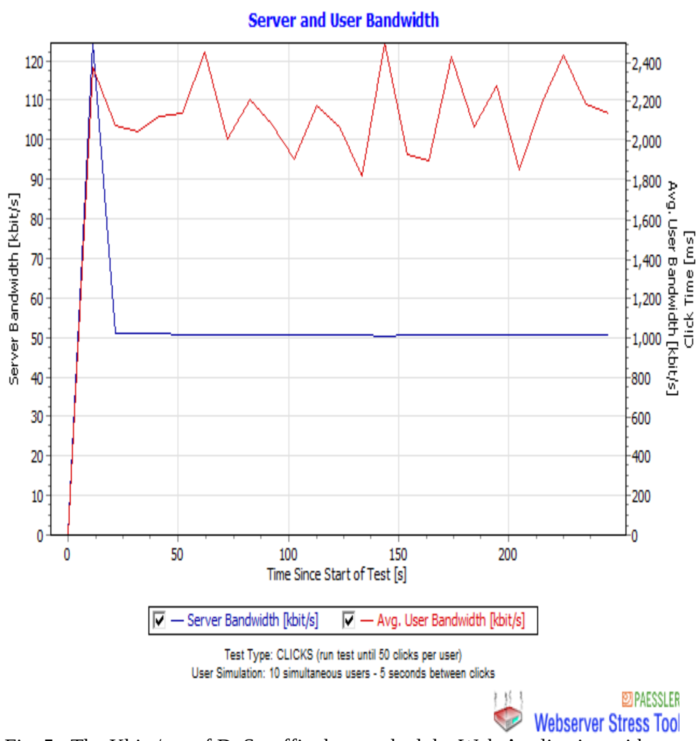


Fig. 5: The Kbits/sec of DoS traffic that reached the Web Application with security module

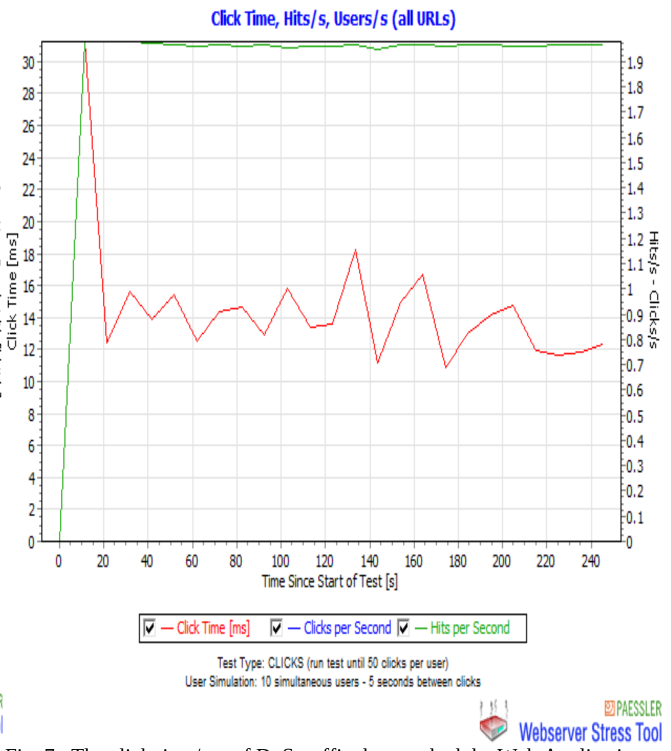


Fig. 7: The click time/sec of DoS traffic that reached the Web Application with security module

V. Conclusions

We have demonstrated a modular test system that effectively detects and mitigates flooding based DoS attacks. This system successfully ameliorates the problem of false positives and secures the application. This system implementation is also capable to slow down the Brute Force Attacks. Furthermore, the detected IP addresses when shared with other security components can block attackers' access to the web application.

The system is verifiably easy to include and the proposed CAPTCHA mechanism makes it relatively easier to implement. The benefit being that it blocks flooder IP addresses by differentiating between human and bot.

By evaluating its performance with an experiment to create DoS attacks, our system detected and mitigated the attack. It was able to restore normal response times to a web application that was experiencing a DoS attack. It did so efficiently and reduced the impact on legitimate traffic.

References

1. L. Garber. Denial-of-Service Attacks Rip the Internet. IEEE Computer Magazine, 33, 2000.An Anomaly-driven Reverse Proxy for Web Applications
2. Breach Security. Breachgate.http://www.breach.com/, June 2005.
3. Kruegel and G. Vigna. Anomaly Detection of Web-based Attacks. In Proceedings of the 10th ACM Conference on Computer and Communication Security (CCS '03), pages 251{261, Washington, DC, October 2003. ACM Press.
4. Liu, Xin, "Mitigating Denial-of-Service Flooding Attacks with Source Authentication", In Diss. Duke University, 2012
5. M Mehra, M Agarwal, R Pawar, and D Shah. "Mitigating denial of service attack using CAPTCHA mechanism". In Proceedings of the International Conference & Workshop on Emerging Trends in Technology, pages 284-287. ACM, 2011.
6. J. Mirkovic, G. Prier, and P. Reiher, "Attacking DDoS at the Source," in Proceedings of the ICNP 2002, November 2002.
7. Jelena Mirkovic, D-WARD: Source-End Defense Against Distributed Denial-of-Service Attacks, Ph.D. thesis, University of California Los Angeles, 2003.
8. D Xuan, R Bettati, and W Zhao, "A gateway-based defense system for distributed dos attacks in high-speed networks," in Proceedings of 2001 IEEE Workshop on Information Assurance and Security, June 2001.
9. Debra L. Cook, William G. Morein, Angelos D. Keromytis, Vishal Misra, and Daniel Rubenstein., "Websos: Protecting web servers from ddos attacks," in In the Proceedings of the 11th IEEE International Conference on Networks (ICON)., 2003..
10. J. Ioannidis and S. M. Bellovin, "Pushback: Router-Based Defense Against DDoS Attacks," in Proceedings of NDSS, February 2002.