A Cost Effective Economic Denial of Sustainability (EDoS) Attack Mitigation Framework for E-Commerce Applications in Cloud Environments

By

Muddassar Masood

2009-NUST-MS-PhD IT-21

Supervisor

Dr.Zahid Anwar

Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Information Technology (MS IT)

In

School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(August 2013)

Approval

It is certified that the contents and form of the thesis entitled "**A Cost Effective Economic Denial of Sustainability (EDoS) Attack Mitigation Framework for E-Commerce Applications in Cloud Environments**" submitted by **Muddassar Masood** have been found satisfactory for the requirement of the degree.

Advisor: **Dr.Zahid Anwar**
Signature: _____

Date: _____

Committee Member 1: **Dr. Hafiz Farooq Ahmad**

Signature: _____
Date: _____

Committee Member 2: **Dr.Fauzan Mirza**

Signature: _____
Date: _____

Committee Member 3: **Dr. Ali Mustafa**

Signature: _____
Date: _____

Abstract

The promise of pay-as-you-go and scalable model of Cloud Computing has attracted a large number of medium and small enterprises to adopt E-Commerce model of conducting on-line businesses. While E-Commerce applications on the Cloud expand businesses by making them more widely accessible, they also makes these applications susceptible to economic denial of service attacks - a form of application layer attacks that drive up the cost of Cloud computing by using up application resources. This paper focuses on detection and mitigation of EDoS for E-Commerce based applications. EDoS is different from traditional DDoS in that, the intention of the latter is to consume all the resources (like memory, bandwidth, CPU etc) of the Web Server thus making it unavailable to its legitimate users. EDoS on the other hand is caused by malicious users who are not interested in following the regular workflow of an E-commerce application by purchasing items but by employing it for their own purposes of entertainment, price-checks and idle surfing. We have a twofold solution, (i) admission control and (ii) congestion control. In the first, we limit number of clients that can simultaneously send requests, thus allowing only enough clients that can be served easily within available resources on the Web server. In the second, we change the priority of allowed clients based on the type of resources they visit and type of activities they perform, thus making the maximum resources available to good clients. Our contribution is two-fold: (1) We model the workflow of a typical E-Commerce application and identify key parameters that identify good and bad users and (2) we present the design of a learning-based classifier that distinguishes good and bad users depending on the values of the parameters they select while web browsing. We have integrated and evaluated this solution in a Web Application Firewall and found it quite effective in term of resources distribution among good and bad clients.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Muddassar Masood**
Signature: _____

Acknowledgment

I am thankful to Almighty Allah, whose blessings have always been enormous and who bestowed the skills, knowledge and strength upon me to complete this thesis.
I offer my sincere gratitude to my supervisor Dr. Zahid Anwar, who has supported and guided me throughout my thesis work with his patience and knowledge. This thesis would not have been completed without his guidelines and encouragement.
I wish to take this opportunity to express my heartiest thanks and deepest sense of gratitude to Dr. Hafiz Farooq, Dr. Fauzan Mirza and Dr. Ali Mustafa who gave me opportunity to conduct this study.
I would also like to thank all my friends and family members for their continuous encouragement and moral support.

I dedicate this thesis to my mother who was always around me whenever I needed her. It is her unconditional love that motivated me to complete this work.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Cloud Computing is a new trend that enables the delivery of computing resources via the Internet. Cloud service providers provide customizable facilities like Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) with minimal maintenance cost to their consumers, thus eradicating the upfront cost of hardware needed for infrastructure setup. Moreover Cloud services are typically highly flexible, scalable owing to the ready availability of almost unlimited resources, making it ideal for organizations which don't want to buy the entire infrastructure required upfront. Cloud Computing follows a pay per use pricing model. Besides all these benefits, Cloud Computing does come at the cost of increased security risks which is currently one of the biggest challenges this technology is facing today, limiting the number of organizations willing to embrace it wholeheartedly. The impact of attacks that previously plagued traditional networked environments increases manifold when applied to the emerging Cloud environments. According to (Jackson, 2011) DDoS is the next battleground for cyber security considerations and is observed as the most devastating and at the same time the most prevailing attack of the current era. According to DDoS security vendor Prolexic (Prolexic, 2013), DDoS attack incidents reported in 2012 were more than 25% of the attacks reported in 2011. Among them, there is a 25% increase in application layer denial of service attacks and this rate is constantly increasing with time (Prolexic, 2013). The intention behind application layer distributed denial of service attacks (see Figure 1.1) is to exhaust the application with a large number of requests, so that it may crash thus making it unavailable to its legitimate users.

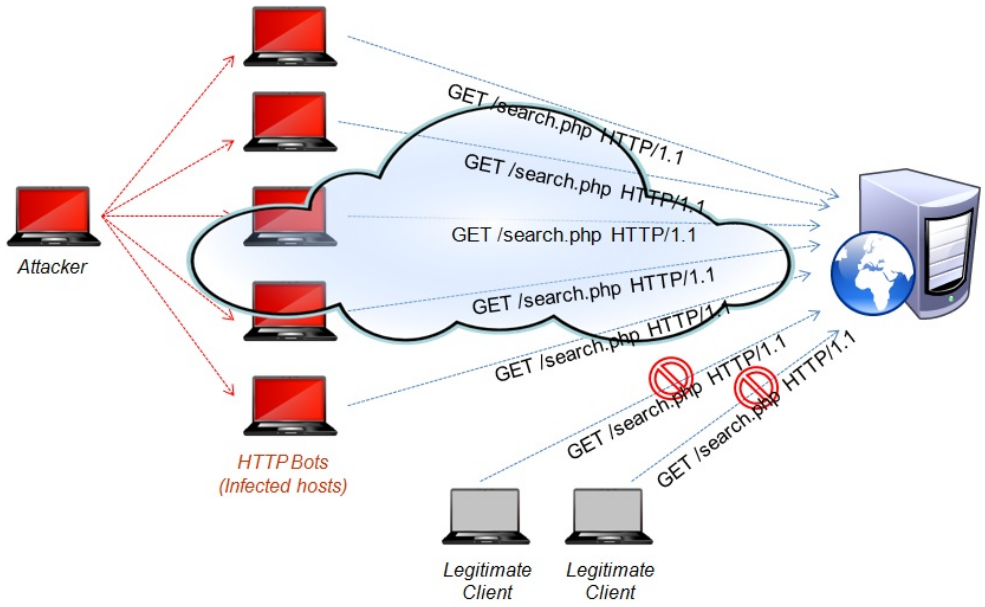

Figure 1.1: Application DDoS Scanario

DDoS attack in traditional networked (non-Cloud) environment usually disrupts the service which hurts reputation and incurs economic loss. In Cloud environments, disrupting a service is not so easy due to its inherent capability of auto-scalability and service level agreements (SLA). However, DDoS attempts on Cloud environments have another more alarming repercussion in that it the consumption of more Cloud resources to provide auto-scalability, normally exceeds the economic bounds for service delivery, thereby incurring Economic Denial of Sustainability (EDoS) for the organization whose service or Virtual Machine (VM) is targeted. EDoS (VivinSandar and Shenai, 2012) is new breed of DDoS attack specific to Cloud environments. In this case, the Cloud service provider activates more and more resources to meet the SLA for the availability of the service for the customer, which eventually adds extra billing cost leading to EDoS. With the advent of the Cloud, a wrong perception has been prevailing that DDoS attacks will no longer be a problem anymore, because it has the tendency to absorb the load in terms of read availability of resources. What consumers don't realize is that on the Cloud, resources are metered on resource billing. Fraudulent consumption of bandwidth and computational resources of Web based cloud services incur financial burden on the Cloud consumer (Idziorek and Tannian, 2011). Table 1.1 depicts the pricing model for one Linux server on Rack Space cloud, which clearly shows how the cloud consumer has to pay exorbitant rates for extra usage of resources as in the unintentional case of DDoS attack. The

motivation of our work is to save this extra cost for the cloud consumer and at the same time making the service more readily available.

Linux Servers			
	RAM	**Bandwidth**	**Monthly Cost**
1	512MB	1GB	**$203.78**
2	1GB	2GB	**$231.64**
3	2GB	5GB	**$275.80**
4	4GB	10GB	**$364.00**
5	8GB	15GB	**$539.80**
6	15GB	30GB	**$848.20**
7	32GB	50GB	**$1,069.60**

Table 1.1: Pricing for Rack Space Cloud server (Managed)

It has been observed that the conversion rate i.e. percentage of visits that leads to purchase and eventual profit is very low for E-Commerce applications which means there are less number of valuable customers who consume application resources than are non-valuable customers. According to (Moe and Fader, 2004) purchase conversion (converting store visits to purchases) rate for E-Commerce application is still less than 5%, which means majority of visitors of applications utilize application resources for only the purpose of window-shopping. Work by (Idziorek et al., 2011) considers fraudulent use of resources on the Cloud a major threat for Cloud consumers apart from security threats, that can cause EDoS quite easily. EDoS can be performed against an E-Commerce application by making simple legitimate requests that have heavy workload effect on the hosting application e.g. requesting large media files (image, videos), making frequent searches on entire product range, which results in large queries on backend databases such as involving joining of tables. Workload can be on any of the resources -bandwidth, processing, memory etc. We assume in this work that customers that visit E-Commerce applications on the Cloud utilizing valuable resources with no intention of making eventual purchases cause a type of EDoS attack.

In this article, we assert that application DDoS is the major cause of EDoS in Cloud Computing environments, and explored application DDoS techniques and proposed a defense solution named EDoS Armor to mitigate it. EDoS Armor is a multi-layered defense system; firstly, we use a challenge mechanism to restrict bots or scripts from flooding the target, secondly, we have an admission control which authenticates only limited number of valid users, to avoid over burdening the hosting application or server. Thirdly, we have

congestion control which uses a browsing behavior learning mechanism and assigns priority to good users on this basis. This priority value decides the distribution of resources among users, so the higher the priority, the higher the bandwidth allocated to browse resources and vice versa.

1.2 Motivation

DDoS attacks are observed as the most devastating and at the same time most prevailing attack of the current era, whether one has resources on Cloud environment or non-Cloud environment. DDoS attack alone has caused a great deal of damage to various businesses worldwide; among the major affectees are Sony PlayStation Network, the Hong Kong stock exchange, Visa, MasterCard, PayPal, Word Press and Joomla. According to DDoS security vendor Prolexic, DDoS attack incidents reported in 2013 are more than the attacks reported in 2012 (Prolexic, 2013). Moreover Cloud Security Alliance (CSA) annoounced the top nine security threats on the Cloud and DoS attack is on the 5th position (CSA, 2013), which strengthened our motivation for this work.

1.3 Thesis Aims and Objectives

The main aims and objectives of this thesis are as follows:

I. To provide deployable defense mechanism against Economic Denial of Sustainability (EDoS) attack on E-commerce applications in the Cloud.

II. Defense system should be efficient together with effectiveness.

1.4 Thesis Contribution

In this research, we have solved the following issues and produced the subsequent contributions in the field of access control models.

I. Modeled the workflow of a typical E-Commerce application and identified key parameters that differentiate good and bad users.

II. Designed a learning-based classifier that distinguishes good and bad clients depending on the values of the parameters they select while web browsing.

1.4.1 Conference Publication

The conference paper on this research work has been written with the title *EDoS Armor: A Cost Effective Economic Denial of Service Attack Mitigation Framework for E-Commerce Applications in Cloud Environments*, and submitted in INMIC 2013 for publication.

1.4.2 EDoS Armor Module

This work serve as a EDoS/DDoS Defense mechanism for E-Commerce applications and can be integrated in a Web Application Firewall. As a test case, I have integrated it in a Semantic based Web Application Firewall (SWAF). EDoS defense enabled SWAF works as a reverse proxy and can be placed in front of any Web server to provide defense for e-commerce applications hosted in Web server.

1.5 Thesis Organization

Rest of the thesis is organized as follow: Chapter 2 describes the discusses related work and existing solutions. Chapter 3 describes system architecture and design of proposed system. Chapter 4 talks about implementation details. Chapter 5 describes testing and evaluation of the system and in the end Chapter 6 concludes thesis work and highlights future research directions.

Chapter 2

Related Research

There is a large body of literature available on DDoS defense mechanisms, but the main limitation is that it mostly focuses on techniques for preventing flooding of seemingly legitimate requests at the network or application layer. We were unable to find lightweight and deployable solutions particularly for mitigating EDoS attacks in Cloud Computing environments where E-Commerce like applications are accessed by bots or by legitimate users but with malicious intent. In these environments the examination of packet payload or violation of HTTP and related web and IP protocols is insufficient and browsing behavior also needs to be taken into consideration. We found very limited work on EDoS for Clouds as it is a fairly new concept. Our review below therefore considers in large part both EDoS detection techniques as well as relevant application layer DDoS detection techniques.

2.1 Traditional DDoS Defense Techniques

(Yu et al., 2010) proposed a trust management helmet (TMH) which uses trust as a base to differentiate between a legitimate user and an attacker. The trust value is calculated on the basis of visit history of a particular client and is stored with the client. (Yu et al., 2010) defined different trust measuring factors like short-term trust, long-term trust, negative trust & misuse trust, on the basis of which overall trust is calculated for a client. This trust is to ensure connectivity of good users in the situation of attack. A license is maintained on client side through cookies, hashed with a key private. This license serves two purposes; one for user identification, and second for trust computation. For each session connection request, TMH checks whether the client is blacklisted (i.e. trust level below threshold) if yes then the request is dropped otherwise trust is computed and a connection request to the server

using trust-based scheduling is made. This technique provides partial defense against application DDoS as it handles only session flooding attacks.

(Ranjan et al., 2009) introduced a DDoS Shield mechanism integrated into the reverse proxy to either allow or deny requests before it reaching the Web server tier. DDoS-shield consists of two modules; i) suspicious assignment module which assigns suspicious value to a session based on historic behavior and ii) DDoS-resilient scheduler that decides when and what request is to be forwarded based on the scheduling policy. DDoS-shield maintains the requests history, the request arrival time and workload profile for a session. This information is used by the suspicious assignment module for assigning a suspicion value to a session. Resource utilization reports together with throughput and response time measures are monitored periodically to detect the DDoS attack without involving the lower layers. This technique involves time-related characteristics for HTTP sessions like session inter-arrival time, request inter-arrival time and session arrival time, which are insufficient for detecting application DDoS because intelligent bots are available to attacker which provides control over the packet sending rate.

(Yu et al., 2007) characterized application layer attacks into three categories; request flooding, session flooding and asymmetric attacks. The objective is to maximize a normal user's service request rate and minimize his delay for service. The authors named their technique as DOW (Defense and Offense Wall) which provides defense against application DDoS by a combination of detection and currency technology. In detection model, suspicious sessions are dropped while in currency model, more legitimate sessions are encouraged. An anomaly detection method based on K-mean clustering is used to detect and filter request flooding and asymmetric attacks (Yu et al., 2007). To provide defense against session flooding attack, an encouragement model is proposed that uses the client's session rate currency. Both the methods in collaboration can provide normal clients, high service rate and lower delay response time. Shortcoming of DOW is that it requires large amount of training data for its detection model.

(Xie and Yu, 2009) authors introduce a new scheme to achieve early attack detection and filtering for application-layer DDoS attack. For this, an extended hidden semi-Markov model is proposed to describe the browsing behaviors of web surfers. In order to reduce the computational amount introduced by the model's large state space, a forward algorithm is derived for the online implementation of the model based on the M-algorithm. Entropy of the user's HTTP request sequence in reference to the model is used as

a criterion to measure the user's normality. If the entropy is larger than a predefined threshold, the user is considered as abnormal, and the request sequence will be discarded. Otherwise the user is considered as benign. The biggest problem of Hidden Semi-Markov method is the algorithmic complexity. Although authors improved this drawback based on M-algorithm, it was still found that tracking each user's visiting sequence was not a practical task (Wen et al., 2010)

(Ali, 2009) proposed a light weight mechanism for mitigation of session flooding and request flooding attacks. To mitigate session flooding attacks, user's visiting history is evaluated and a trust value is assigned. Requests are then scheduled in decreasing order of trust to allocate available session to the user. For request flooding attack, the concept of source throttling is used. This mechanism will get activated only when the server is under load. The server will send a puzzle to the client along with JavaScript to solve it, hence CPU cycle of client are utilized as apposed to that of the server. Clients who will send correct results of the puzzle will get their requests processed and not otherwise. Thus request flooding is handled through throttling client's CPU. This technique is limited to only session flooding and request flooding attack. It involves client side changes which can malfunction in case of users using unconventional Web browsers.

(Wang et al., 2011) discussed the detection of application DDoS by presenting two models; click-ratio based model and Markov process based model, to characterize Web clients access behavior. Authors then used Large Deviation (LD) theory to measure the deviation of ongoing client's access behavior from priori normal Web client access behavior for these two models and come up with two detection schemes LD-IIP (click-ration) and LD-MP(Markov process). Large deviation theory is mainly used to compute the probability of occurring rare events in a system. LD-IIP technique proved better in comparison to LD-MP by results.

In (Lee et al., 2011) authors proposed a sequence order independent method for profiling web browsing behavior to detect application DDoS attack. Sequence order is considered to be more harmful than helpful in the profiling of web browsing behavior because it varies significantly for different individuals and different browsing behaviors. Authors extracted sequence-order-independent informative attributes from web page request sequences; like web user's activeness, pages of interest, and the breadth and intensity of their interest. These are described in a matrix and use multiple principal component analysis (PCA) to model profiles of normal browsing patterns.

Authors used the reconstruction error of the multiple PCA as a criterion for distinguishing Application-DDoS attacks from normal usage. This technique lacks automatic learning which make it a non practical solution.

In (Srivatsa et al., 2008) the author proposed a twofold mechanism for mitigation of application DDoS. First is admission control to limit the number of concurrent clients served by the online service. It is based on port hiding that renders the online service invisible to unauthorized clients by hiding the port number on which the service accepts incoming requests. Second is congestion control on admitted clients to allocate more resources to good clients. Congestion control is achieved by adaptively setting a client's priority level in response to the client's requests in a way that can incorporate application-level semantics (Srivatsa et al., 2008). This technique has limited incorporation of application level semantics as it is only suitable for static applications.

2.2 EDoS Defense Techniques

(VivinSandar and Shenai, 2012) proposed a defense framework for mitigation of HTTP and XML DDoS attack on the Cloud. Authors have talked about a new breed of DDoS on the Cloud called Economic Denial of Sustainability (EDoS) which targets unfair utilization of resources of service on the Cloud thus causing economic loss to consumer. The author proposed a combination of a firewall and challenge server to provide defense against EDoS. The challenge server sends the challenge to the client, and if the client is able to solve that challenge then a rule is added into firewall for that client to allow its future requests. Conversely if the client is unable to solve the challenge which means it is some type of bot, then a rule is added to the firewall to block its requests in the future. This technique is good in that it limits traffic from bots but it has no defense for situation where bots are utilized after solving challenge by humans. Moreover this technique is too basic to provide a stable defense mechanism for EDoS the Cloud Computing version of DDoS.

2.3 Conclusion

In our literature study, we have noted that there are a lot of defense mechanisms available for application DDoS but few for Cloud based EDoS which are not enough mature. Both categories of defenses try to ensure availability

of resources to legitimate clients. But some techniques are complex making them unpractical, some require more training data, some are not adaptable to new circumstances, and some become the target of DDoS themselves. Our proposed technique is client transparent, requires less training data, and provides a multi-layered defense for mitigation of EDoS in Cloud environments specific for E-Commerce applications, which is the most popular category of applications hosted on Clouds.

Chapter 3

Design and Methodology

3.1 Threat Model

In this article, we assumed that the attacker can launch attack on a Web or application service hosted in the cloud. We also assumed a sophisticated attacker that can launch both network as well as application layer DDoS. This means the attacker targets TCP/IP layer exploits for DDoS e.g. sync flooding attack. Objective of network layer DDoS attacks is to exhaust processing, network and memory resources of target server by flooding with either large number of IP packets or TCP connections requests. Moreover he can launch application layer DDoS attack. Due to advanced application server technologies, modern application server now perform complex operations like heavy weight transactional and database queries processing. By understanding application semantics, the attacker can craft a DDoS attack, which involves highly resource intensive requests for instance in an online book store, the attacker can perform frequent searches, which consume bandwidth and put processing load on the backend database. Hence application level DDoS attacks are harder to detect at IP layer because of unavailability of application-layer semantics that are mostly only known to the application developers.

3.2 Defense Model

DDoS attack is detrimental to applications deployed on both Cloud as well as well as conventional web service based environment. In Cloud Computing environments a DDoS attack manifests itself as an Economic Denial of Sustainability EDoS attack. To provide an effective defense against DDoS attack we must have client transparency and light weight mechanisms and ensure

that our defense strategy covers both application layer and well as network layer DDoS attacks. Security experts also recommend defense for DDoS to be multi-layered (Durcekova, 2013). In this regards, we found IBM solution for application DDoS (Srivatsa et al., 2008) which also tried to achieve more or less the same objectives, but it has talked at a very abstract level about the algorithm and attributes used, thus it has enough roam for details and improvements. So we designed EDoS Armor, focusing the missing and ambiguous part of (Srivatsa et al., 2008) and added some improvements in it and finally came up with a more improved solution for DDoS over cloud.

3.3 Architecture

Similar to (Srivatsa et al., 2008) EDoS Armor is a multi-layered defense for application DDoS. First we restrict the bots or automated scripts by mean of a challenge; challenge-based mechanism is a neat way to throttle the intensity of a DDoS attack. Due to readily availability of free tools for application DDoS, things are quite easy for attacker in term of flooding the target with a high or customized rate of HTTP request to crash the target application or service. So to avoid this damage, we have a challenge server in place, which allows to differentiate bots or scripts from human by throwing a challenge in response to first request from client, if client responds by solving the challenge correctly, then it is allowed to communicate further, otherwise it is considered as bot or automated script, and its requests are blocked.

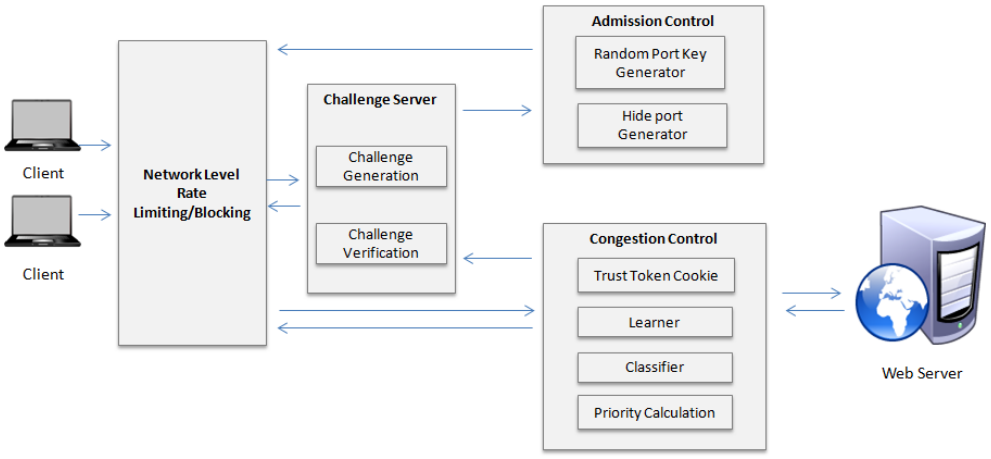

Figure 3.1: Architecture of EDoS Armor Defense

Now if requester is a human user, which responds with correct solution of challenge, it is authenticated into our admission control module, and assigned

a secret/hide port on which further communication is carried out. This hide port will be invisible to other non-authenticated clients. In this way, we are also limiting number of concurrent clients that can be served by the service at one time. Another advantage of invisible hide port is that, unauthenticated clients can't even launch network DDoS (e.g. TCP SYN flooding attack) on the server, thus limiting them before than they try to consume CPU, memory and network resources high above the network stack.

Next comes the congestion control module, which is to keep a check on the admitted clients. For this, a priority value is maintained; a priority is dynamically allocated on the basis of client's history or on the basis of requests types keeping in view the application semantics. Motivation for this is to make maximum resources available to good clients in comparison to bad clients.

EDoS Armor works in combination of HTTP and IP layers i.e. HTTP layer on client side i.e. making it client transparent using Web browser and IP layer on server side i.e. packet filtering performed before it reaches application layer up the stack, thus saves a lot of computation, memory and disk resources for processing of packets.

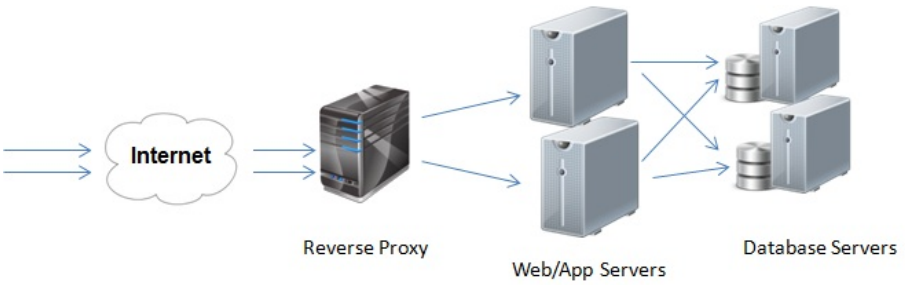

Figure 3.2: Deployment Scenario for EDoS Armor

Our solution is client transparent, as it does not require any extra software to be installed at client side. DDoS Armor will be deployed as a reverse proxy in front of Web server as shown in figure 3, thus making it a deployable solution as it does not require any extra hardware. By evaluation, we have come to know that our solution is light weight in term of processing thus there is no performance overhead.

Chapter 4

Implementation

4.1 Challenge Server

When the client tries to start a session with the server by making first request, its request is forwarded to challenge server which sends a challenge to every client at the start of the session. This challenge can be of image based or any cryptographic challenge. If the client solves the challenge correctly, then authentication into admission control is possible, otherwise further requests from that IP are dropped. Challenge mechanism is most widely used for dealing with flooding attacks.

4.2 Admission Control

The admission control module is used to limit the number of clients that can connect to Web server at one time. The clients which have solved the challenge correctly, provided by the challenge server, can only get authenticated in admission control and hence can communicate with the web server.

When the client solves the challenge correctly, the challenge server sends the client a random port key. On the basis of which hide port is calculated. This hide port is calculated using the client IP, server IP, current time and the port key. This hide port calculation process is carried out on both server and client end. Port key is shared with the client at HTTP layer in browser cookie. After hide port calculation at the server, it starts listening on the hide port for that client, while client starts communicating to server over newly calculated hide port i.e. send request on hide port, and further communication is carried out through this hide port. The hide port is a 16 bit integer, which means that its value can vary from 0 to 65535. The server

distinguishes the clients on the basis of valid hide port, which is checked from a table maintained on the server containing client IP with allocated hide port. Only the clients, who have valid hide port, can connect to web or application server. To keep the communication totally safe from attackers, hide port changes periodically during the session. The hide port calculation involves port key which in term involves server key. So by changing server key, new port key is generated and shared with client, on the basis of which, new hide port is calculated and further communication shifts on this.

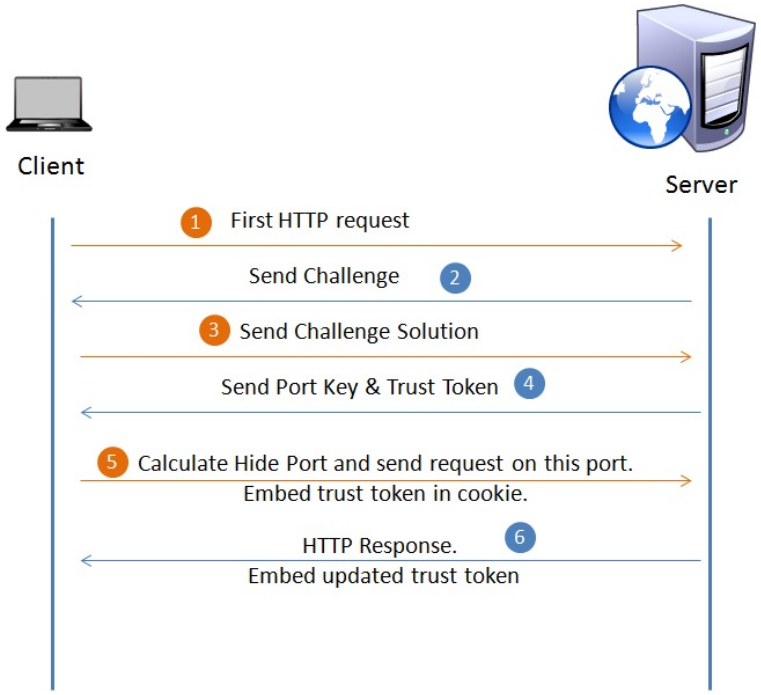

Figure 4.1: Working Scenario of DDoS Armor

The admission control filter also involves IP layer in combination of HTTP layer. It takes the benefit of network firewall i.e. iptables. Rules are added into iptables to only allow authenticated clients requests to pass through and clients with invalid ports are denied at IP layer before reaching at HTTP layer, thus prevents the consumption of CPU, memory and network resources of the server as it passes through lower layers to higher layers in network stack. Hence, without knowing the valid hide port attacker can't perform network DoS against the server because this hide port is added in destination port section of TCP packet. In other words, admission control helps in providing defense against network DDoS attack. Figure 4.1 shows the interaction of client server during a client session.

Hide port calculation process is carried out on server side as well as on the client side, and here client's transparency is achieved through JavaScript. That hide port is added as destination port in TCP packet. When packet reaches at server IP layer, this destination port is matched with hide port calculated on server side, if it matches, then this TCP packet is accepted otherwise dropped. In the case of acceptance, NAT forwarding is use to map destination port to HTTP default port i.e. 80. In this way, admission control involves IP layer together with HTTP layer for its seamless processing. Figure 4.2 below shows the pictorial representation of working of admission control.

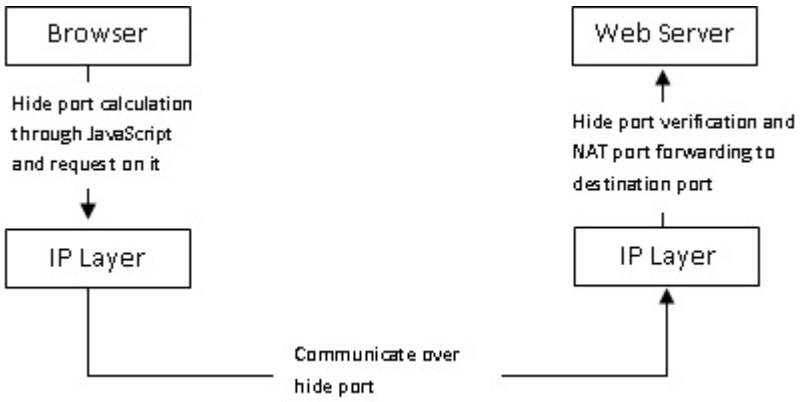

Figure 4.2: Admission Control Module

4.3 Congestion Control

The congestion filter operates on top of the admission control filter. The purpose of the congestion control filter is to perform allocation of available resources among admitted clients. Additionally it ensures that more resources are provided to good clients in comparison to bad clients. Congestion control is implemented through a client priority table that records the current priority level for every client. Each client is entertained on the basis of its priority; good client gets the high priority while bad clients get low priority. A client's priority level is adaptively varied by the server using application-specific knowledge and nature of requests made by the client.

For instance in the case of E-commerce application, one may choose to increase a client's priority level if the client performs a purchase transaction

and decrease a client's priority level if the client performs resource-intensive operations like perform heavy search queries on the application. A client's priority level defines the request rate limit for that client at IP layer. Setting the client's priority involves the incorporation of application-specific semantics (domain knowledge) together with client application access behavior. Request rate filtering at IP-level (firewall) ensures that most HTTP DDoS attack requests are dropped before they can reach above in network stack. Now there is a policy similar to (Srivatsa et al., 2008) to reward or set penalty for the good and bad clients respectively i.e. an additive increase in priority if good behavior is observed and multiplicative decrease in priority if bad behavior is observed. This behavior analysis is made with respect to legitimate client utility model. The challenge server initializes a

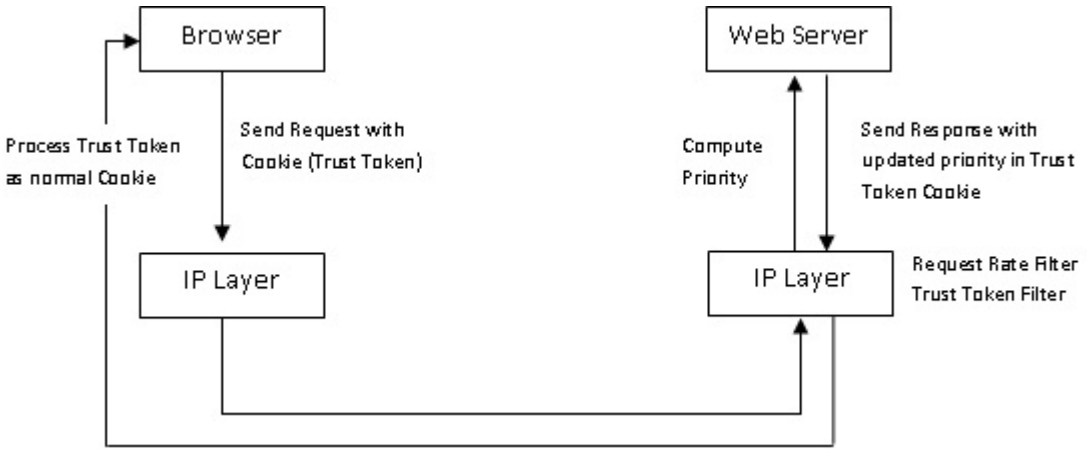

Figure 4.3: Congestion Control Module

client's priority when the client first accesses the secured Web application. We filter the traffic of admitted clients at server firewall (IP level) based on priority set at HTTP level. This filtering implements the throughput priority using weighted fair queuing, that is, it ensures that a client with priority level 100 may send twice as many requests (per time unit) than a client with priority level 50. Requests from clients attempting to issue a disproportionately large number of requests (relative to their priority level) are dropped at the IP layer itself, thereby significantly reducing the amount of processing/memory/network/disk resources consumption. The application layer implements the response time priority, that is, it ensures that a client with a higher priority level experiences a smaller response time for the processing of its requests. The tweaking the client's priority based on application-specific semantics is also done at application layer. The working of the Congestion Control module can be explained by the following figure.

The request rate γ for client is set on IP layer and requests from client are accepted at this defined rate on the server. Request rate is according to the priority ρ of that particular client at application layer. The client whose priority is high is allowed to have a high rate of request where as the client having low priority can have a lower rate of request. Thus $\rho \ \alpha \ \gamma$. If a client tries to send request more than its rate of request, then such request packets would be dropped. The advantage of filtering the traffic early below the network stack saves lot of computing and memory resources on the server. On the application layer, the application server handles a request based on the request's priority level. The server can improve the response time to clients with higher response time priority levels by setting a higher priority for the threads that handle their requests. In general, the handling thread priority is proportional to the request's response time priority level. As congestion control examines clients requests on application layer, thus it can be said that it help in detection of application layer DDoS attacks.

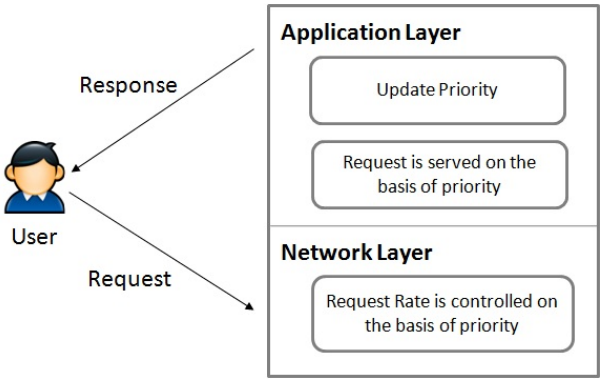

Figure 4.4: Two layered Defense for DDoS

4.4 Priority Calculation

In congestion control module, there is another check on already admitted clients through admission control. Congestion control module assigns priority to all clients and adaptively change their priority based on the browsing behavior of clients e.g. in case of e-commerce application, the ultimate objective is to predict clients intension for purchase. Now there are different attributes/parameters that can help us predict purchase for a client e.g. purchase history, average page visit time, if some certain brands are visited or if there is certain spending on average per order etc, then there is high chance

for a purchase. These factors can be vendor specific based on their experiences. Hence if there is a chance for purchase by a client, then its priority will be increased otherwise, if a client is only traversing the application without buying intension then its priority will be decreased. In this case, prior client is considered good and later one bad. Here our intension is to make more resources available to our good clients rather bad clients. More resources can be available to good clients in a way that they should be allowed to send requests at high rate to the server with less response time from server while bad clients get opposite to it i.e. low request rate and high response time from server end.

There can be an additional factor which can be used to set priority for a client i.e. weight of the resources accessed by client. Weight of resource means, how much valuable a particular resource is in the whole application. For example order confirmation page will have high weight because this page eventually leads to the purchase, while a search page has low weight because it put load on the server for fulfilling search queries, and at this point it is also not sure that client will make a purchase or not in current session. This weighting of resources is good option for static applications but for dynamic applications where resources add, delete or change on the go, this will not be practical approach.

Priority updating is not a one time process but server will keep on changing client's priority based on its browsing behavior. This is done after certain number of requests from that client. As the good clients gets high priority and bad clients the other way round, the penalty and rewarding factors are multiplicative and additive respectively i.e. additive increase priority for good clients and multiplicative decrease for bad clients. Changing priority on every request for every client is not feasible as it will create overhead on the server, thus degrade performance. New priority is sent to client in browser cookie as response. Priority calculation formula is

Good Clients Priority:
NewPriority = OldPriority + (α * OldPriority)
Where α is a constant value.

Bad Clients Priority:
NewPriority = OldPriority / (β * OldPriority)
Where β is a constant value.

Here value of α and β can be set by vendor based on how much s/he

wants to reward and punish clients.

4.5 Classification of Users

Consider an e-commerce application like amazon.com, the general walk through the e-commerce application follows as; first client browse product categories for product or search product, then on finding the desired product client add the product to cart, then place order by entering account information, then confirm order by providing payment and shipping information. We have characterized application clients into two types; one that follows application full walk through till the end i.e. make a purchase, are good clients while other category which surf the application without buying intension, these are bad clients. Good clients are potential customers, in principle they must be given preference over bad clients in term of resources.

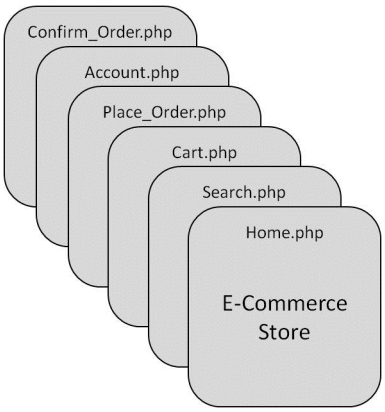

Figure 4.5: E-Commerce Application Walk Through

In implementation, we perform classification using decision tree algorithm J48. To classify a client in real-time environment, we first have to perform learning on some training data taken out from access log of legitimate clients in application server. Once learning has been performed, a classifier model has been generated. Then on real time, when a new request arrives from some client, it is first parsed and parameters are extracted from it, then these parameters are provided to the classification system which uses classifier model and client information as input and classifies the client either good or bad as a result. Then on the basis of this we increase or decrease priority of a client.

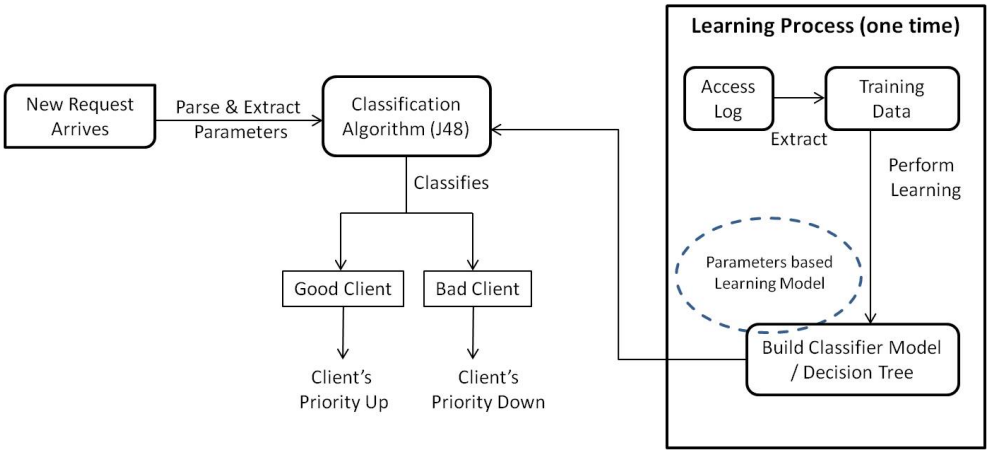

Figure 4.6: Clients Classification Mechanism

The classification of clients are totally based on the parameters we get during the session. Learning is performed on the pattern of these parameters and then classification is done using this learning as input together with new client session parameters. I have identified from literature, some general e-commerce application parameters and categorized into major categories. They are mentioned below.

To get more control on the whole client session, the classification of client must be calculated multiple times in a session; for example a clients visits an online bookstore, in start of session, we can have parameters like session ID, demographics, session depth etc from client's search results, these parameters help us initial classification of client. From now onward there are two scenarios, either client will quit the session or will continue to add product to cart. If session ends, then in our defense system it is classified as bad client, otherwise more parameters are collected with further clicks and after client place order, when we have parameters like average page visit time, session time elapsed, session clicks etc which shows how much serious a particular client is in making purchase, at this point in time, it is appropriate to perform classification again to validate our initial result. Here again there are two scenarios, either client quits the session or continues with providing account information. So on quitting, client will be classified as bad, in other case parameter collection process starts again. Then client enters account information and confirms the order by providing payment and shipping information. After this we have all information parameters of client like age, gender, purchase history, visits frequency, average order amount etc which are really useful parameter from the perspective of classification. So per-

Category	**Attributes**
Purchasing history	Recency of last Purchase Average difference in Purchase Recency Total No. of Purchases Total Spending on Site Average Purchases Per Visit Purchased at Last Visit
CPU Processing Time	Processing Time for Request Average Processing Time for Accessing Resource
Session Information	Session ID Customer ID Frequency of Visits Recency of Visit Average Time Between Visit Average time of Visit Average Clicks in a Session Average Time per Click
Resources Access Pattern	Page Access Pattern Total No. of Products Viewed Purchased Made or Not Ordering Cost of Current Order

Table 4.1: Deciding Attributes for Client's Classification

forming classification here again can really help in confirmation of our prior classification result. The reason for classification multiple times during the session is to ensure that Web or application server's resources are not wasted due to wrong intermediate classifications, because on cloud environment you are paying for every single resource you utilize, so only useful utilization of resources is desirable.

The mechanism for rewarding and penalizing is based on priority which in turn relies on the classification result. If a client is classified as bad then its priority will be decreased and if it is good, then it is the other way around. This priority has direct effect on the request rate at IP layer which is assigned to a particular client for future transactions. There can be a chance that in start of session client is classified as bad but then client made such moves which depict the good traits then it can be classified as good and will

be assigned high priority for rest of session. This was the motivation for classifying the clients multiple times during a session to ensure that we can track the client's intension during the whole session.

4.5.1 Classification Algorithms Tried

We have tried different classification algorithms on our data set. As we have labeled data set, so effectiveness of algorithm can be checked by error rate in classification i.e. lower the error rate, higher the effectiveness of algorithm. Among all the tried algorithms we found decision tree best of all due to its lowest error rate in classification. All the tried algorithms with their error rate are mentioned in table 4.2

Classification Algorithm	**Classification Error Rate**
Naive Bayes	12.02%
Decision Rule	4.44%
Decision Tree	1.37%
Nearest Neighbor	3.17%
Instance based Classifier	17.07%

Table 4.2: Tried Classification Algorithms

4.6 Tools and Technologies

4.6.1 Development Environment

- EDoS Armor is developed using JAVA 6.0 in IDE NetBeans 6.7 on Linux platform utilizing iptables firewall
- Sample E-commerce application is a JSP application developed in IDE Eclipse

4.6.2 Libraries Used

- HTTP Core API for implementation of HTTP request reply model for Web server
- WEKA API for classification of E-commerce application clients

Chapter 5

Testing and Evaluation

5.1 Testing

EDoS Armor needs to be tested for the evaluation purpose. The main objective was to test its behavior on allocation of E-commerce application resources among good and bad clients. For this, we need to simulate the good and bad clients' behaviors on E-commerce application to the EDoS Armor server. Bandwidth allocation, CPU processing time and response time allocation among good and bad clients are the means to evaluate the EDoS Armor effectiveness. Moreover, there is a need to test the efficiency of the EDoS Armor in term of CPU load, memory consumption.

To perform above mentioned tests, we used a sample JSP E-commerce application, and for automating its multiple sessions, we used Apache JMeter. JMeter works in this way that first we record a web application plan into it and then automate this through multiple threads on the target server. JMeter keep track of requests and their responses and gives results as a measure of user throughput and response time on the server.

5.1.1 Tools Used

- JMeter: For automating the clients behavior and measuring their throughput, response time generation by server.
- Bandwidthd: For measuring bandwidth allocation among clients at server.
- JConsole: For measuring EDoS Armor overhead cost in term of CPU load, memory utilization.

5.2 Evaluation

5.2.1 Data Set

For evaluation of our system, we have used KDD Cup 2000 data set (Discovery and Mining, 2000), which is an E-Commerce application Gazelle.com data set, containing the click stream and purchase transactions data. It contains 3 types of data sets in total. Each data set is derived by aggregating two un-aggregated data sets. Each data set is specific to the question/s asked in the KDD Cup competition, out of three data sets two are aggregated at session level (joined click stream data to session data) i.e. each record represents a single session, while third data set is aggregated at customer level i.e. each record represents a customer (joined click stream data to order line data). The numbers of attributes for first two aggregated data sets are 296 and 299 respectively, while third data set contains 518 attributes. Number of records for the first two data sets are same i.e. 234,954 and for third data set records are 1,781.

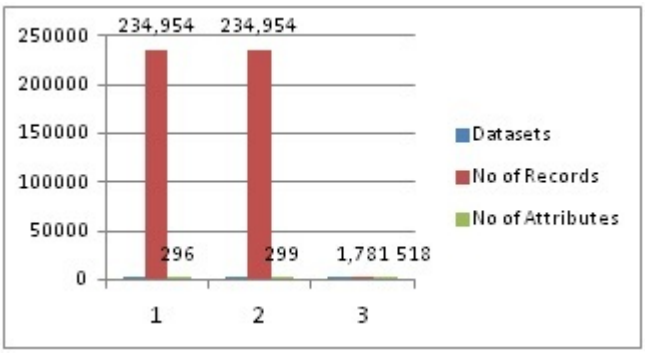

Figure 5.1: Data Set Details

In our case, we considered third data set containing orders information aggregated with customer click stream information. This data set contains the differentiating attributes like users who spend more than $12 on average per order or if a client visits expensive brand products and client has purchase history and session depth average etc which help us classifying a client as good or bad. The threshold values for attributes can vary from vendor to vendor; moreover attributes may also vary depending upon vendor's requirements.

We have a class attributes in each data set which already has classification result value. This attribute confirms the effectiveness of our classification

algorithm. It is desirable to match classification result to this class attribute with minimal error rate. The schema for our data set is as follow comprising some of major attributes.

| **...** | **Customer Session ID** | **Age** | **Gender** | **Purchase Count** |

| **Session Visits Count** | **Session Depth** | **Session Processing Time** |

| **Avg Order Amount** | **Avg page visit time** | **Session time elapsed** |

| **Session clicks** | **...** |

Table 5.1: Major Attributes from Data set

5.2.2 Test Bed

We have performed evaluation of DDoS Armor for resources distribution among good and bad clients. For this, we hosted a JSP e-commerce application on Apache Tomcat 7.0 Web server. This application is hosted on machine with hardware specifications Intel corei5 CPU 2.30 GHz processor with 4 GB memory and running Windows as operating system. DDoS Armor is deployed as a reverse proxy on separate machine in front of Web server with hardware specifications; 2.1 GHz Intel core2duo processor, 2 GB memory.

Good and bad client's requests are generated simultaneously from separate machines to monitor the behavior of our system in term of resources allocation and performance. We started our experiments with few numbers of sessions and gradually increased the sessions and closely observed the behavior of DDoS Armor in term of throughput, response time and bandwidth allocation. Below are the figures that depict resources allocation between good and bad clients.

5.2.3 Test Results

In our test, we bombarded DDoS Armor with up to 400 mix requests from good and bad clients and checked how our system behaved in term resources distribution among good and bad clients. We have analyzed DDoS Armor

on three major factors i.e. bandwidth utilization, throughput and response time for processing requests on server.

Figure 5.2 shows the bandwidth utilization behavior by good and bad clients during multiple sessions comprising varying number of requests. Ideally, good clients must utilize major portion of bandwidth than bad users to limit the wastage of bandwidth utilization by bad clients because on the Cloud, billing is done on quantitative utilization of these resources and one don't want to pay other than requirements. Here bandwidth difference is not so obvious when we have less number of requests but as we increased the number of requests, the bandwidth difference is majorally increased which is a good sign for effectiveness of DDoS Armor.

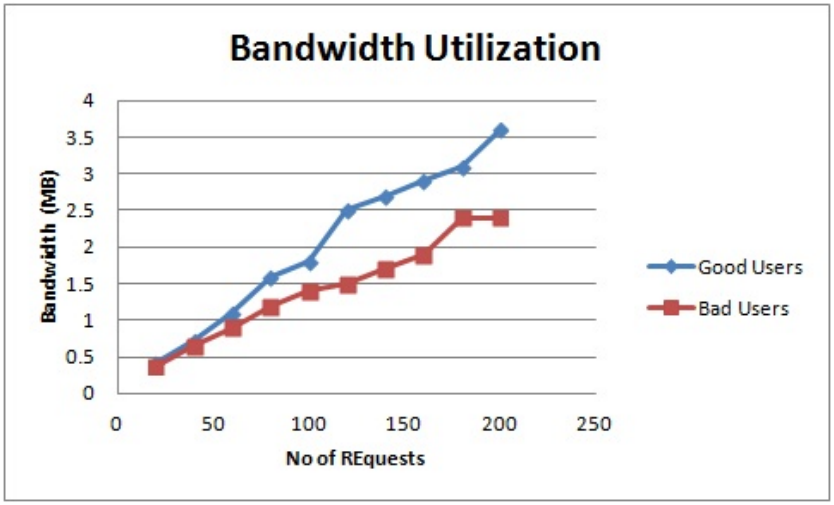

Figure 5.2: Bandwidth Allocation among Good and Bad Clients

Throughput is the measure of number of requests target system can handle per second. We have used throughput as a measure for identifying processing time occupied by good and bad clients during a session. Figure 5.3 shows that DDoS Armor responded well for both the type of clients requests (good and bad) i.e. bad clients in comparison to good clients got low throughput from start and difference remained constant till the end, which means preference is given to good clients requests for processing. We don't want to stop or drop the requests of bad clients but limit their CPU share on the server. Bad clients are actually non-valuable customers who never bought anything online and just visit e-commerce site for fun or window shopping.

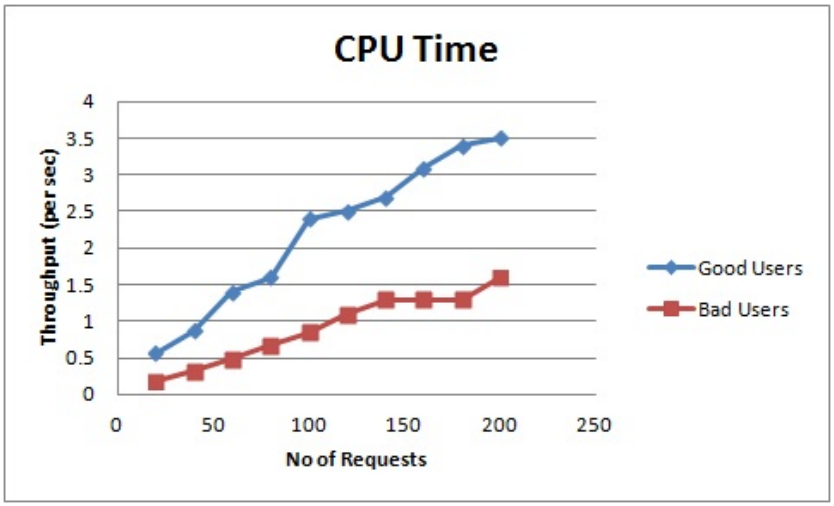

Figure 5.3: CPU Time Distribution among Good and Bad Clients

Next measuring factor is response time, which involves request processing time on the server. DDoS Armor has implemented threading concept together with fair queuing which made this functionality work.

Figure 5.4: Average Response Time Comparison for Good and Bad Clients

Ideally, bad clients must get higher response time than good users. Good clients are the potential customers, so they must not get service delay; on the other hand, if bad users get high response time, then they will not be bothered about this. Figure 12 showed the comparison of average response time allocation to good and bad users which clearly limit the response time

to below 3000ms for our good users, thus making DDoS Armor a practical solution.

5.2.4 EDoS Working Overhead

EDoS Armor has shown good results in perspective of resources distribution among good and bad clients, now there is a need to evaluate EDoS Armor in term of overhead. So EDoS Armor is evaluated in term of CPU, memory utilization and the results are shown in figures below.

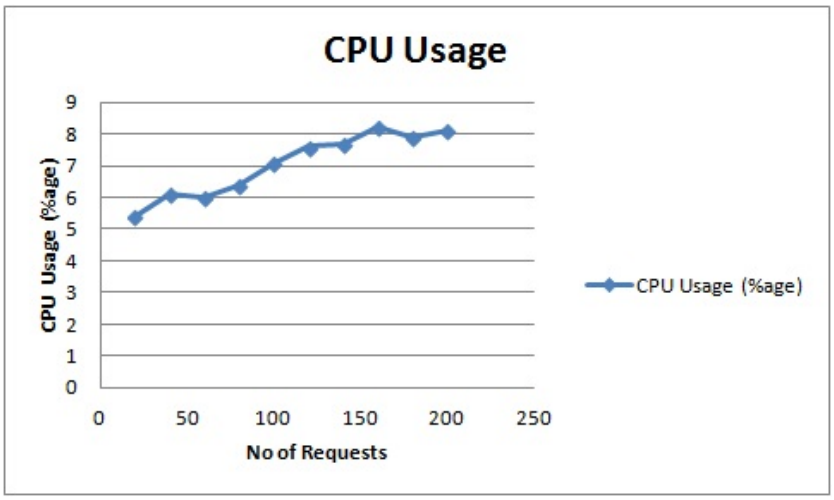

Figure 5.5: CPU Utilization by EDoS Armor

The above figure shows the CPU utilization by EDoS Armor. Here we can see that, EDoS Armor utilized only less than 9% for its working, thus there is no major overhead on the EDoS Armor server's CPU. This is the reason which makes this favorable for real world deployment.

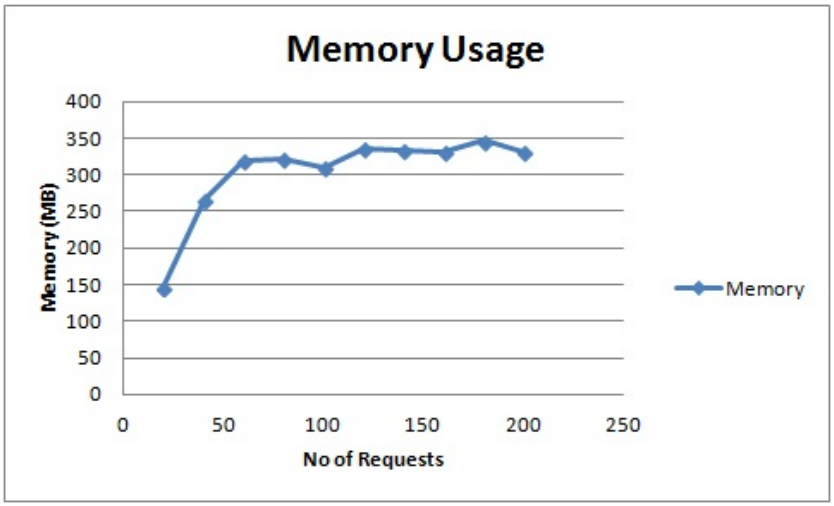

Figure 5.6: Memory Utilization by EDoS Armor

This is the memory utilization graph of EDoS Armor, which shows linear behavior in start, then as the number of requests increases, it become stable. This is again good behavior in term of memory utilization by EDoS Armor.

5.2.5 Comparison with IBM Technique

We have implemented IBM technique and tested it on the same test bed. Table 5.2 shows the tabular representation of comparison between EDoS Armor and IBM technique. EDoS Armor fairly has better features than the IBM technique, which proved EDoS Arbor as a major improvement over IBM technique, thus making it deployable and practical defense solution for EDoS attack on the Cloud.

Properties	**DDoS Armor**	**IBM Solution**
Classification of Users	Yes	No
Support Dynamic Web App.	Yes	No
Weights of Resources	No	Yes
Priority Calculation Verification	Yes	No
Human Intervention	No	Yes
Continuous Learning	Yes	No
Chance of Error	Very Low	High

Table 5.2: DDoS Armor vs. IBM Technique

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This research focuses on detection and mitigation of EDoS for E-Commerce based applications. EDoS is different from traditional DDoS in that, the intention of the latter is to consume all the resources (like memory, bandwidth, CPU etc) of the Web Server thus making it unavailable to its legitimate users. We have a twofold solution, admission control and congestion control. In the first, we limit number of clients that can simultaneously send requests, thus allowing only enough clients that can be served easily within available resources on the Web server. In the second, we change the priority of allowed clients based on the type of resources they visit and type of activities they perform, thus making the maximum resources available to good clients. Our contribution is two-fold: (1) We model the workflow of a typical E-Commerce application and identify key parameters that identify good and bad users and (2) we present the design of a learning-based classifier that distinguishes good and bad users depending on the values of the parameters they select while web browsing.

6.2 Future Work

The defense against DDoS attacks can not be foolproof, but it can be tried to minimize attack effect. In order to address this, we have recommended the following future research directions.

- Modification of current solution with respect to SaaS based Cloud architecture to deploy it in public Cloud.
- This work provides defense for only e-commerce applications in cloud

for now, it can be extended to include other DDoS target applications e.g. banking applications.

- Classification algorithm performance enhancement can be a direction to work on.

- Clients priority can be set based on the parametric value, which means now vendor can get more control on the clients. Vendor can define useful parameters with their weights in advance and priority updation can be done based on accumulative weight of parameters gathered from a client session e.g. clients having more useful parameter in session will automatically get high priority and vice versa.
  Some useful parameters with their respective weights are described below as a test case.

Parameters	**Weight (Total = 1.0)**
Purchase Count	0.4
Avg. Order Amount	0.2
Session Depth	0.15
Avg. page visit time	0.1
Transaction Time	0.075
Age	0.05
Gender	0.025

Table 6.1: Weight of Resources w.r.t their Usefulness in Classification

Bibliography

Ali, S. T. (2009). Throttling DDoS Attacks Using Integer Factorization and its Substantiation Using Enhanced Web Stress Tool.

CSA (2013). Cloud Security Alliance Top 9 Cloud Threats 2013. http://cloudtimes.org/2013/03/07/csa-report-top-nine-cloud-security-threats-in-2013/. [Online; accessed 15-May-2013].

Discovery, K. and Mining, D. (2000). KDD Cup 2000: Online Retailer Website Clickstream Analysis. http://www.kdd.org/kdd-cup-2000-online-retailer-website-clickstream-analysis. [Online; accessed 19-Sept-2012].

Durcekova, V. (2013). Novel Trends and Techniques Usable for Sophisticated Application Layer Denial of Service. pages 115–121.

Idziorek, J. and Tannian, M. (2011). Exploiting Cloud Utility Models for Profit and Ruin. *2011 IEEE 4th International Conference on Cloud Computing*, pages 33–40.

Idziorek, J., Tannian, M., and Jacobson, D. (2011). Detecting Fraudulent Use of Cloud Resources. *Proceedings of the 3rd ACM workshop on Cloud computing security workshop - CCSW '11*, page 61.

Jackson, W. (2011). Could a New Breed of DOS Attacks Make the Cloud Unaffordable? http://gcn.com/articles/2011/01/19/black-hat-cloud-dos-attacks.aspx. [Online; accessed 10-May-2013].

Jhansi, M., Radha, M., and Simmi, B. (2012). Mechanism to Mitigate Application Layer DDoS Attack with a Light Weight Trust Approach. *International Journal of Computer Science Informatics (IJCSI)*, (1):173–180.

Lee, S., Kim, G., and Kim, S. (2011). Sequence-Order-Independent Network Profiling for Detecting Application Layer DDoS Attacks. *EURASIP Journal on Wireless Communications and Networking*, 2011(1):50.

Moe, W. W. and Fader, P. S. (2004). Dynamic Conversion Behavior at E-Commerce Sites. *Management Science*, 50(3):326–335.

Patil, M. M. and Kulkarni, U. (2011). Mitigating App-DDoS Attacks on Web Servers. *International Journal of Computer Science and Telecommunications*, 5(2):13–18.

Prolexic (2013). Prolexic Quarterly Global DDoS Attack Report - Q1 2013.

Technical report.

Ranjan, S., Swaminathan, R., Uysal, M., Nucci, A., and Knightly, E. (2009). DDoS-shield: DDoS-Resilient Scheduling to Counter Application Layer Attacks. *IEEE/ACM Transactions on Networking (TON)*, 17(1):26–39.

Sqalli, M. H., Al-Haidari, F., and Salah, K. (2011). EDoS-Shield-A Two-Steps Mitigation Technique against EDoS Attacks in Cloud Computing. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 49–56. IEEE.

Srivatsa, M., Iyengar, A., Yin, J., and Liu, L. (2008). Mitigating Application-Level Denial of Service Attacks on Web Servers. *ACM Transactions on the Web*, 2(3):1–49.

Van den Poel, D. and Buckinx, W. (2005). Predicting Online-Purchasing Behaviour. *European Journal of Operational Research*, 166(2):557–575.

VivinSandar, S. and Shenai, S. (2012). Economic Denial of Sustainability (EDoS) in Cloud Services using HTTP and XML based DDoS Attacks. *International Journal of Computer Applications*, 41(20):11–16.

Wang, J., Yang, X., and Long, K. (2011). Web DDoS Detection Schemes Based on Measuring User's Access Behavior with Large Deviation. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–5. IEEE.

Wen, S., Jia, W., Zhou, W., Zhou, W., and Xu, C. (2010). CALD: Surviving Various Application-Layer DDoS Attacks That Mimic Flash Crowd. *2010 Fourth International Conference on Network and System Security*, pages 247–254.

Xie, Y. and Yu, S.-Z. (2009). Monitoring the Application-Layer DDoS Attacks for Popular Websites. *Networking, IEEE/AcM Transactions on*, 17(1):15–25.

Yu, J., Fang, C., Lu, L., and Li, Z. (2010). Mitigating Application Layer Distributed Denial of Service Attacks via Effective Trust Management. *IET Communications*, 4(16):1952.

Yu, J., Li, Z., Chen, H., and Chen, X. (2007). A Detection and Offense Mechanism to Defend Against Application Layer DDoS Attacks. *International Conference on Networking and Services (ICNS '07)*, pages 54–54.