A Cluster Based Fault Tolerant and Highly Available Architecture for Stateful Web Application Firewalls

By

Zafar Ali

2008-NUST-MS PhD-IT-32

Supervisor

Dr. Zahid Anwar

A thesis submitted in partial fulfillment of the requirements for the degree of

Masters of Science in Information Technology (MSIT)

In

NUST School of Electrical Engineering and Computer Science (SEECS),

National University of Sciences and Technology (NUST), Islamabad, Pakistan

CERTIFICATE

Certified that the Scrutinizing Committee has reviewed the final documentation of Mr. Zafar Ali Reg. no.2008-NUST-MS PhD-IT-32 student of MS-IT-9 thesis title A Cluster Based Fault Tolerant and Highly Available Architecture for Stateful Web Application Firewalls and found satisfactory as per NUST's standard format for Master Thesis.

President

WgCdr (R) Muhammad Ramzan

APPROVAL

It is certified that the contents and form of thesis entitled **"A Cluster Based Fault Tolerant and Highly Available Architecture for Stateful Web Application Firewalls"** submitted by **Zafar Ali** have been found satisfactory for the requirement of the degree.

Advisor: Dr. Zahid Anwar

Signature: _____

Date: _____

Committee Member: Dr. Hafiz Farooq Ahmed

Signature: _____

Date: _____

Committee Member: Dr.Raihan ur Rasool

Signature: _____

Date: _____

Committee Member: Mr. Aatif Kamal

Signature: _____

Date: _____

IN THE NAME OF ALMIGHTY ALLAH

THE MOST BENEFICENT AND THE MOST MERCIFUL

TO MY PARENTS, SISTERS AND A FRIEND

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 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 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: Zafar Ali

Signature: _____

ACKNOWLEDGEMENTS

First and foremost, I would like to extend my humble gratitude to Almighty Allah who always bestowed His blessings on me and gave me courage to accomplish this task. Darood-o- Salaam to Prophet Muhammad (P.B.U.H) chosen by Almighty Allah to guide the mankind to divine path.

I am truly indebted to my supervisor Dr. Zahid Anwar for his support, guidance and unending tolerance. He patiently spelled out all new concepts and completely guided me in all technical directions. I own that without the inspiring guidance of Dr. Zahid Anwar, this research would not have materialized. I extend my appreciation to my co-supervisor Dr. Hafiz Farooq Ahmad. Whenever there was a problem he steered me through. I am thankful to all of my committee members, for their guidance throughout the research work of this thesis. I really have no words of thanks for my friends and research fellows for their co-operation. I would always cherish the moments spent with them.

I can never forget the contribution of my parents in providing their all assistance to me. I owe all my achievements to my parents whose assistance and prayers enabled me to surpass all the hurdles in my life.

My Friend Neelofar provided me full support during the entire period of my research. Whenever I was downhearted, she encouraged and provided me with the power to overcome my hardships.

Zafar Ali

ABSTRACT

Application layer Firewalls are required to prevent attacks on the top three layers of OSI model i.e. session, presentation and application because traditional network firewalls do not understand attacks directed at disrupting HTTP(S) traffic. Web Application Firewall (WAF) architecture based on a single process does not provides fault tolerance and scalability because it doesn't utilizes the system resources completely. This thesis proposes a virtual cluster of WAFs in a reverse proxy configuration that aims to solve the problem of resource utilization and scalability by deploying it on multiple machines. The virtual cluster is incremental; it creates on-demand WAF nodes in presence of heavy load. Load balancing among virtual node is stateful to maintain session integrity and transparent fault tolerance mechanism to the users. Solutions for load balancing and fault tolerance exists at the network layer e.g. Heartbeat/Linux package, Ultra Monkey and LVS/Linux but these solutions are limited to application layer. We have implemented an application layer heart beat mechanism for transparent fail over that is adaptive and provides a sophisticated load balancing algorithm without an overhead of probe packets. The central component of our proposed architecture is the Dispatcher that receives HTTP traffic and distributes it among WAF nodes in a round robin fashion. The functionality of the Load balancer component is to provide intelligent routing decisions, handle cache, heart beat mechanisms and fail over. For evaluating the implemented solution the selected WAF product used is SWAF (A Semantic Based Web Application Firewall highly available). SWAF is a java based WAF consisting only of a single JVM process. Evaluation is performed by comparing performance results of SWAF addition or removal of our virtual cluster architecture with multiple SWAF nodes. Benchmark results show that load balancer with 8 and 12 SWAF nodes increased the performance(in terms of response time, data transferred in KB and error ratio) of the system significantly when the number of users are increased to tens of thousands in presence of session-based attack traffic and SWAF still performed detection correctly.

Table of Contents

List of Figures

List of Tables

CHAPTER # 01

Motivation &Introduction

1.1 Web Application Firewalls

A firewall restricts unauthorized access into a network. Firewall can be implemented through hardware, software or by combining both. Firewall doesn't permit unauthorized usage of private networks connected to internet, especially intranets. Some firewall techniques are packet filtering, circuit level gateway implementation and proxy server [1].

Web applications have grown the business nowadays. Ecommerce applications are usually web applications and contain major financial transactions. Security is high priority in these applications. Security is required not only at network layer but also at application layer a lot of work is done for the security at network layer and these security mechanisms are mature enough to prevent attacks through this layer. However, attackers have a fair chance of attack using application layer [2] [3] [4] [5]. According to an estimate from major cyber security organizations like MITRE, OWASP [3], WHITE HAT, ACUNETIX almost 75% of the attacks are launched at application layer now a days.

Web application firewalls are used to protect attacks at the application layer on http traffic or on web applications. They do not require modifications of application layer source code in most of the cases. These firewalls are deployed as reverse proxy as shown in figure 1. Due to inspection of heavy traffic, these firewalls cause a delay in response time depending on the complexity of the filter [6]. These firewalls are known as "Deep packet inspection firewall". Due to complex filtering mechanisms these firewalls introduce higher processing loads than routing [7] [8] [9]. For instance 300,000 packets per second have to be processed by a firewall that interconnects two 100MBPS networks [10]. Such a thorough filtering of traffic causes significant performance degradation [7] [11].

Application firewalls work by determining whether a process should accept the given connection. They accomplish their functionality by the connections between lower layers of the OSI model and the application layer by hooking into socket calls. These application layer firewalls are also called socket filters.

Application firewalls work in two modes i.e.: active or passive. Active application firewalls inspect all incoming requests actively against known vulnerabilities such as parameter and

cookies tampering, cross-site scripting and SQL injection. Only “clean” requests are passed to the application. Passive application firewalls acts like IDS (Intrusion Detection System). They also inspect the incoming requests for known vulnerabilities, but they do not deny or reject those requests if a potential attack is discovered [12].

Application layer firewalls improve the overall security of the application infrastructure by preventing attacks that are likely to cause a service outage or structural damage to data sources. Application layer firewalls are generally remotely updateable, which allows them to prevent newly discovered vulnerabilities. These firewalls are more advance as compared to adding specific security-focused code in applications, as it requires longer development and testing cycles.

Application firewalls improve overall security of infrastructure of an application by defending against attacks that can cause structural damage to data sources or service outage. These firewalls are generally remotely updateable and thus they prevent recently discovered vulnerabilities as well.

The motivation of this research is to design WAF architecture to prevent all the attack vulnerabilities mentioned above.

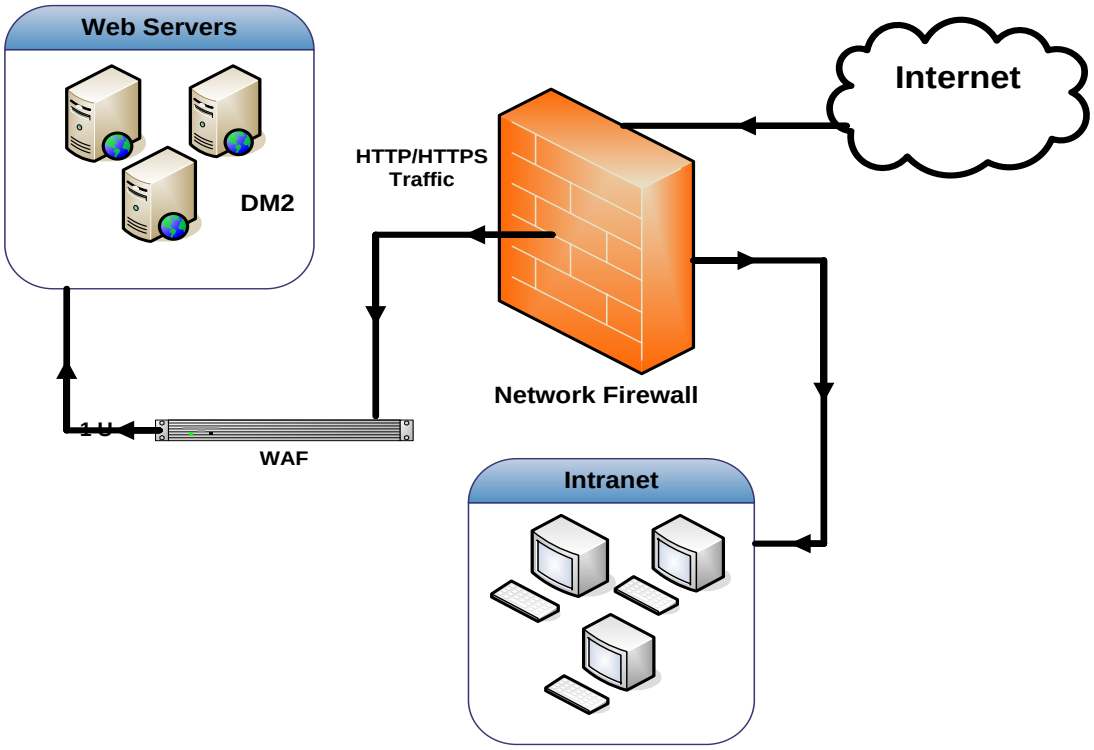

Figure 1: Deployment of Web Application Firewall as Reverse Proxy

1.2 Scalability

It's the application or system property to cope with increased amount of work in an expanded way in situations like increased demand for processing, network, file system resources or database access. Scalability is a major need of web-bases systems to be able to accommodate for more user requests in terms of complexity and number. Increasing the number of servers to cope with increased user requests is not an actual solution of scalability problem as it creates bottle neck at front end by moving it from back end. This risk even higher when put into consideration that web-based applications require that the front-end component can catch more information that exists at application level instead of TCP level.

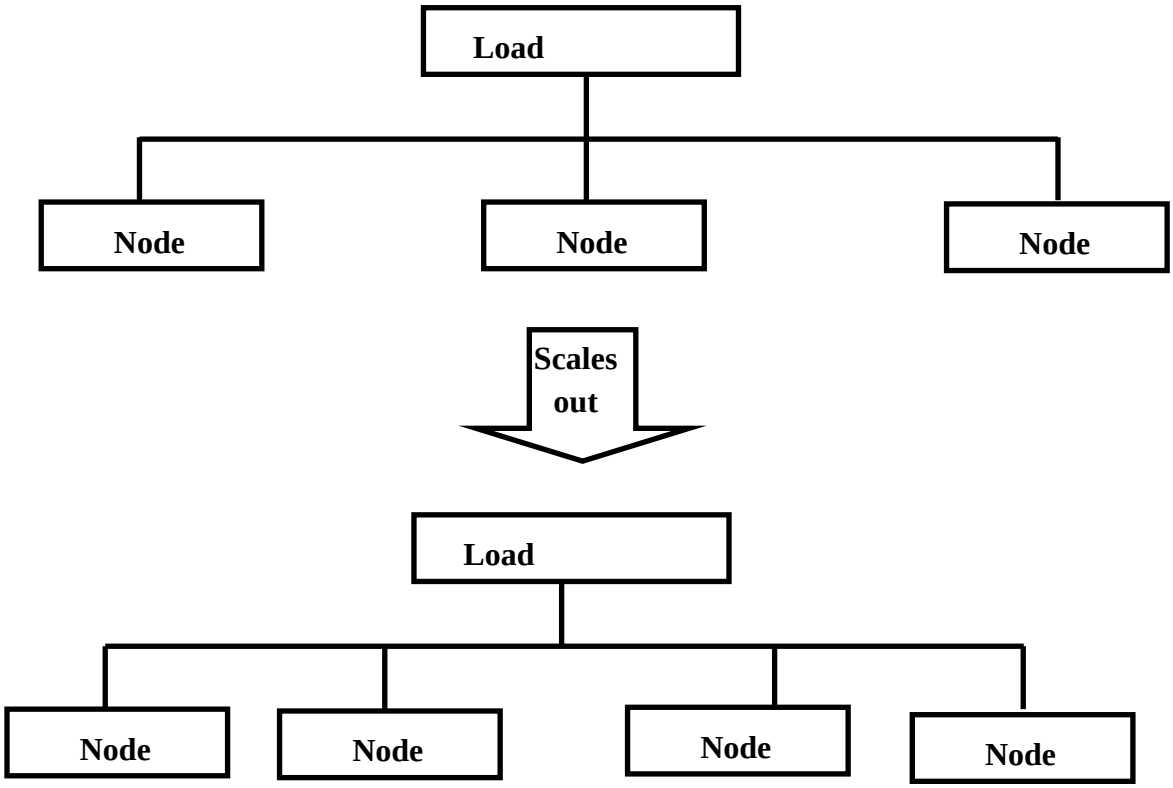

Figure 2: Horizontal Scalability-new node with identical functionality is added in a system to redistribute the load

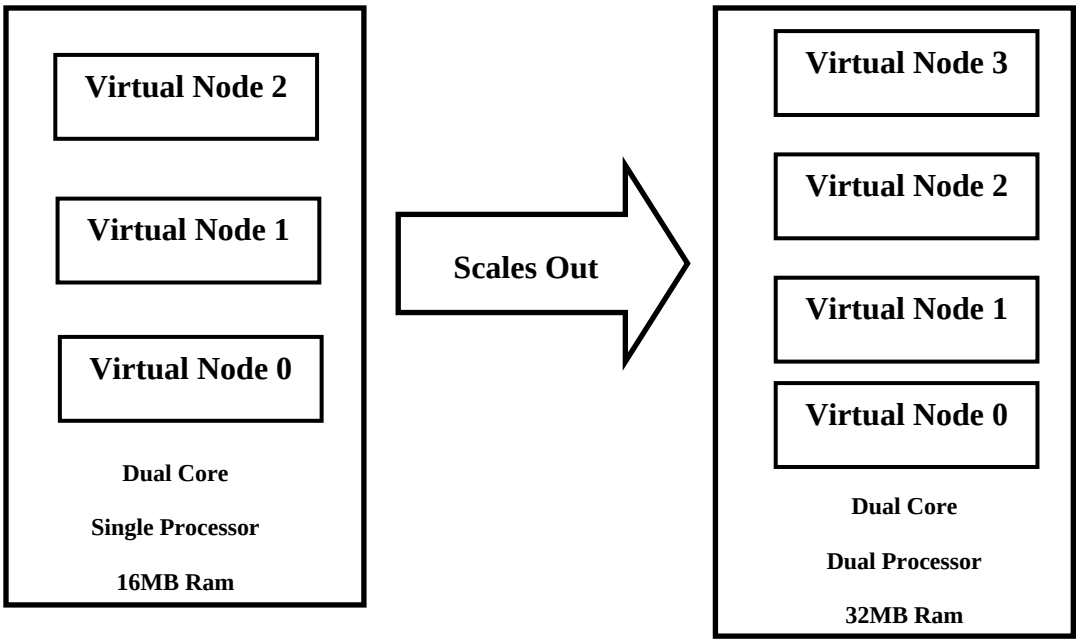

Figure 3: Vertical Scalability- adding more main memory, network interfaces to a node or processing to satisfy more requests.

There are two types of scalability horizontal and vertical. Horizontal scalability means adding new nodes with identical functionality in a system to redistribute the load. Web servers and SOA scale out by adding more servers to network which is load balanced so that incoming user requests many be distributed among them. Common term for this approach is cluster. Vertical scalability means expanding system by adding more main memory, network interfaces to a node or processing to satisfy more requests. Hosting services companies usually follow this approach by increasing the main memory and number of processors to host more virtual servers in the same hardware.

Single process based WAF is not suitable for large scale organizations due to heavy load. In this research we have proposed a more scalable WAF architecture suitable for heavy traffic.

1.3 High Availability

Availability is defined as the provision of useful resources by the system over a defined period of time. High availability is represented by high functional continuity with in a time window represented by the relationship between downtime and uptime.

$$A = 100\ (100*D/U)$$

In above equation U stands for uptime and D for downtime. U and D are represented in minutes. Availability and uptime is not the same thing. System may be up but unreachable and thus unavailable due to network issues. However unavailability and downtime are synonymous. Availability can be measured as the number of seconds or minutes of estimated downtime with respect to number of seconds or minutes in 525,600 minutes or 365 day year, with U as constant term.

WAF is deployed as reverse proxy. If it fails, all web servers become unavailable. The WAF architecture proposed in this thesis is highly available as if any WAF node fails or overloads, another WAF node takes its charge.

1.4 Cluster

Cluster is a group of computers which are connected in a manner that they work together in such way that for the users they appear to be a single system. They are used to improve availability, services, computational power or data manipulation performance. Clusters are more cost

effective than a single computer with the same computational power. Systems which are part of cluster are connected to each other over very high speed LAN like gigabit Ethernet, Myrinet, Infiniband or other technologies. WAF architecture proposed in this research is cluster based so provides scalability and high availability.

1.5 Load Balancing

Load balancing is used for maximizing throughput and minimizing response time by distributing requests among maximum available resources. There are two types of load balancing stateless and state-full. Scheduling algorithms are used to determine which user request is to be forwarded to which server. Web services and applications are mostly balanced by using round robin scheduling algorithms. Expiration algorithms and frequency rules are used to balance caching pools. Pseudo-random schedulers may use in applications where stateless requests arrive with uniform probability for any number of servers. Applications where some content is statically more popular, like music stores, may use asymmetric load balancers to forward popular requests to higher performance servers and rest of the requests to less powerful clusters or systems.

In state-full applications customer is guaranteed to maintain a session with a specific server in a pool. These applications require sticky or persistent load balancing. Figure 5 shows persistent balancer that maintains sessions from multiple clients.

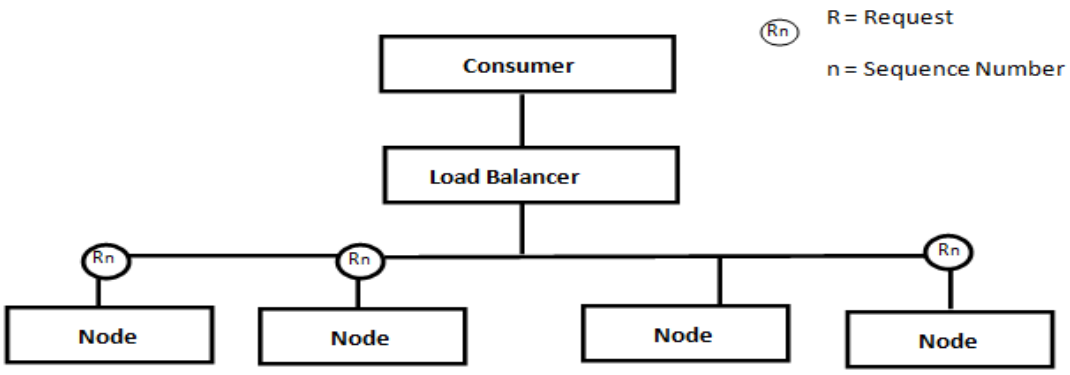

Figure 4: Stateless load balancing- scheduling algorithms are used to determine which user request is to be forwarded to which server.

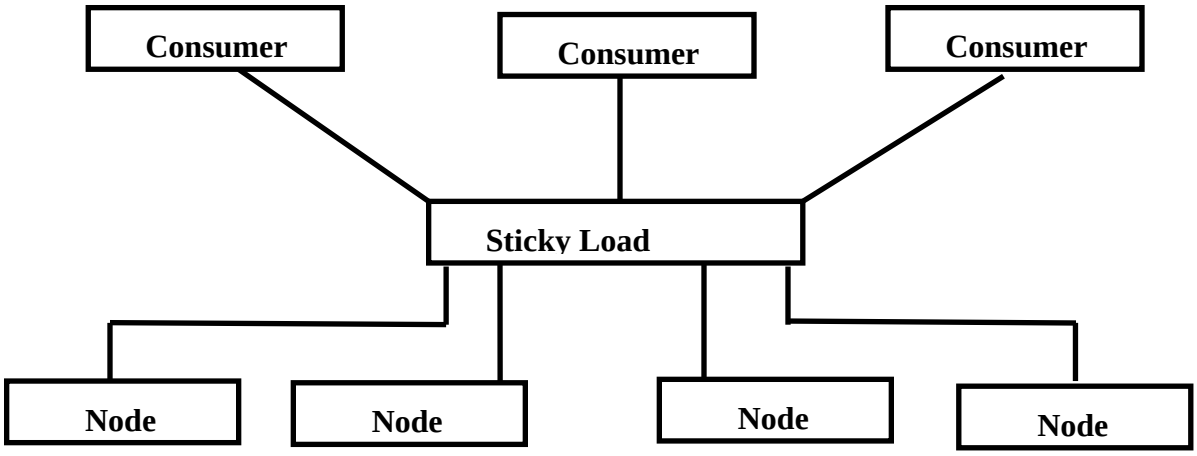

Figure 5: State full load balancing- customer is guaranteed to maintain a session with a specific server in a pool.

WAF slows down as the number of users increase from three thousand and is almost unavailable after seven to eight thousand users load. The WAF architecture proposed in this research automatically shifts the load to other WAF nodes if one node is overburdened.

1.6 Fault Tolerance

Redundancy in system design is based on assumption that failure of any system component is independent of failure of other components. Fault tolerant system remains available in case of failure of one or more components. Although overall efficiency and throughput decreases but still these systems remain available. Component redundancy is use to handle software or hardware faults. Fault tolerance requirements are taken from SLAs and their implementation depends on software or hardware components, and on their interaction rules.

As the WAF architecture proposed in this research uses multiple WAF nodes, system is fault tolerant along with highly available as failure of one node does not fail the overall system.

1.7 Motivation

Web application security is a vast area which is expanding day by day. WhiteHat Security's own research from weekly assessments of hundreds of the largest and most popular public-facing and pre-production websites confirms this fact: 9 out 10 websites have vulnerabilities [13].82% of websites have had at least one security issue, with 63% still having issues of high, critical or urgent severity [14]. Due to the increasing number of attacks on the web applications, prevention and detection of these attacks are very difficult at the application layer. There is a need for

common way of representing knowledge of web attacks which can help security community in detection and prevention of these attacks. Current architecture is suitable for SME (small and medium size enterprise), but has performance issues in large scale enterprises. Current architecture of WAF has the following limitations: It does not provide fault tolerance and scalability and it does not utilize the resources of the system completely being a single JVM process. Motivation is to create a scalable and fault tolerant system that would be suitable for large scale organizations.

1.8 Objective

Application layer Firewall is required because traditional network firewalls do not understand attacks directed at the code of the application using the normal channels through which the application is reached legitimately such as HTTP and HTTPS (SSL) for web applications. If the traditional firewall monitors TCP port 80, the traffic is allowed through; no matter what malicious code it may contain. So there is a need of application layer firewall which prevents attacks on the top 3 layers of OSI model i.e. session, presentation and application layer. Scalability and fault tolerance are desirable attributes for every system specially WAFs. The objective of this research is to introduce a scalable and transparent fault tolerance mechanism that is a need to design a distributed architecture for WAFs. The objective is how to introduce the scalability and transparent fault tolerance without causing any performance degradation in the system.

1.9 Thesis Organization

This thesis is ordered into seven different chapters. **Chapter 2** provides an extensive literature survey of the existing ontology in the web application security. **Chapter 3** presents the current system architecture. **Chapter 4** presents the proposed system architecture. **Chapter 5** presents the system design and Implementation. **Chapter 6** presents evaluation and testing. **Chapter 7** presents conclusion and future work.

CHAPTER # 02

Existing Work & Literature Survey

This chapter provides the background knowledge for this research. It gives a detailed overview of ontology that currently exists in the Web Application Security domain.

2.1 Resource Utilization in Java based Process

Each java program runs as a single process. It does not share memory with other processes. Each process is allocated with a memory called as heap memory with other processes. In normal circumstances JVM handles the heap by creating java heap for each JVM process. -Xms and -Xmx settings are used to configure the heap size. A JVM process can only get 2GB of memory if underlying OS is 32-bit at maximum [15], but actually this level has never been attained. According to various blogs only 1.5 GB can be achieved by a single JVM process, including Heap Size and others (others includes permanent space, code generation, socket buffers, thread stacks, directed memory space, JNI code, garbage collection, and JNI allocated memory) as shown in figure 6. Since the process of object creation leads to memory locking, hence on a multi-CPU machine (threads run concurrently) there can be contention of memory locking resulting in performance degradation [15]. If JVM have not enough memory for such operation, it is liable to crash. Although current architecture is suitable for SME (small and medium size enterprise), but has associated performance issues on large scale enterprises. Being a single JVM process, it does not utilize the resources of the system completely.

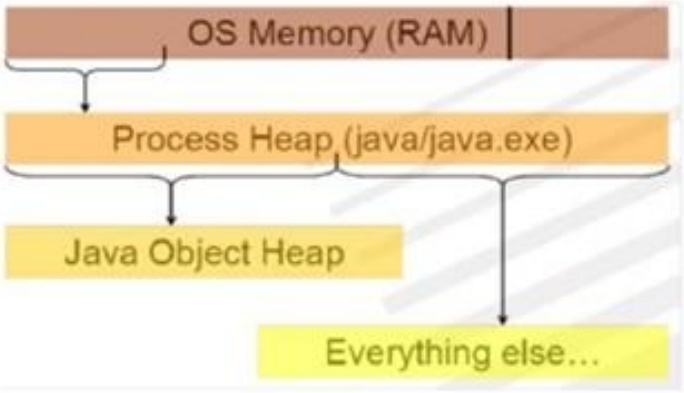

Figure 6: Memory allocation to a java process

2.2 URL Formalization

The purpose of URL formalization is to implement the URL table so that it can facilitate a fast lookup. Each node in the URL table is a variable length alphabet string. Therefore, tree-like [16] data structure is considered to be the most common solution to implement a data structure that is generally used for storing strings. Trio is based on the idea of tree structure in which each string is represented by a leaf and the value of the string corresponds to the path from the root of the tree to the leaf. But basic tree-like data structures demand large storage requirement. Moreover they need multiple (depend on the string length) costly memory access. If such string searching function is implemented in the layer-7 router that will be severe performance degradation. The primary task is to convert each variable-length string into fixed-length binary string by using a hash function. The binary string is stored in a LC-tree [17] [18], which is a level-compressed version of trio having the ability of efficient lookup [17]. When a packet conveying HTTP header arrives, the content aware routing mechanism retrieves the URL in the HTTP header. Then it uses the same hash function to convert the URL into the fixed length binary string. For example, a URL /entertainment/music/JAZZ/ has been converted to a string composed of 6e70, 4aTf and aTb3 (here, hex is used for convenient expression). In the URL table, a search for an entry is being made by the muting mechanism to get the longest match when compared to the binary string. This mechanism experiences the overhead of retrieving variable length string and name conversion. Since the HTTP header is composed of variable length strings, hence parsing the header to retrieve the necessary information for content-aware routing turns out to be a significant burden. For further accelerating the lookup in the URL table, a novel mechanism termed as URL Formalization is used. The problem can be solved by making every directory and file of the Web content as formalized expression. Hash function then is used to convert the original name of every directory and file into a fixed length and formatted name. Afterward, the html files and script files that generate dynamic content is parsed by another program. Then it modifies the embedded hyper-links to conform the new name. For example, if an embedded link points to the URL for example “http://pds.cse.nsysu.edu.tw/people/myluo”, should be converted to “http”//pds.cse.nsysu.edu.t /[[/4593/6827/”, where the name “people” and “myluo” are converted to a formalized name 4593 and 6827 respectively, and the “[!” is a preamble. The preamble is a “magic number”, which is designed to specify that the following path name is a formalized URL. This also implies that the name of the first level directory is the name of

preamble, and all the hosted content should be placed under this directory. The preamble design is of significant importance. It enables the routing mechanism to know whether the URL of a request is in normal form or formalized form. Since the operations of parsing and reconstructing the HTML files and scripts are pre-computed off-line hence they do not inflict any performance penalty on regular operations of the server system and the request routing mechanism. The reason of using a variable-length alphabet string to name a file or directory is because it is mnemonic, therefore becomes easier for humans to memorize. However, in most cases URLs are invisible to the users and they do not care about the name of URL. Ann HTTP request is issued when the browser follows a link: either explicitly, when the user clicks on an anchor, or implicitly, via an embedded image or object.

2.3 Content-Aware Routing

Numerous distributing mechanisms have been proposed over the past few years. These schemes can be classified into the following categories: client side approach [2] [3], DNS-based approach [4] [5], TCP connection routing [6] [7], HTTP redirection [8], and content-based routing [9] [10] [19]. Among these, Content based routing mechanism is the best choice to the Web hosting environment. The reason is that other schemes only can perform request routing based on some simple criterion, thus these simple routing schemes are no longer adequate. On the contrary, the content-aware routing mechanism can offer many potential benefits [19], such as sophisticated load balancing, QoS support, session integrity and flexibility in content deployment. The method of content-aware routing can be summarized as follows: a dispatcher node that executes the routing mechanism is responsible for pre-forking a number of persistent connections to the back-end nodes. The system resources are then allocated by dispatching client requests on these trunks. The client side browser first need to create TCP connection, as a client tries to retrieve specific content. The incoming TCP connection requests are acknowledged and handled at the dispatcher until the client sends packets conveying the HTTP request, which contains the URL and other HTTP client header information. A decision, regarding how to route the request is made by the dispatcher by looking into the HTTP header. The dispatcher then opts for a server that is best suitable to this request. It then chooses and idle pre-forked connection from the available connection list of the target server. The related information about the selected connection in an internal data structure termed as “mapping table” i.e. binding the user

connection to the pre-forked connection is then stored by the dispatcher. Once the connection binding is established, the dispatcher handles the consequent packets by changing each packet's IP and TCP headers. The packet between the user connection and the pre-forked connection are seamlessly relayed so that the client and the server can transparently receive and recognize the packets. In case of server overload or server failure, the request can be migrated to another node. To ensure high reliability the server cluster should comprise on two important capabilities: checking and failover. That is, some intermediate state of user requests should be logged periodically by failover mechanism. It should enable the ongoing requests on the failed node to be continued processing with a valid intermediate state in another working node. Both these techniques have been scrutinized and are well-known in the research area of fault tolerance, but implementing these techniques in the distributed web server still causes many novel challenges. It is quite expensive to log every incoming request for check pointing. In a Web hosting system, all contents are not equally important to the client and the service provider because of their importance or cost, some of the hosting contents cannot tolerate service disruptions. The distributing mechanism can differentiate the important requests (e.g. requests for mission critical services or requests for content owned by important customer(s) from regular web surfing requests with the content-aware routing capability. Moreover to recover a web request of failed server to continue execution in another working node is a challenge. Such recovery mechanism should be user-transparent and smooth.

2.4 Content Aware Distribution vs. NAT Base Routing

For routing request to individual servers within cluster, the existing NAT base approaches have usually emphasized only on user transparency, load distribution and scalability. While content aware distribution take into account other issues as well and are integrated in content aware distribution.

2.4.1 Session Integrity

Since the HTTP protocol is stateless i.e. Web server executes each request independently without relating that request to pervious or subsequent requests. However in many cases maintaining state information is of vital importance for example, such state might contain the contents of an electronic "shopping cart" (a purchase list in a shopping mall site) or list of results from a search request. When the user visits a shop, or asks for the next 25 items from a search, the state

information from the previous request is required. Cookies or hidden variables within html forms are among the schemes that have been employed to maintain state information. In a cluster-based server, these methods might not be processed appropriately. In selecting a server, if the routing mechanism doesn't examine the content of each request then it can be possible that a request belonging to session is dispatched to the wrong server because the state concept is an increasingly critical part of web behavior for commerce, web-oriented database, and other dynamic transaction applications; this might limit the usefulness of cluster architecture.

2.4.2 Sophisticated Load Balancing

To utilize the cluster resources evenly and efficiently, a server cluster requires some sort of load-balancing mechanism for directing requests. In existing web sites, the service type of incoming requests can be of various types as static web pages, dynamic content generated by CGI scripts, or multimedia data such as streaming audio or video. Each request consuming the service time and the amount of resources vary widely and depends on several other factors for example a request for executing a CGI script normally requires a great deal of computing resources to static file retrieval requests. This heterogeneity in request often results in skewed utilization of server cluster consequently, a more sophisticated load-balancing mechanism based on the service type of each request is essential. In many existing systems the load balancing capability is still limited because they don't consider service type of each request.

2.4.3 Differentiated Services

The web persists to develop from its preliminary role as a provider of read only access to static documentation-based information, and is becoming a platform for supporting complex services. However, most current web servers, both cluster-based and monolithic provide service in a best-effort manner that does not distinguish between the requirements of different requests. This approach does not work well. Different services may have different requirements for quality of service. It is difficult to enforce priority policies and provide desired quality of service if the routing schemes don't match the server type of each request. Otherwise, not all content are equally important to the client and server provider. However, requests for popular pages have the tendency to overcome the requests for other critical pages such as product list or shopping-related pages. Consequently, enterprisers and service providers want to exert explicit control over resource-consumption policies in order to provide differentiate quality of services due to the variety of content.

2.4.4 URL formalization and context aware routing

Mon-Yen Luo and chu-Sing Yang discussed the architecture and some key mechanisms of an integrated framework for providing reliable and highly manageable web hosting service on a scalable server clusters system [20]. A novel idea of URL formalization and corresponding data structure has been proposed by them for load balancing and fault tolerance, they had used a content aware routing algorithm. Content aware routing algorithm takes decision on the basis of URL table where URL table contains tree model of web application, pages content size, priority and processing nodes. By using hash table or hash tree, a URL table can be implemented. On base of parent URL, they have provided decision specific to URL as well as wild card. By introducing the cache in the system, the performance of routing decision can be improved. A heart beat mechanism is used to monitor the reliability of a node which is based on the adoptable beat rate depending on the load on that node. Due to that dispatcher it is able to recognize the load on the particular node. Two kinds of values are being sent in a probe packet one is warning value that is sent when the node is loaded the other value is dead value which is assumed when packet is not received to the dispatcher after the specified time interval. They have provided a system claiming no performance degradation; providing intelligent heart beat system and transparent fault tolerant mechanism but the drawback of the system is that it is application specific solution and it needs lot of details from the application.

2.4.5 Highly available clusters

Pablo Neira Ayuso, Rafael M. Gasca and Laurent Lefèvre discussed the three major issues about the firewall architecture in their research [21]. Among these issues, the first issue one is the performance issue which causes reduction in the bandwidth and throughput of the network. Second issue is availability and the third issue is complexity. According to them, performance is the principal issue because if that is not determined the firewall may turn out to be bottleneck in the system. The four different architectures of firewalls conferred by the authors are as follows:

- The Primary-Backup approach
- Multi-Primary Multipath Firewall Cluster
- Multi-Primary Firewall Cluster Sandwich
- Multi-Primary Hash-Based State full Firewall Cluster

Some of these architectures are applicable to stateful firewalls while the rest are applicable to stateless firewalls. The most effective architecture also relevant to WAF is ’Multi-Primary Hash-

Based State full Firewall Cluster' because this architecture is for state full firewalls like WAF.

2.4.6 High performance load balancing algorithm in Jcluster

Bao-Yin Zhang1, Ze-YaoMo1, Guang-Wen Yang2 and Wei-Min Zheng illustrated the API provided for the dynamic load balancing and high performance communication named Jcluster in their research [22]. For efficient load balancing, Jcluster is considered to be an efficient Java parallel environment. They designed a task scheduler which is responsible for scheduling the task between the nodes. If any node is becoming stale, it aids that node to steal a task from the busy node. The task scheduler is based on a Transitive Random Stealing algorithm. They have also improved the random stealing algorithm which was based on selecting the task randomly if the working queue of that node is empty. By introducing the transitive policy, they have improved the RS algorithm. In their scheme any idle node can obtain a task from another node with much fewer stealing times on a large-scale cluster. This significantly reduces the idle time for all nodes and also the network communication overhead and ultimately contributes in enhancing the scalable performance of the system. The system is implemented in Java pure Java implementation of the system makes it suitable for heterogeneous clusters.

CHAPTER # 03

Current System Architecture

3.1 SWAF

As discussed previously (section 1.3) a brief about Web Application Firewalls has been elaborated. When Network Firewalls came up with maturity to handle attacks on network layer, a great shift of attacks were cited when it focused on application layer too. Until recently a great volume of research has been dedicated to handle attacks on this layer including the current developed product. For real time results and effectiveness it is tested with an already existing firewall known as SWAF (Semantic Based Web Application Firewall).

The current system of SWAF is multi-threaded single JVM process. All components discussed later are the part of the single JVM process. The system has the following features:

- The system is based on some effective semantic technique that enables system to understand the context of user input.
- The system generates attack rules automatically.
- The attack rules contain the general representation of attack.
- The system minimizes the human intervention by making most of the task automatic and provides user friendly interface to the administrator.

Application level attacks can only be catered at the application layer. Security mechanism on the application layer means that it knows the context of the application layer. Therefore to capture the context of application layer some semantic technique is required. In a Web application with TCP/IP suite, the application context refers to the deployed application and the underlying application layer protocol.

3.2 Ontology for designing of Context

The proposed solution contains the context of underlying protocol i.e., HTTP, general web application working, and the application level attacks. In addition, it also contains the content structure i.e. HTML. Ontology has been used for engineering the required knowledge. The ontological representation enables the system to reason over the context supplied and generates more assertions. An explicit specification for a conceptualization is termed as Ontology [23]. To provide a formal specification of the concepts and relationships that can exists between entities within a domain, IMP Ontology is used. It offers powerful constructs that include machine interpretable definitions of the concepts and the relations between them. Ontology provides the significant and sufficient constructs that are required to enable a software system to reason over

an instance of the entity within domain [23]. An ontology representation language is used to design and implement a data model, depicting the domain of application layer attacks and intrusions. Ontology facilitates powerful constructs that expresses concepts as machine interpretable definitions and their relations within the domain. This feature enables different software systems (most probably deployed in heterogeneous environment) to share a common understanding of information under consideration. Moreover, it allows the software systems with an immense ability to reason over and analyze this information. Accordingly, ontology is designed to enable knowledge sharing and reuse between the entities within a domain [24]. Ontology representation languages may be represented into first-order relational sentences and a set of first-order logic axioms. This mapping doesn't allow the interpretations of the non-logical symbols [25], which enable the instances of the ontology to be operated over using formal and complete theorem proves.

3.3 Rule Based Reasoning

Similarly, in order to manipulate ontology rules are required. Rules also enable rule based reasoning on the layer above the ontology. Hence Knowledge base contains the ontology and rules. The information required to create attack representation automatically is enclosed in the inferred result. This functionality is required to query the inferred model to construct attack representation using that inferred model. By using rule based reasoning over knowledge base, inferred model is constructed. This model is then used to generate attack detection rules.

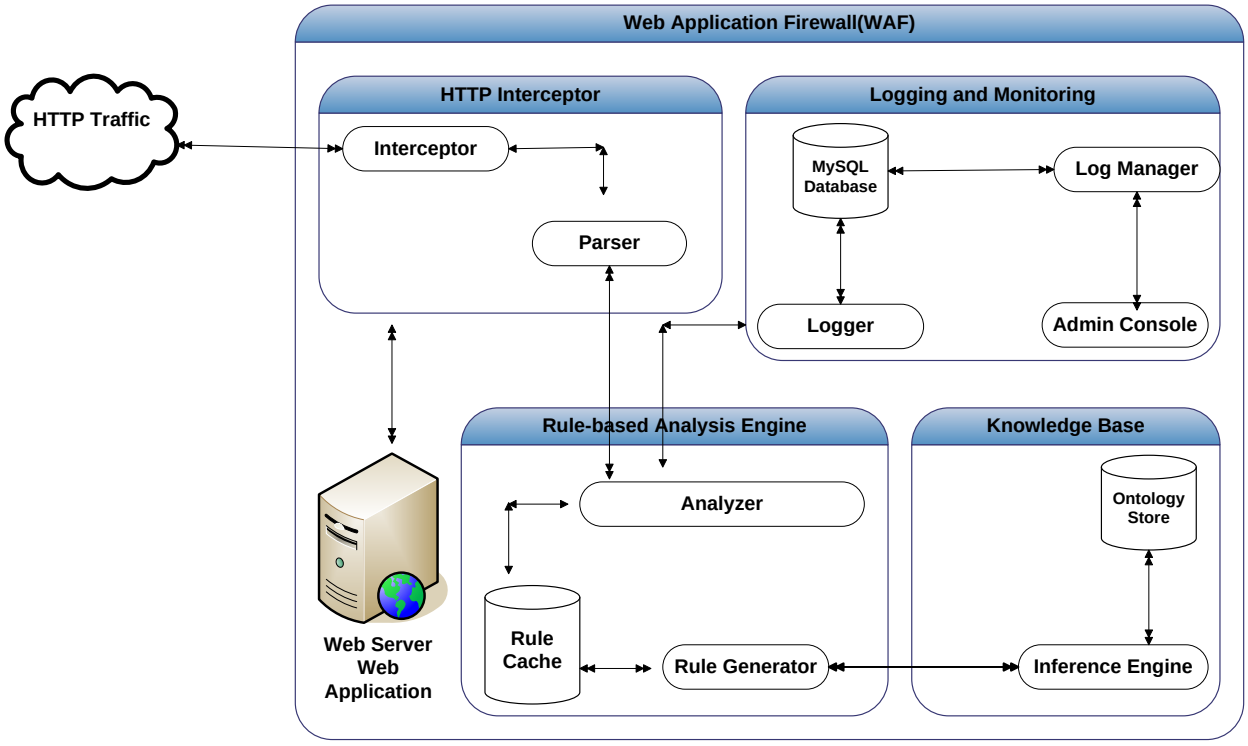

Figure 7: Current System Architecture- the System is designed as a surrogate proxy/reverse proxy that is deployed in front of web server intercepting all incoming and outgoing requests to and from the web server.

3.4 System Architecture

The System is designed as a surrogate proxy/reverse proxy that is deployed in front of web server intercepting all incoming and outgoing requests to and from the web server. The system has the capability to shield all attacks to web application. The architecture of the proposed system is shown in figure 7. The main components include HTTP Extractor, HTTP Parser, Packet Analyzer, Rule Engine, Ontology Manager, Inference Engine, Logging Module, and Knowledge Base. The different components are summarized as follows:

3.4.1 Ontology Manager

For querying of knowledge base, the interface is provided by Ontology manager. A popular Jena API is contained in it. This free and open source API is developed by IBM. For manipulating Ontology, Jena API provides an extensive set of API.

3.4.2 HTTP Extractor and Parser

HTTP Extractor is accountable for extracting HTTP messages from raw network stream. The

extracted messages are then sent to the HTTP Parser. According to the standardized message format, the Parser parses the HTTP message. The convention of HTTP 1.1 has been followed by us in parsing out the HTTP message. It creates HTTP parsed message object and then send it to analyzer.

3.4.3 Analyzer

On the basis of semantic rule object, analyzer performs analysis on the received HTTP parsed message object. This module is accountable for detecting intrusion, generates alerts, directs logging module to log single message or whole session. The actions such as packet drop, session tear-down, redirecting to the logging module are also carried out by it.

3.4.4 Rule Engine

For creating semantic based rule object, rule engine is used. The rule engine is used to define the principle of governing, conduct or to guide behavior actions. In SIDS, on the basis of the knowledge stored in the knowledge base, rules are created automatically. Rules are generated by the Rule engine after querying the inferred model. Rule object consist of two parts.

3.4.4.1 Filter/ Indicator

The attacks at specific portion in the HTTP Message are detected by Indicators. Regular expression is used to build indicators.

3.4.4.2 Rule action

The actions that should be executed when the corresponding indicator is found or matched, is contained in rule action. Alert, Log, Discard, Allow and Ignore are the examples of rule action.

3.4.5 Admin Console

It facilitates the administrator by providing an interface. Through the interface, the admin can either configure the system or can write commands for further assistance of administrator alerts are also displayed on admin console.

3.4.6 Logging Module

It provides the facility of logging messages or whole session.

3.4.7 Cache Controller

Cache Controller is responsible for managing cache content. For additional inspection, it temporarily caches the message or session. Moreover, it provides faster mechanism for sending message to the web application. One copy of the message is send to the cache while the other is

send to the Parser by HTTP extractor. Message is from the cache to the web application, if it is legitimate. It obviously saves time of rebuilding the message from parsed message object.

3.5 Session Based Attacks

SWAF not only detects the request base attacks, but also keeps track of attacks that may occur not based on single request. The attacks that can be launched using more than one request are called session base attacks for these types of attacks SWAF maintain state regarding client traffic and session. Following are the types of attacks currently handled by SWAF

3.5.1 Session Fixation

System vulnerability can be exploited through session fixation attack which is possible to fixate some person's session identifier. Most of these attacks are web based, and rely mostly on acceptance of session identifiers from POST data or URLs (query string). For fixing this attack SWAF uses following prevention mechanism. SWAFTOKENID is generated and associated with each client's session. SWAFTOKENID is attached with each request in that client session is either in query string or in post data parameter. All SWAFTOKENIDs are store in memory table in SWAF. Each SWAFTOKENID is associated with cookies and client IP. One SWAFTOKENID can only be assigned to a single client and if same SWAFTOKENID is received from different client IP, SWAF considers it as session fixation attack.

3.5.2 Session Hijacking

Session hijacking is a valid computer session exploitation to gain unauthorized access to services or information in a computer system. The attack is basically launched using the cookies use to authenticate a user to remote server. This attack is particularly relevant to web developers, as the HTTP cookies which are used to maintain session on websites can easily be stolen using an intermediary computer or by accessing the saved cookies on victim's computer. SWAF handles this attack in a way similar to session fixation. Same SWAFTOKENID and memory table is used to prevent this attack and cross site scripts (XSS) are disabled as well.

3.5.3 CSRF

CSRF is abbreviation of Cross-site forgery. It is also known as session riding or one click attack. In this attack unauthorized commands are transmitted from a user that the website trusts. In contrast to XSS (Cross Site Scripting) which exploits the user using a site user trusts, CSRF exploits the trust that site has in some user's browser. SWAFTOKENID is also used to prevent

CSRF attacks. All responses have some links. SWAF attaches a SWAFTOKENID with each link before forwarding it to client. When any request sends through this link from client, SWAFTOKENID in request query string and at SWAF server is matched. If both SWAFTOKENID are same, the request is legitimate else considered as CSRF attack.

3.5.4 DOS (Denial of services)

Using denial of service attack, an attacker makes a network resource or computer unavailable to its intended users. Although motives for, targets of and means to carry out this attack may vary, but it is actually an effort to prevent a site or service from functioning efficiently either indefinitely or temporarily. For preventing DOS attack IP and IP/resource tables are used for SWAF. Information recorded in IP table is client IP, total no of requests sent/second and information recorded in IP/resource table is client IP, resource URI and total no of requests per second. If any client IP exceeds limit from any table, its request will be blocked, thus preventing DOS attacks.

3.6 Limitations of current system architecture

Java based technology is used to implement the current system. It works as a single JVM process. There is an obvious limitation of being single process (JVM process). Current architecture is suitable for SME (Small and Medium size Enterprise), but has performance issues on large scale enterprises. Current architecture of WAF has the following limitations:

- It does not provide fault tolerance and scalability.
- Being a single JVM process, it does not utilize the resources of the system completely.

This architecture works as the single JVM process hence limited to utilize the only resources provided to a single process by operating system. Even though the system is multi-threaded but still it is treated as single process by operating system. Because of tight mapping of different components with each other, current architecture does not support scalability. Depending upon user load, other instance can be created. Using only a single thread to intercept all requests will result in creating a bottle neck in system hence failure of any component can cause the whole system to crash.

The weaknesses of current solution are listed as under:

- It works as a single JVM process
- It is not scalable for large scale organizations.

- In case of failure, no fault tolerant mechanism is available.

3.7 Summary

This chapter scrutinizes the idea for an effective application level security mechanism. Most of the issues in low traffic server solutions have been the major concern of current system. The present high level architecture for semantic based application level intrusion detection system has also been presented by the current system besides, various components in the current system architecture has been highlighted.

CHAPTER # 04

Proposed System Architecture

This chapter discusses proposed architecture detail. It states that we can overcome the limitations of the current system and the mechanism to over comes those limitations Implementation of those components is discussed in detail in next chapter. This chapter explains high level overview of the components of the proposed architecture. It also describes high level architecture diagram of our proposed system shown in Figure8.

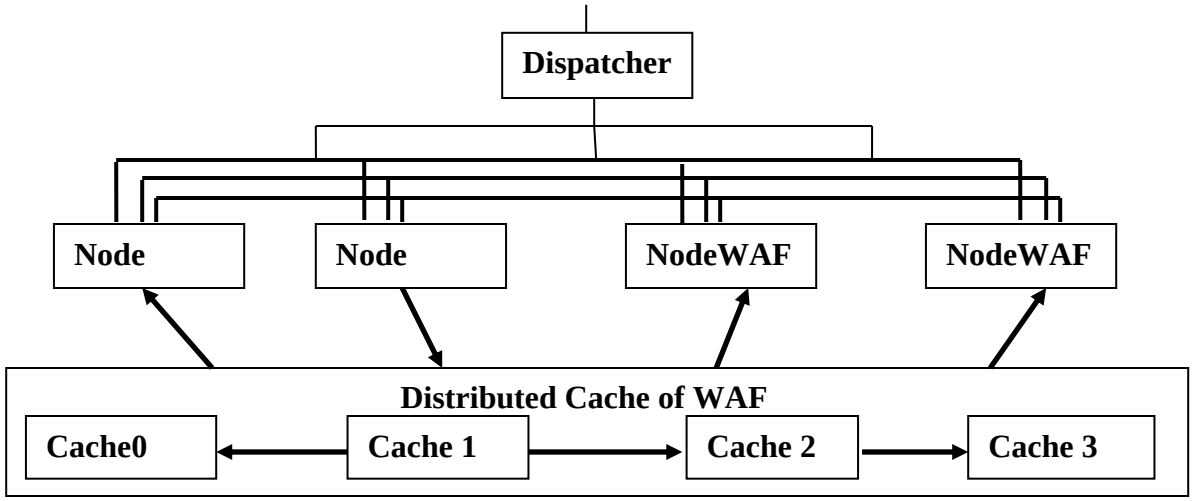

Figure 8: High level architecture diagram of proposed Architecture- for load balancing a very thin dispatcher is used which takes very little time to load balance

The major problem in current architecture is that it is creating a bottleneck for overall system. The main reason is discussed in previous chapter was the deep inspection of the contents as HTTP request and response both are inspected so it takes lot of time in over all request response cycle. In order to avoid this delay more than one WAF can be used to divide the load. This approach also works better in the case of large number of users. For load balancing a very thin dispatcher is used which takes small amount of time for load balancing Due to multiple instances of WAF in proposed architecture not only load balancing but fault tolerance is achievable. In case of a failure of one WAF node all its traffic can be redirected to other WAFs. By making the fault tolerance mechanism transparent, the request can be cached at dispatcher till the response receives and state of WAF is maintained at central point so that all WAFs can access that information.

4.1 Dispatcher

Dispatcher is introduced in front of WAF in proposed architecture to make the architecture distributed. First goal is to design the dispatcher in such a way that it should take minimum amount of time to parse the request, to take decision in selection of WAF node for request and to send back response to the user. Similarly transparent fault tolerance can be achieved in case of failure of any WAF node, and dispatcher stores any necessary information regarding that purpose. Following are the major components of dispatcher as shown in figure 9

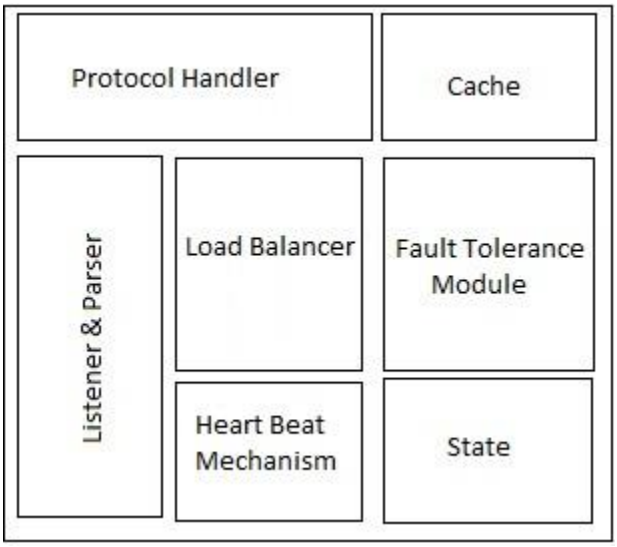

Figure 9: High level architecture diagram of dispatcher- major components are protocol handler, cache, listener and parser, heart beat mechanism, fault tolerance module and load balancer.

4.1.1 Protocol Handler

This module is used to define the implementation of protocols to be handled by the dispatcher and create listener for that protocol. Dispatcher is not protocol specific; it can be used for any kind of protocol provided that handling mechanism of the protocol is defined in dispatcher. Any Protocol can be added to this module by providing details of that protocol.

4.1.2 Listener and Parser

This module is used to create listener for the protocols specified in the protocol handler. Listener is responsible to open a port (either the default port or configurable port) defined for that protocol. Similarly parser is responsible to parse the incoming traffic in required format. The

parser is again protocol specific; for each protocol its parser should be written. In proposed architecture main goal is to create a very light weight parser. The parser for HTTP protocol is written in such a way that it takes minimum time and gets only required information in formatted output. For now it is only getting request line of HTTP protocol and session information and client information is being used from listener part of the module.

4.1.3 Load Balancer

This module is used to distribute the load among the available WAF nodes. The load balancer is using weighted round robin algorithm for this purpose. Each WAF node has been assigned a weight on the basis of resources available on that node. By the time the weight of the node is being incremented on the basis of the traffic load assigned to that node. Load balancer selects next WAF node on the basis of load balancing algorithm defined in section 4.1.2. Stateful load balancer is used in the dispatcher which keeps track of client state. If any WAF node is assigned to a particular client same WAF node will be used for that client in that particular session until and unless that node is not failed.

4.1.4 Cache

This module caches the client request until its response is received from the WAF node. This module is used to hold client request information for short period of time so that if request sent to the WAF node get failed due to failure of WAF node that request can be send again using other WAF node and provide transparent fault tolerance to the user.

4.1.5 State

This module is used to hold the states of client. The state information includes the client IP, session ID and the WAF node assigned. This information is useful for stateful load balancing. As dispatcher is using stateful load balancing it gets client information from this module and assigns WAF node accordingly.

4.1.6 Fault Tolerance Module

This module handles any failure in WAF node. This module uses information from cache module and state module. When any failure detected by heart beat system, it sends its

information to fault tolerance module which in turn gets that request information from cache module. System gets next WAF node with the help of load balancer module and send request to the newly selected WAF node with failed WAF node information. This information is being sent using special HTTP header. When any WAF node reads that header, it reloads the state of the failed WAF node in its own state and process that request.

4.1.7 Heart Beat System

This module is used to ping WAF nodes on configured time to check the availability and load on the WAF nodes. When heart beat mechanism finds any WAF node's failure it activates fault tolerance mechanism.

4.2 Load Balancing Algorithm

The stateful load balancer is used in the proposed architecture so that client's first request is assigned to any WAF node and all subsequent requests from that client will be sent to that particular WAF. Load balancing algorithm selects node on the first request and assign the same WAF node on subsequent requests until and unless that WAF node fails. The steps of algorithm are given below.

1. When HTTP request is received in listener module, it calls load balancer which is responsible for selecting the WAF node.
2. Load balancer used in dispatcher is stateful. So in the first step it coordinates with state module and checks if the state of the client is stored there.
3. If the client's state with same session ID exists in the state module; it assigns that request to that particular WAF node.
4. In case state of client is not present in state module it means it is first request from that client and load balancing algorithm uses weighted round robin algorithm to select WAF
5. The request is assigned to that selected WAF.
6. The request is cached in cache module so in case of any failure it can be used. The state of client is stored in state module. State contains client IP, session ID and selected WAF node.

4.3 WAF's virtual cluster

After the Dispatcher the second level in high level architecture diagram shown in figure 8 is WAF nodes. Web application firewalls deeply inspects the HTTP traffic and takes time to detect different kinds of application layer attacks. This process takes a lot of time so having single WAF node as in current architecture was basic reason of bottleneck as all the traffic for the web servers need to be passed through these WAF servers. Increasing resources on Web servers' size does not provide any scalability or fault tolerance in the system. Due to increase of users in the system, system gets unavailable by the passage of time and traffic load. So there was a need of multiple WAF nodes, which can share load among themselves. For this reason cluster of WAF node is create. This cluster is called virtual cluster as it is a cluster of JVM processes rather than physical machines. As discussed in earlier chapters there is certain limitation on resource utilization of JVM process in a single system even if resources are available, so creating multiple JVM processes on single machine can help to maximum utilization of resources on single machine to provide vertical scalability. JVM process can also run on other machine to provide horizontal scalability and any number of WAF node can be attached provided the support in dispatcher module. Now in proposed architecture as WAF is not single, there is no single point of failure. If any WAF node fails whole system will not unavailable. Thus fault tolerance can be achieved with scalability of the system.

4.4 Distributed Cache of WAF

Web Application Firewalls WAFs are not only used for stateless application layer attacks but they also take care of session based attacks, where attack can't be detected from a single request but it is detected from all the requests in that session. Similarly WAFs also take care of DOS attacks. Mostly for handling session based attacks, WAFs store some information for that client session. So for transparent fault tolerance mechanism if any WAF node gets failed there should be some mechanism to recover the state of failed node so that session base attacks can be detected. For this purpose distributed cache of WAFs is introduced in proposed architecture; where each WAF node stores its state. That distributed cache is accessible to all the WAF nodes in the cluster so that if any WAF node gets down, its state can be stored to other WAF nodes and can handle session base attacks on bases of state maintained by the failure node.

4.5 Summary

This chapter shows top level proposed architecture of the system, different parts of the proposed .are discussed Dispatcher contains Listener and parser, cache, state, load balancer, fault tolerance and heartbeat module, which not only provide load balancing but also transparent fault tolerance. Virtual cluster is used for scalability and fault tolerance purpose which provides horizontal and vertical scalability. Distributed cache of WAF is maintained so that state of any WAF can be restored in case of failure.

CHAPTER # 05

System Development

In chapter four proposed architecture was discussed. This chapter discusses in detail how the proposed architecture is being designed and implemented. Implementation of all the components along with all the classes and code snippet are discussed in detail. This chapter describes class diagram of our system design shown in Figure10.

5.1 Dispatcher

As there is single dispatcher which is serving all the requests and distributes the load among all WAF nodes, it takes minimum time to process the request. If the processing of dispatcher takes more time it creates bottleneck in the system. So the main idea behind design and development of the dispatcher is to **"keep things simple and fast"**. Similarly transparent fault tolerance can be achieved in case of failure of any WAF node and for that necessary information is stored.

5.1.1 Protocol Handler

This module is designed in generic way it is not any protocol specific. Protocol is the top class in the hierarchy as shown in figure 11. Any protocol needs to be child of the Protocol class. Currently the implementation of HTTP protocol and FTP protocol is provided. Any protocol can be extended easily in the dispatcher by providing the implementation and handling of that protocol, which must be child of the main Protocol class as shown in figure 11.

5.1.2 Listener and Parser

This module is used to create listener for the protocols specified in the protocol handler. Listener is responsible to open a port, either the default port or configurable port defined for that protocol. Port is the main class of whole dispatcher. It has functionality of listener and parser. It runs thread to listen traffic for the protocols specified in the configurations. Port is responsible to parse the request. It only parses the request line of http protocol. Complete parsing of the HTTP packet is avoided because it is time consuming. When any WAF node has been assigned, the request is sent and received by LoadHandler class. In case of any error, ErrorHandler class takes control and if necessary, gives control to QuitRunnable to kill that thread.

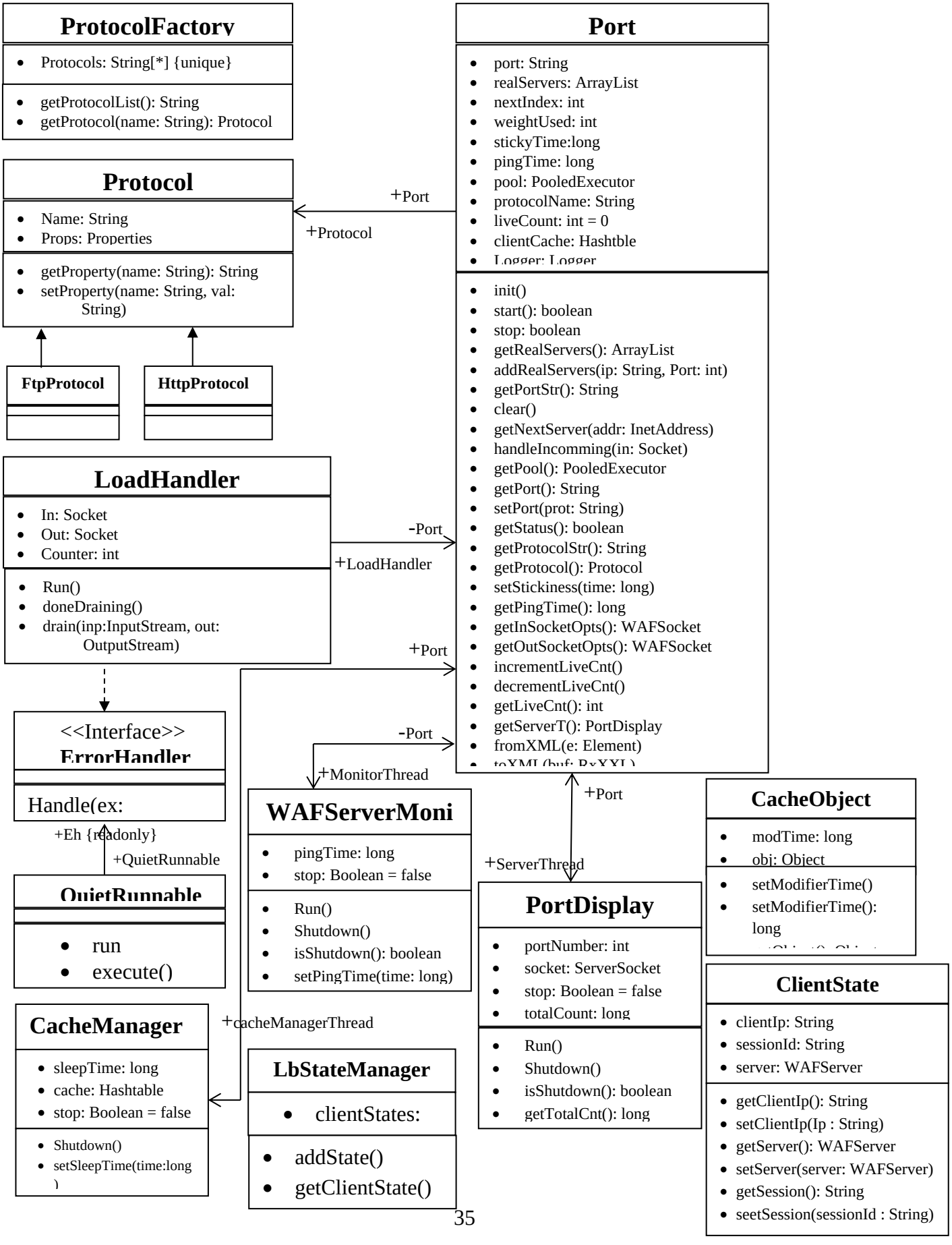
ProtocolFactory
Protocols: String[*] {unique}
getProtocolList(): String
getProtocol(name: String): Protocol
Protocol
Name: String
Props: Properties
getProperty(name: String): String
setProperty(name: String, val: String)
FtpProtocol
HttpProtocol
+Port
+Protocol
Port
port: String
realServers: ArrayList
nextIndex: int
weightUsed: int
stickyTime:long
pingTime: long
pool: PooledExecutor
protocolName: String
liveCount: int = 0
clientCache: Hashtble
Logger: Logger
init()
start(): boolean
stop: boolean
getRealServers(): ArrayList
addRealServers(ip: String, Port: int)
getPortStr(): String
clear()
getNextServer(addr: InetAddress)
handleIncomming(in: Socket)
getPool(): PooledExecutor
getPort(): String
setPort(prot: String)
getStatus(): boolean
getProtocolStr(): String
getProtocol(): Protocol
setStickiness(time: long)
getPingTime(): long
getInSocketOpts(): WAFSocket
getOutSocketOpts(): WAFSocket
incrementLiveCnt()
decrementLiveCnt()
getLiveCnt(): int
getServerT(): PortDisplay
fromXML(e: Element)
toXML(buf: RsXXL)
LoadHandler
In: Socket
Out: Socket
Counter: int
Run()
doneDraining()
drain(inp:InputStream, out: OutputStream)
-Port
+LoadHandler
+Port
<<Interface>>
ErrorHandler
Handle(ex:
+Eh {readonly}
+QuietRunnable
QuietRunnable
run
execute()
-Port
+MonitorThread
WAFServerMoni
pingTime: long
stop: Boolean = false
Run()
Shutdown()
isShutdown(): boolean
setPingTime(time: long)
+Port
+ServerThread
PortDisplay
portNumber: int
socket: ServerSocket
stop: Boolean = false
totalCount: long
Run()
Shutdown()
isShutdown(): boolean
getTotalCnt(): long
CacheObject
modTime: long
obj: Object
setModifierTime()
setModifierTime(): long
ClientState
clientIp: String
sessionId: String
server: WAFServer
getClientIp(): String
setClientIp(Ip : String)
getServer(): WAFServer
setServer(server: WAFServer)
getSession(): String
seetSession(sessionId : String)
CacheManager
sleepTime: long
cache: Hashtable
stop: Boolean = false
Shutdown()
setSleepTime(time:long)
+cacheManagerThread
LbStateManager
clientStates:
addState()
getClientState()

Figure 10: Class Diagram of Dispatcher

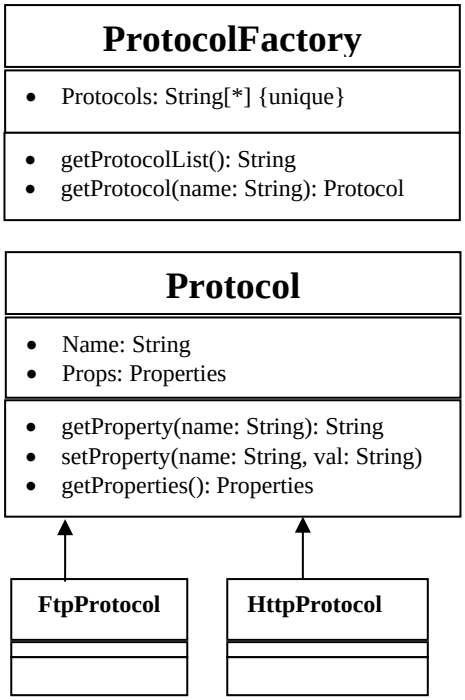

Figure 11: Class Diagram of Protocol

5.1.3 Load Balancer

This module is used to distribute the load among the available WAF nodes. Again Port class is using weighted round robin algorithm to select next WAF node by using getNextServer() method as shown in figure 12 above. This module keeps track of WAFs using WAFServer class. This module maintains cluster table of WAF. WAFServer class contains IP, port and weight of WAF node. WAFProcessor class is used to determine different characteristics of WAF node.

5.1.4 Cache

This module caches the client request until its response is not received from the WAF node. This module consists of two main classes, one class is CacheObject and other is CacheManager as shown in figure 12. CacheObject holds the client request and request time when sent to any

WAF node. Request time updates if same request is sent to other WAF node in case of failure. CacheManager is a thread which keeps track of requests which are being sent to WAF nodes but still there response is not received.

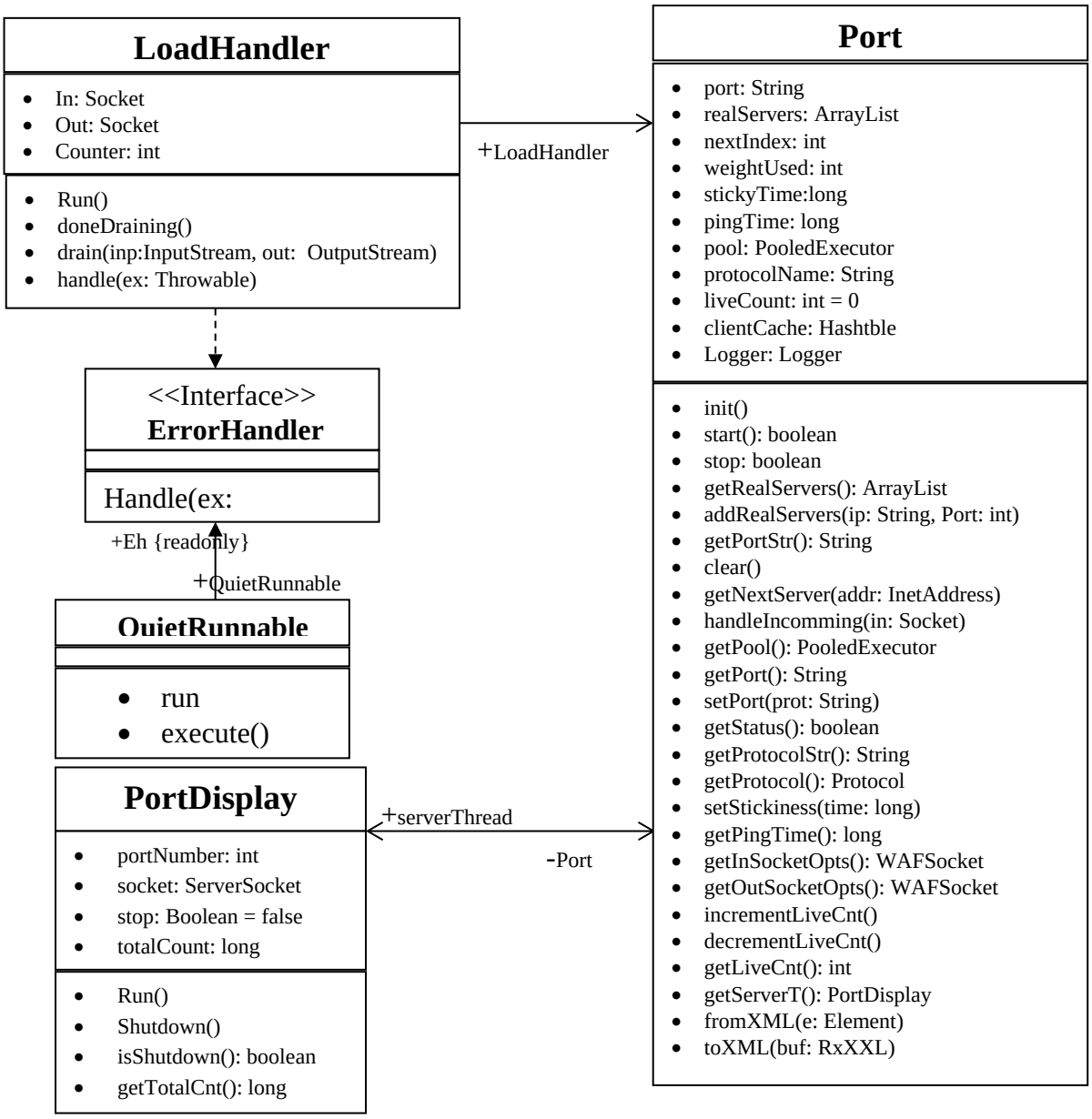

Figure 12: Class Diagram of Listener and Load Balancer

5.1.5 State

This module is used to hold the states of client. The state information includes the client IP, session ID and the WAF node assigned. ClientState and LbStateManager classes are used for maintaining state of client in Dispatcher. ClientState has attributed to hold the information required storing client information and LbStateManager keeps track of all the client states.

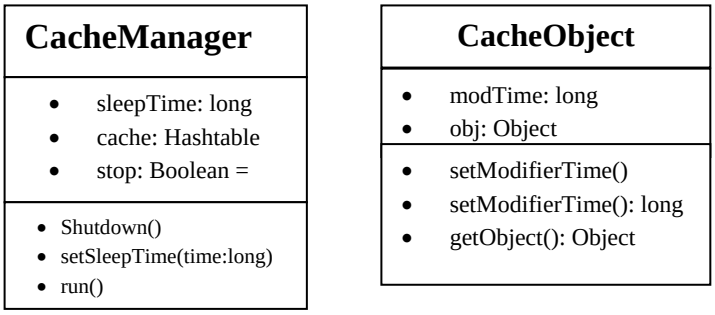

Figure 13: Class Diagram of Cache

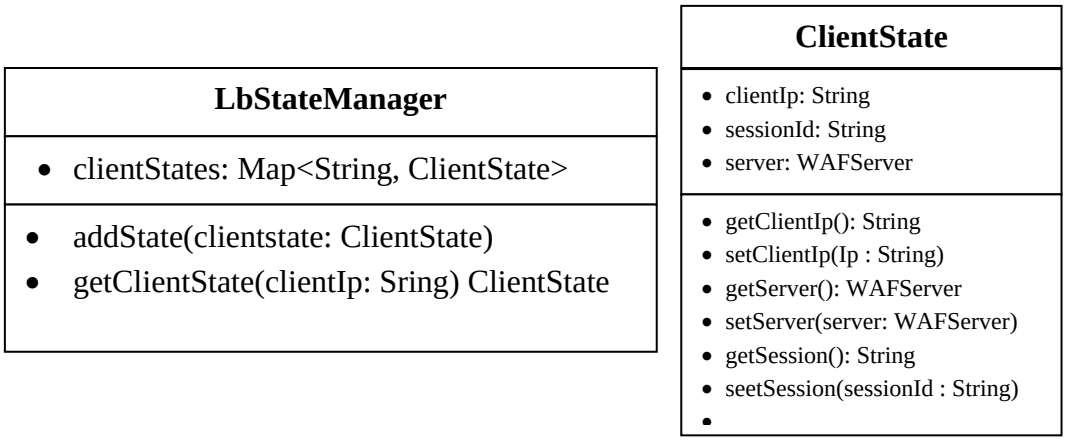

Figure 14: Class Diagram of State

5.1.6 Fault Tolerance Module

This module handles any failure in WAF node. This module includes Port and LoadHandler classes, ErrorHandler in the dispatcher and StateBackup and StateRecover classes in WAF node. When any failure is detected by Port and LoadHandler classes, control is being transferred to the

ErrorHandler class which starts the fault tolerance mechanism. It gets next WAF with the help of Port class update state, caches accordingly and sends request from cache to next WAF node. As discussed earlier WAF also maintains state so in current design distributed cache is maintained in shared directory where each WAF stores its state on a file using StateBackup class. When some WAF node gets failed, selected WAF node recovers failed WAF node state by using StateRecover class.

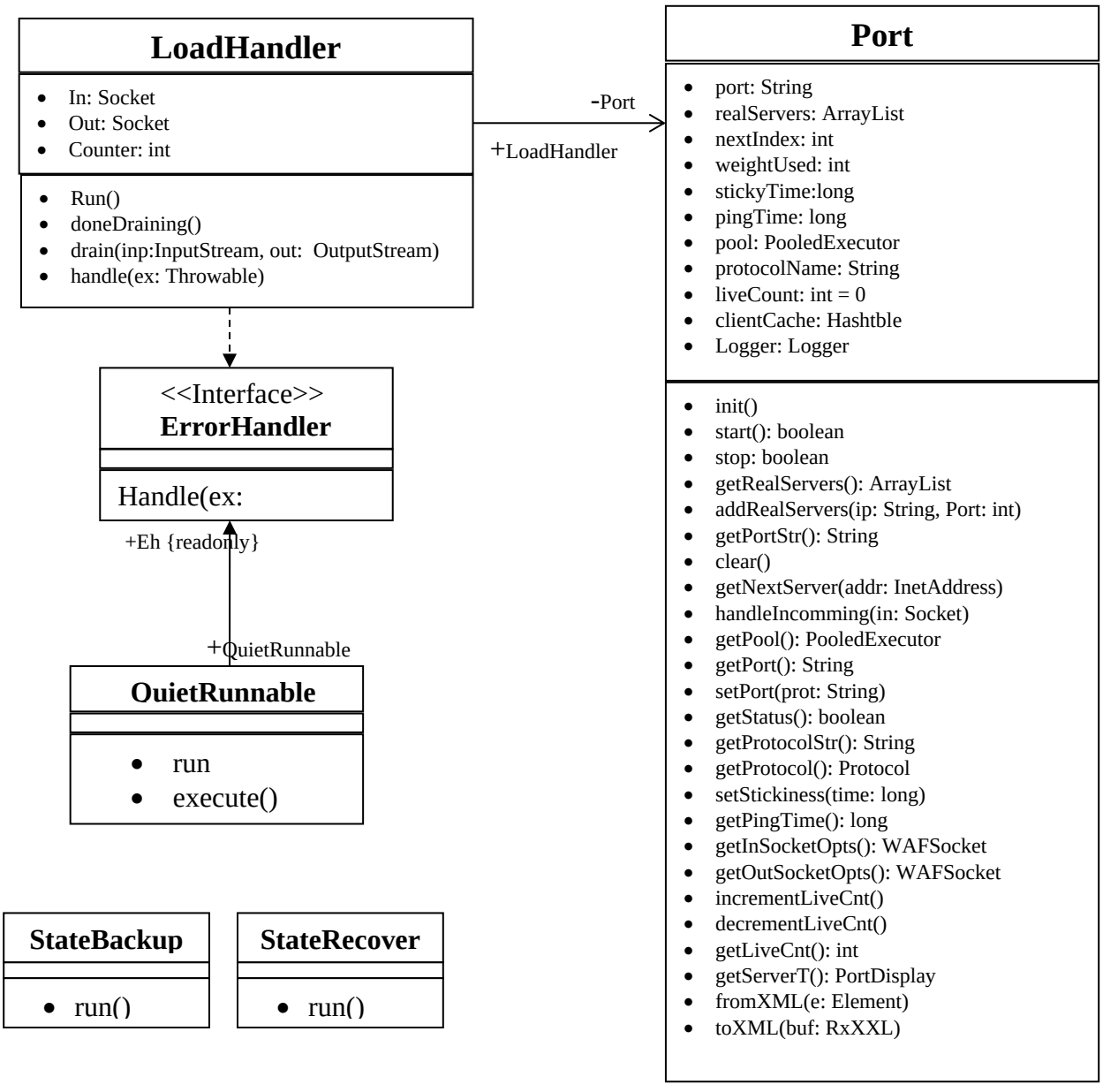

Figure 15: Class Diagram of Fault Tolerance

5.1.7 Heart Beat System

This module is used to ping WAF nodes on configured time to check the availability and load on the WAF nodes. WAFServerMonitor class is a thread which has specified ping time. It checks periodically if all the WAFServers are responding or not. If any WAFServer becomes unavailable it sets it's wafServerStatus as false.

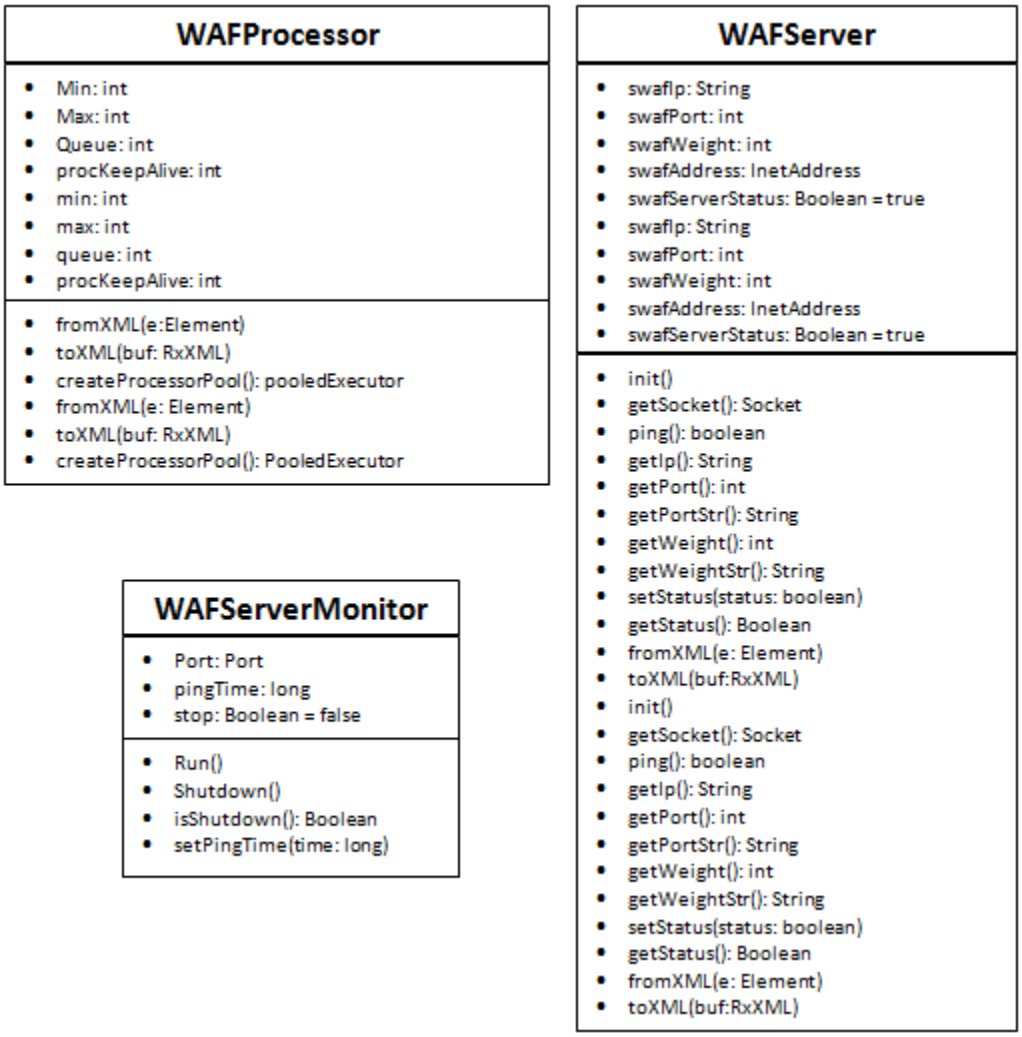

Figure 16: Class Diagram of Heart Beat System

5.2 Distributed Cache of WAF

In previous chapters it was discussed that WAF maintains state for prevention of session based attack. So the concept of distributed cache is introduced i.e. there is a central repository where all

WAF nodes maintain their state which is accessible by all WAF nodes so if any WAF node fail down, its status can be recovered. A shared directory is used where each WAF node writes its status on any shared file after some regular time interval to prevent session based and DOS attacks. This task is being done by using StateBackup class shown in figure 12. When any WAF gets failed, its state is recovered by using StateRecover class shown in figure 12.

5.3 Summary

This chapter described detailed system design and implementation of all the components of proposed system. It describes the class diagram for Dispatcher ant its module like Listener and parser, cache, state, load balancer, fault tolerance and heartbeat module. Distributed cache of WAFs is maintained in form of shared directory where each WAF writes its response in files so the state of any WAF can be restored in case of failure.

CHAPTER # 06

Evaluation

6.1 Overview

In this chapter the focused evaluation is discussed while making the test runs and comparing them over different setups. Test runs are performed on both Windows machines and on high end Linux machines using different loads. Results are concluded at the end.

6.2 Evaluation Criteria

A tool named 'Jmeter' [26] by apache is used to test the system. Jmeter is stress testing tool which is used to measure the performance outcome of an application. In this tool there are several characteristics which help in setting up a test run for an application. Stress varies based on these characteristics. The major characteristics of a test run includes following properties.

- Number of Requests / Second
- Total Number of samples in the test run
- Total Throughput of the test
- Error Rate

6.2.1 Number of Requests / Second

This parameter defines the concurrent number of requests generated by Jmeter in that test run. Number of requests per second is defined as total number of users in the test run into the number of requests generated by one user in one second. This parameter defines the work load to the application in one second.

6.2.2 Total number of samples in the test run

This parameter defines the total number of sample (Requests) made to the application by the test run. This parameter mainly depends on the number of samples recorded by Jmeter during the recording phase of testing.

6.2.3 Total throughput of the test

Throughput is defined as rate of successful end to end communication. So in Jmeter all the requests having valid response add in to throughput parameter.

6.2.4 Error Rate

Error rate represents any type of error code in response message or delayed or no response from the application. The more the error rate, the fewer throughputs will be generated by the test run. High error rate means application gets down during the test run. This factor helps us determining the applications capacity.

6.3 Sample Applications

There are three sample applications used for the testing purpose. The ratio between static and dynamic pages in each application is shown in figure 17.

- WebGoat
- WackoPecko
- Sample Static website

6.3.1 WebGoat

WebGoat is a deliberately insecure J2EE web application maintained by OWASP (Open Web Application Security Project) designed to teach web application security lessons. In each lesson, users must demonstrate their understanding of a security issues by exploiting a real vulnerability in the WebGoat application. For example, in one of the lessons the user must use SQL injection to steal fake credit card numbers. The application is a realistic teaching environment, providing users with hints and code to further explain the lesson. This is completely dynamic website, where pages are being created run time by using parameters in query String with static image resources. This application contains 71 dynamic pages created on run time bases.

6.3.2 WackoPecko

WackoPecko is vulnerable website for learning and security tool evaluation. This application has been built in PHP and hosted on wamp server. This application contains guest book and upload picture and guest note with add delete functionality. This application has almost dynamic pages while some pages are static like home page, contact us page and term and conditions page. So this application contains 75% dynamic pages and 25% static pages. This application contains 12 pages.

6.3.3 Sample Static website

This is a sample static web site that contains static html pages and images. This application contains 7 sample web pages plus images used on that website. This application is hosted on wamp server.

6.4 Testing Results

6.4.1 Machine Specification

First tests are performed with small number of sample size. The machine has window 7 operating system installed. The system specification of machine on which load balancer, multiple SWAF application, web application (WebGoat, WeckoPecko and sample static site) and Jmeter was installed to test the systems is given in figure 18.

6.4.2 Testing Scenarios

In testing, four machines are used.

- One is running load balancer and WAFs.
- All other running single or multiple WAFs.
- JMeter is running from 1 machine to 6 machines.

Load is tested with following scenarios

- Without load balancer 1 WAF node
- 1 load balancer and 1 WAF node
- 1 load balancer and 2 WAF nodes
- 1 load balancer and 8 WAF nodes
- 1 load balancer and 12 WAF nodes

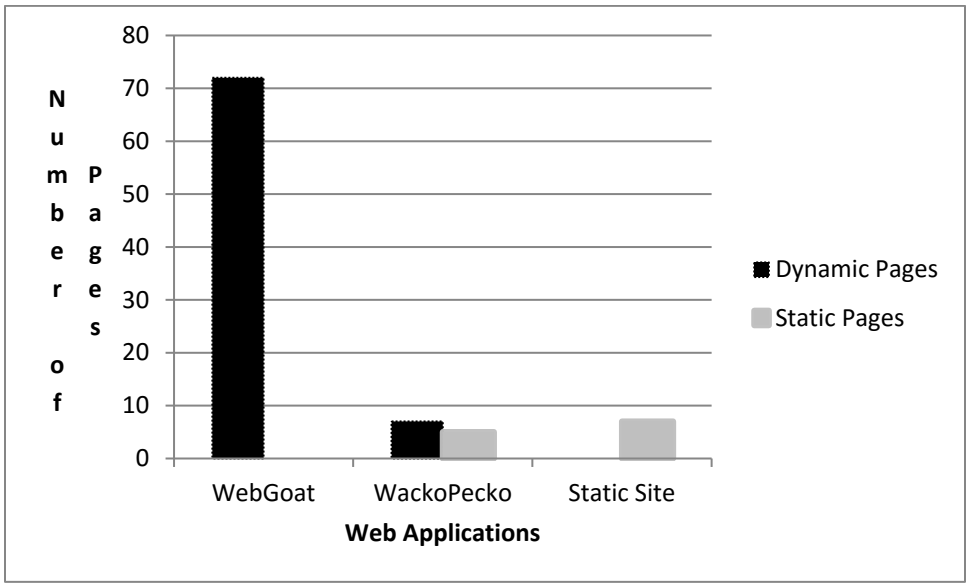

Figure 17: Static versus Dynamic Pages in Sample Web Applications

Figure 18: Specifications of Windows System used for Evaluation

Label	#Samples	Average	Min	Max	Std. Dev	Error %	Throughput	KB/sec	Avg. Bytes
/WebGoat/image….	58	1055	7	2068	816.64	0	8.9	0.07	476
/WebGoat/image….	58	1096	10	1985	818.05	0	8.9	0.52	3605
/WebGoat/image….	58	747	7	2026	791.12	0	8.9	0.1	690
/WebGoat/image….	58	618	10	1981	737.42	0	8.9	0.46	3163
/wacko/	100	9529	54	28670	7839.03	0	1.4	4.83	3563
/wacko/pictures….	100	15131	263	32012	10152.07	0	12.2	0.82	4126
/wacko/upload	50	20431	15	31974	10914.04	0	24.3	4.92	12444
/wacko/upload	50	22508	260	32012	10969.32	0	19.5	4.65	14672
/wacko/upload	50	22766	40	319818	9251.31	0	17	4.27	15389
/wacko/upload	50	21605	1862	31943	6693.03	0	15	3.35	1373
/wacko/upload	50	23513	1641	31923	6470.35	0	13.5	3.52	16045
/wacko/upload	50	23651	1759	33376	6237.49	0	12.2	3.07	15421
/wacko/upload	50	22866	1905	33387	5978.86	0	11.2	2.28	12471
/wacko/upload	50	24646	238	33374	6926.38	0	10.3	2.04	12142
/wacko/upload	50	22908	2334	31850	6147.47	0	9.8	2.05	12831
/wacko/guests/	98	15963	1145	51604	11832.87	0	10.6	0.54	3112
/wacko/pictures….	100	24251	12904	58408	12393.65	0	15.4	0.75	2983
/wacko/users	198	11474	1187	43369	7920.55	0	23.8	1.16	2979.2
/wacko/users	50	19025	8410	57343	14235.1	0	10.3	0.5	2983
/wacko/pictures….	50	28538	10876	75871	16862.08	0	11.6	0.56	2983
/WebGoat/image….	16	4493	840	40224	9647.92	0	1.1	0	49
/WebGoat/javas….	3	436	1962	1227	559.32	0	9.1	31	2079
Total	**3477**	**7404**	**6**	**87179**	**10899.21**	**0.12**	**2.7**	**14.63**	**5524.3**

Table 1: Summary Report without Load Balancer

6.4.3 Attackers to Normal Users Requests

During testing 4 types of attacks were simulated i.e. session fixation, SQL injection, XSS and DTS. Ratio of these attacks is given in table 2. In every thousand requests four hundred requests (40%) were attacks. Out of ten thousand requests fifteen hundred were Session fixation, one thousand were SQL injection, hundred were XSS and nine hundred were DTS attacks.

Attack	Percentage of occurrence
Session Fixation	40%
SQL injection	27.5%
XSS	7.5%
DTS	25%

Table 2: Percentage of occurrence of different attacks

6.4.4 Throughput (Requests/second/WAF) VS Number of Users

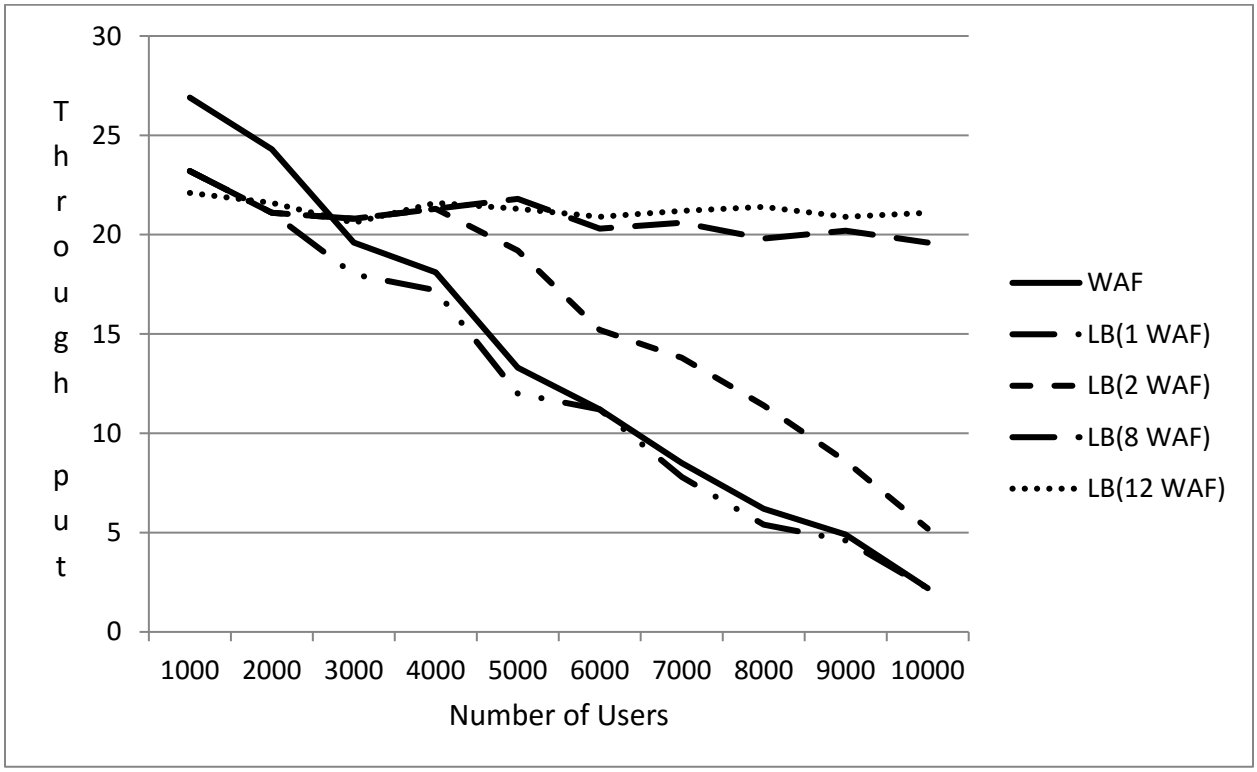

Figure 19: Throughput achieved by the proposed system against number of requests/ normal users. Results are evaluated using multiple WAF nodes and Load balancer.

Figure 19 shows the throughput achieved by the proposed system against number of requests from normal users. Graph clearly shows that initially when the numbers of users were less, WAF gave the highest throughput. But as the number of users increases, throughput decreases. System with "WAF only" reaches to almost zero throughput with 10,000 Users. Adding a load balancer

with single WAF further decreases the throughput. By adding 2, 8 and 12 WAFs respectively with a load balancer gives almost same throughput with fewer users but as the number of users increase there is a comparatively noticeable increase in throughput. Adding two WAFs with a load balancer shows negligible increase in throughput as compared to WAF without load balancer and combination of single WAF plus a load balancer. System with 8 and 12 WAFs and a load balancer stabilizes the throughput even with increase in number of requests.

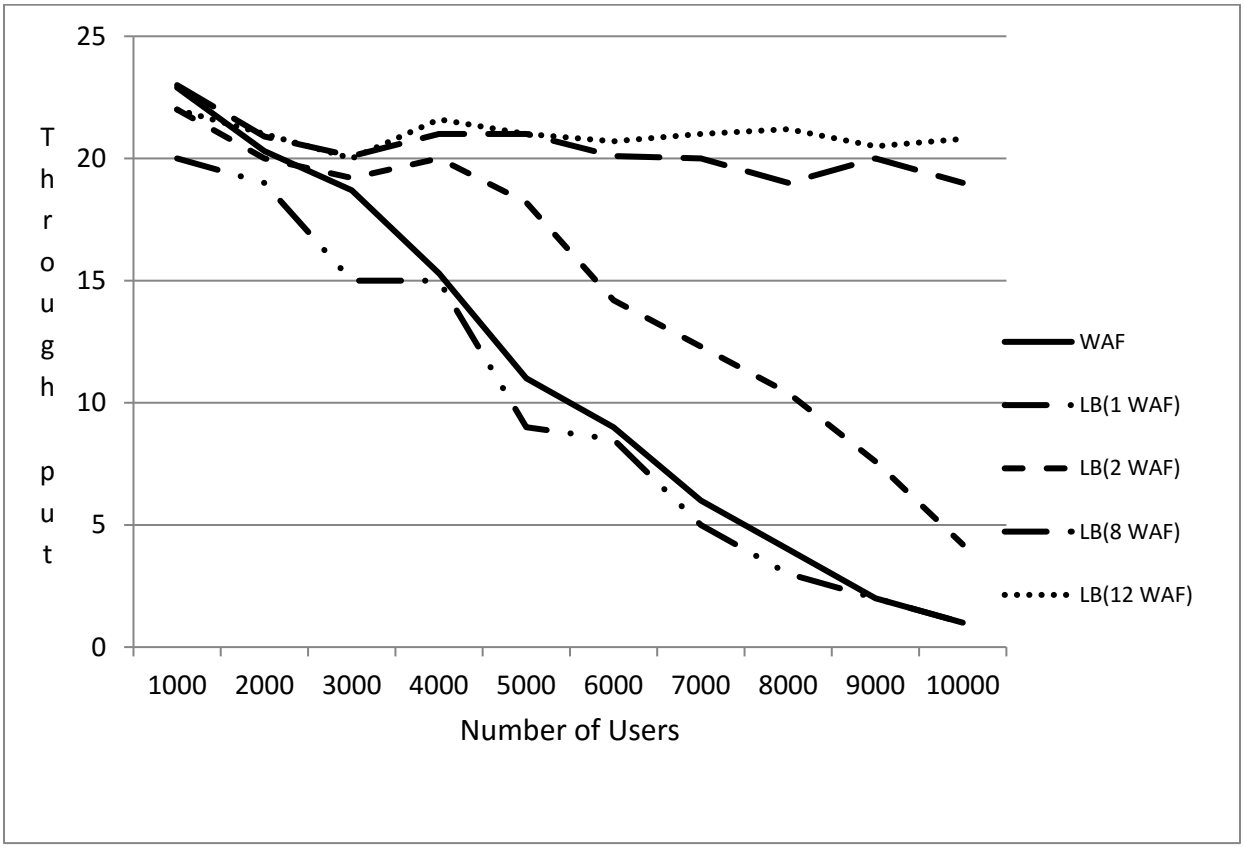

Figure 20: Throughput achieved by the proposed system against number of requests/malicious users. Results are evaluated using multiple WAF nodes and Load balancer.

Figure 20 plots the same statistics as in figure 19 but in this case requests are from malicious users. Although throughput varies in a same way as in figure 19 for the varying number of requests, but over all throughput achieved by the system is less as compared to throughput achieved in case of normal users. The difference in throughput decreases as the number of WAF nodes increases and it becomes almost negligible in case of eight and twelve WAF nodes.

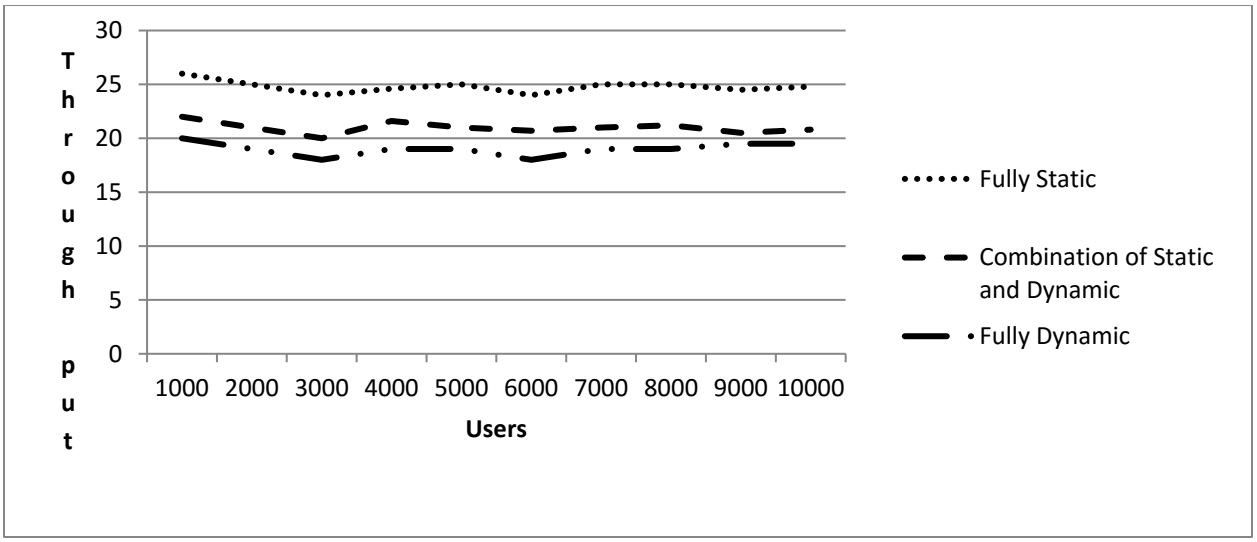

Figure 21: Through put (Static VS. Dynamic VS. Combination of Static and Dynamic Websites) with LB (12 WAFs)

Figure 21 shows the throughput achieved by the system in case of fully static, fully dynamic and combination of static and dynamic websites. System gives the maximum throughput when all the pages in a website are static and gives minimum throughput if the website under test is a fully dynamic. Most of the websites are combination of static and dynamic pages and throughput achieved in this case is between fully static and fully dynamic websites.

6.4.5 Throughput (KB/second/WAF) VS amount of data transferred

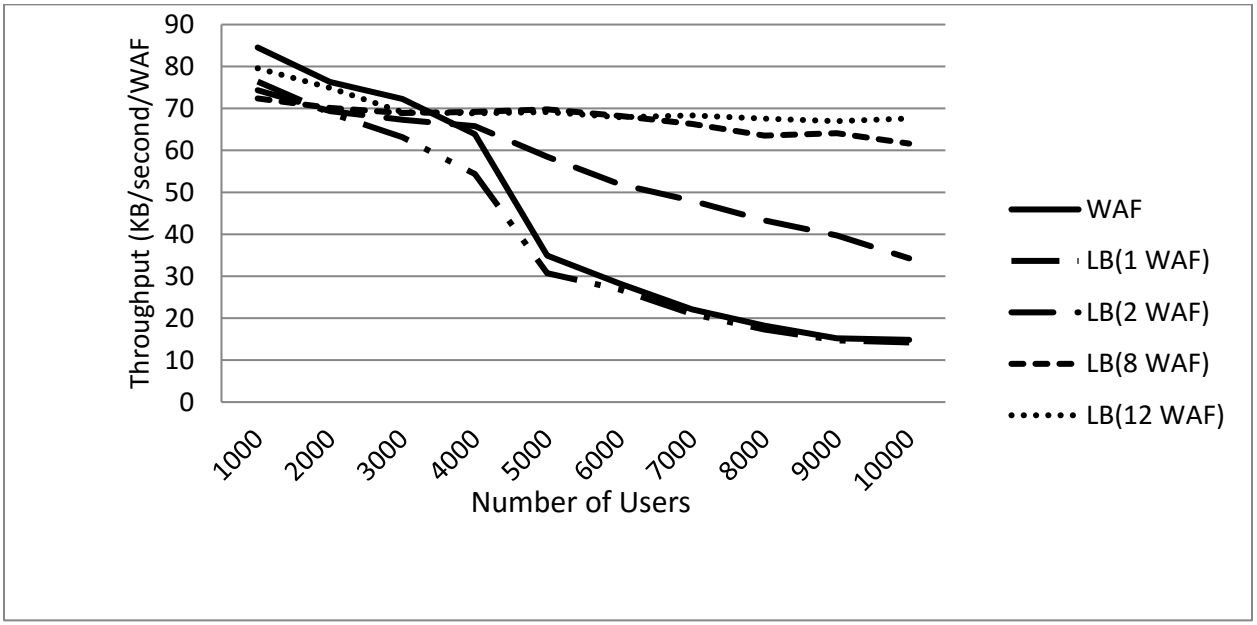

Figure 22: Throughput achieved by the system against amount of data transmitted by the users. Results are evaluated using multiple WAF nodes and Load balancer.

Figure 22 plots the throughput against number of users/requests. In this testing scenario the main focus is amount of data (in KBs) send by the users instead of number of users/requests. Results are almost same as in figure 19. With less transmission data WAF only system gives the highest throughput, but as the amount of data increases throughput decreases. System with one WAF and load balancer cannot improve the results; instead throughput is even less then WAF without loadbalancer. System gives better throughput by increasing number of WAFs in combination with load balancer. Initially with less amount of data 2, 8 and 12 WAFs in combination with load balancer gives almost same throughput but as the amount of data increases there is a visible difference in throughput. There are much improved results for 2 WAFs and load balancer but complete stabilization is achieved by adding 8 and 12 WAFs with a load balancer.

6.4.6 Error Rate (%) VS Number of Users

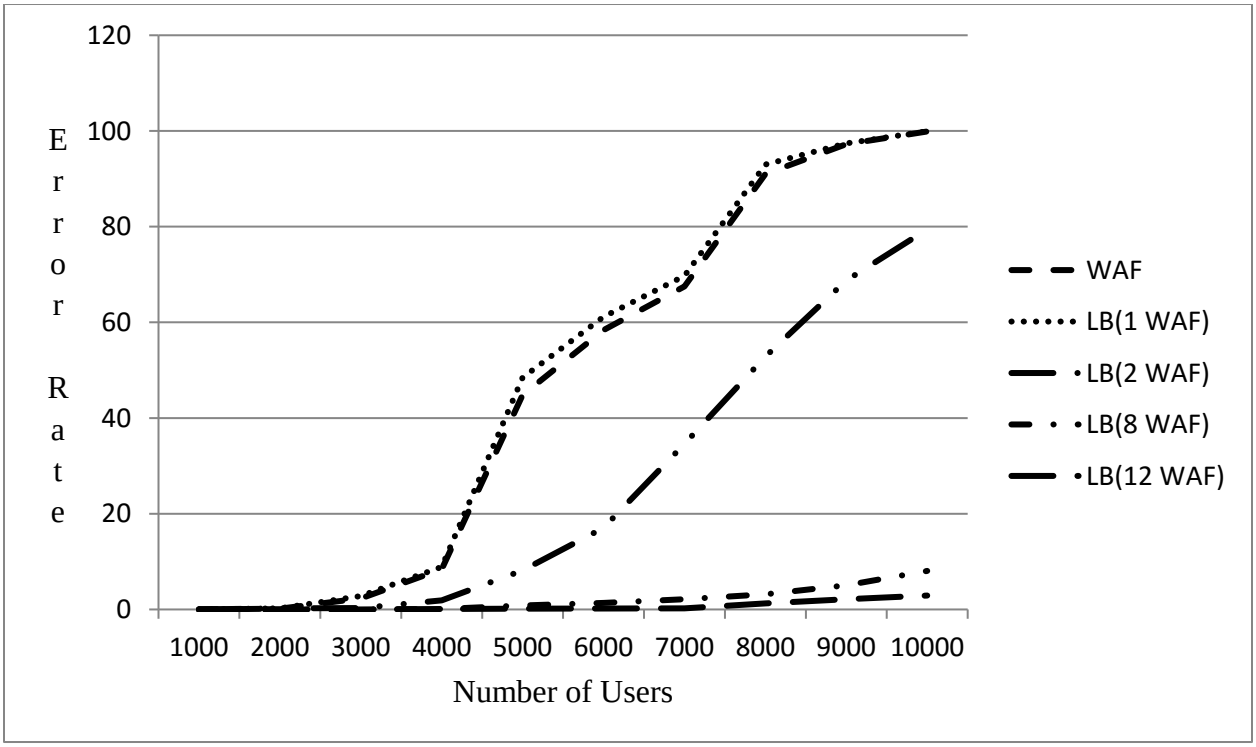

Figure 23: Error Rate (%) against number of requests/users

Error rate represents lost requests. A lost request is a request whose response is not received. Figure 23 plots error rate against number of requests/users. Initially when the number of requests were as low as 1000, error rate was 0% with all the implementations (with and without load balancer). As the number of requests increases, there is an increase in error rate Error rate is

increased in case of WAF only and single WAF and load balancer. With two WAFs in combination with load balancer error rate is less as compared to WAF only and single WAF and load balancer combination but this error rate is not negligible. By adding 8 and 12 WAFs with load balancer error rate remains near to zero even with the maximum number of requests under testing scenario.

6.4.7 Parser Comparison

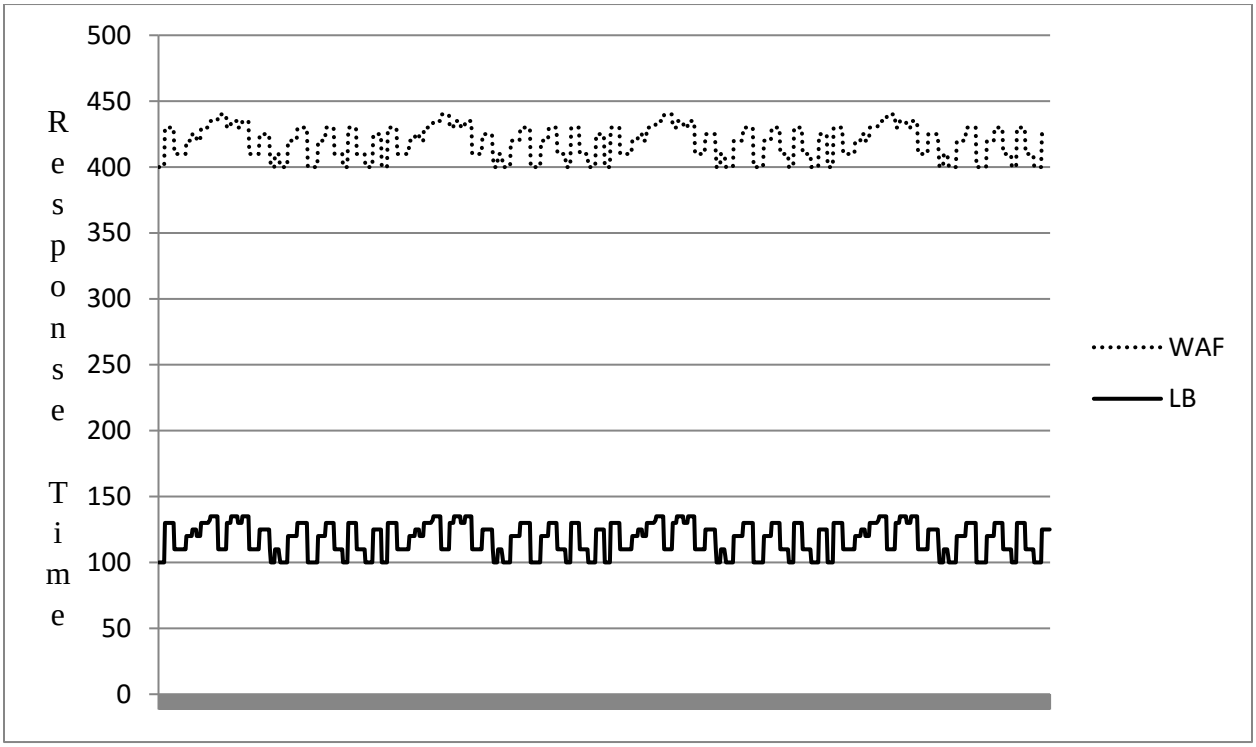

Figure 24: Parser Comparison

Parser is a program that parses http request string and creates java objects out of it. Parsing is a time consuming job. Including parser in current implementation increases the response time. In order to get best possible results in terms of throughput and response time, a thin parser is written. This parser is implemented to parse only the first line of http request. As whole request is not to be parsed, there is a visible decrease in response time as shown in figure 24.

6.4.8 Performance Comparison (Response Time in millisecond)

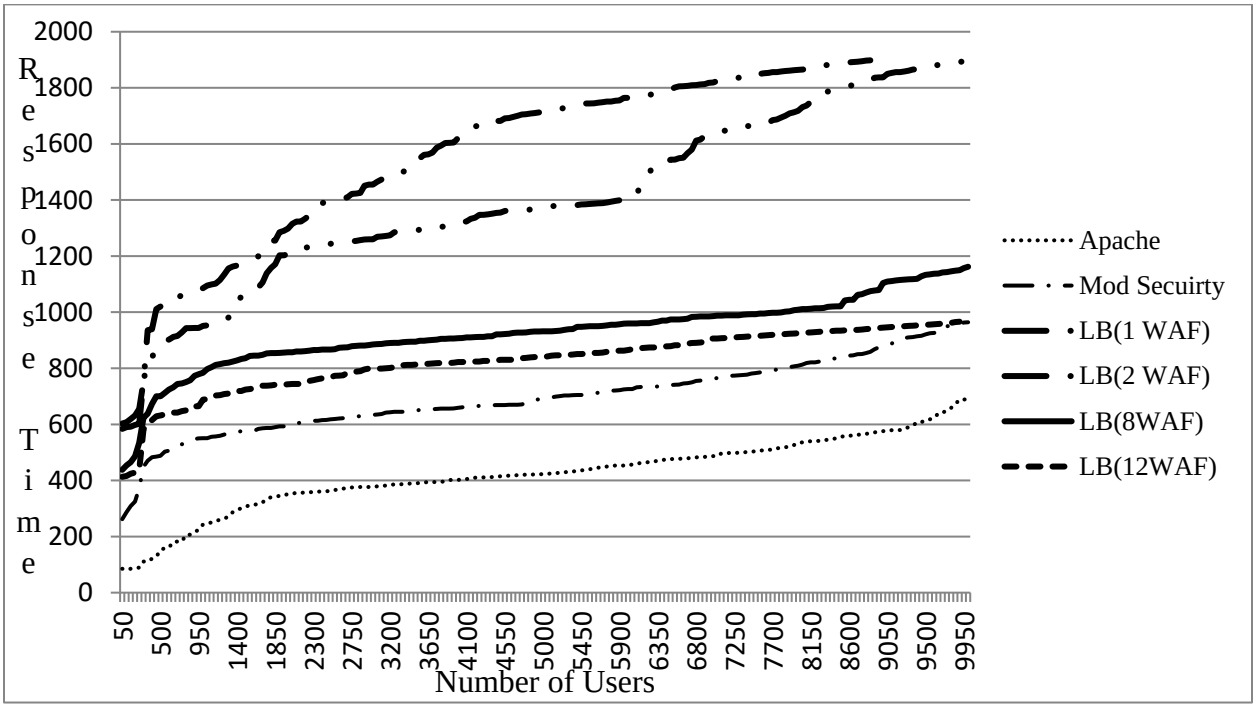

Figure 25: Performance Comparison (Response Time in millisecond) of proposed architecture with Apache and Mod Security

Figure 25 shows that there is a very huge performance gap between single WAF in combination with load balancer and Apache and Mod Security. The gap is decreased by adding WAFs. Graph shows that gap is almost negligible and performance plots of Apache and Mod Security touches 8 and 12 WAFs plus load balancer combination performance plots. Performance is measured in terms of response time in milliseconds.

CHAPTER # 07

Conclusion & Future Enhancements

This chapter concludes the overall work done in this thesis from the study of the scalability and fault tolerance till its implementation and results, which shows that scalability and fault tolerance is achieved.

7.1 Conclusion

In the beginning the idea of an effective application level security mechanism is scrutinized. Most of the issues in low traffic server solutions have been the major concern of the current research. The present high level architecture for semantic based application level intrusion detection system has also been presented by the current system. Besides, various components in the current system architecture have been highlighted. Top level architecture of the system and the details of different parts are discussed. Dispatcher contains Listener and parser, cache, state, load balancer, fault tolerance and heartbeat modules which not only provide load balancing but also transparent fault tolerance. Virtual cluster is used for scalability and fault tolerance purpose. Distributed cache of WAF is maintained so that state of any WAF can be restored in case of failure. Detailed system design and implementation is discussed along with class diagrams of dispatcher and its modules like Listener and parser, cache, state, load balancer, fault tolerance and heartbeat module. Distributed cache of WAF is maintained in form of shared network directory. Testing results clearly shows that proposed system is scalable and remains stable in case of large number of users. In case of failure of any WAF node, any other WAF node takes charge and all state of failed WAF node is recovered.

7.2 Future Work

The light weight dispatcher which is developed in this approach is working on weighted round robin algorithm. Auto increment of cluster can be used in future to make system stable for longer period of time. It means that if all WAF nodes seem to be busy, dispatcher should be intelligent enough to create WAF node (JVM process) by itself for balancing the node. Currently shared network drive is used for distributed cache which is bit slower. A “on memory” distributed cache solution can be designed to speed up fault tolerance mechanism.

Annexure - A

References

Bibliography

[1] "FireWall," Webopedia, [Online]. Available: http://www.webopedia.com/TERM/F/firewall.html.

[2] T. Rowan, "Application firewalls: filling the void," 2007, pp. 4-7.

[3] "The ten most critical Web application security vulnerabilities," 2007. [Online]. Available: https://www.owasp.org/images/e/e8/OWASP_Top_10_2007.pdf.

[4] "Only 10% of Web applications are secured against common hacking techniques," 2004. [Online]. Available: http://www.imperva.com/company/news/2004-feb-02.html.

[5] G. Hulme, "New software may improve application security," 2001. [Online]. Available: http://www.informationweek.com/story/.

[6] E. W. Flup and R. J. Farley, "A Function-Parallel Architecture for High-Speed Firewalls," *IEEE International Conference on Digital Object Identification,* 2006.

[7] E. D. Zwicky, S. Cooper and D. B. Chapman, "Building Internet Firewalls," Spring 2006. [Online]. Available: http://pages.cs.wisc.edu/~jha/course-archive/642-spring-2005/index.html.

[8] L. Qui, G. Varghese and S. Suri, "Fast firewall implementations for software and hardware-based routers," in *in Proceedings of ACM SIGMETRICS*, June 2001.

[9] S. Suri and G. Varghese, "Packet filtering in high speed networks," in *Proceedings of the Symposium on Discrete Algorithms*, 1999.

[10] R. L. Ziegler, Linux Firewalls, München: Markt + Technik Verl, 2002.

[11] P. Byrne, "Application firewalls in a defense-in-depth design," in *Network Security*, September 2006, pp. 9-11.

[12] "Software Firewalls: Made of Straw?," Symantec.com, [Online]. Available: http://www.symantec.com/connect/articles/software-firewalls-made-straw-part-1-2.

[13] "Software-as-a-service (SAAS) vs "Do-it-yourself" with a web application scanner," [Online]. Available: https://www.whitehatsec.com/resource/whitepapers/saas.html.

[14] "WhiteHat Security "Website Security Statistics Report"," [Online]. Available: http://www.slideshare.net/jeremiahgrossman/whitehat-security-website-security-statistics-report-q109.

[15] F. Hanik, "Inside the java virtual machine," 29 08 2007. [Online]. Available: http://www.springsource.com/files/uploads/all/pdf_files/news_event/Inside_the_JVM.pdf.

[16] E. Fredkin, "Trie memory," Communication. Of ACM, May 1988, pp. 490-500.

[17] S. Nilason and G. Karlsson, "lP-address lookup using LC-tries," *IEEE Journal on Selected Areas in Communications,* vol. 17, June 1999.

[18] A. Andersson and S. Nilsson, "Improved behavior of tries by adaptive branching," *Information Processing Letters,* vol. 46, pp. 295-300, 1993.

[19] C. Benecke, "A parallel packet screen for high speed networks," in *Proceedings of the 15th Annual Computer Security Applications Conference*, 1999.

[20] M. Y. Luo and C. S. Yang, "System support for Scalable, Reliable and Highly manageable Web hosting service," *Symposium on Internet Technologies and Systems,* 2001.

[21] P. N. Ayuso, R. M. Gasca and l. lefevre, "Demystifying Cluster-Based Fault-Tolerant Firewalls," *IEEE Internet Computing,* vol. 13, no. 6, 2009.

[22] B. Y. Zhang, Z. YaoMo, G.-W. Yang and W.-M. Zheng, "Dynamic Load-Balancing and High Performance Communication in Jcluster," *IEEE International,* 2007.

[23] T. Grubber, "A Translation Approach to Portable Ontologies. Knowledge Acquisition Archive," June 1993, pp. 199-220.

[24] J. Undercoer, A. Joshi and J. Pinkston, "Modelling computer attacks: An ontology for Intrusion Detection," *The Sixth International Symposium on Recent Advances in Intrusion Detection,* pp. 113-135, 2003.

[25] R. Fikes and D. I. McGuinness, "An Axiomatic Semantics for RDF," 18 12 2001. [Online]. Available: http://www.w3.org/TR/daml+oil-axioms.

[26] "Criteria Evaluated by Apache Jmeter," [Online]. Available: http://jakarta.apache.org/jmeter.

[27] A. Razzaq, A. Hur, N. Haider and F. Ahmad, "Multi-Layered Defense against Web Application Attacks," *Information Technology New Generations. Sixth International*

Conference on Digital Object Identifier, pp. 492 - 497, 2009.

[28] "Application Layer Firewall," F5 Networks, Inc., [Online]. Available: http://www.f5.com/glossary/application-layer-firewall.html.