Progress Report: Implementation of a Web Application Firewall for a high availability front end

Hernan Espinosa Reboredo
SIGMA Gestió Universitaria,
Universitat Autonoma de Barcelona
Sant Cugat del valles, Barcelona, Spain
hespinosar@gmail.com

I. Introduction

The aim of this project is to build a high availability web application firewall [1] that will monitor and analyse client's requests data with the aim of protecting the backend server's security.

A Web Application Firewall helps to protect web applications by filtering and monitoring HTTP traffic between a web application and the Internet. By deploying a WAF in front of a web application, a shield is placed between the web application and the Internet, increasing security, performance and reliability by having client's requests pass through the Web Application Firewall before reaching the servers. A Web Application Firewall operates through a set of rules often called policies. These policies aim to protect against vulnerabilities in the application by monitoring and filtering out malicious traffic.
In this project, we are going to carry out an exhaustive analysis of the events that take place in the built architecture, using different development tools to monitor the traffic.

The main goal that has motivated this work is to improve the security of any architecture's servers, to have a better control over the events of their activity, and to reduce the time of action before an incident. Therefore, this project is deployable to any web application architecture in order to improve its security.

II. Objectives

The Web Application Firewall must be able to protect web applications by filtering and monitoring HTTP traffic between a web application and the Internet. We have planned a set of objectives to achieve at the end of this work. The objectives are listed in incremental priority.

- The Comprehension of how a Web Application Firewall works, and it's uses.
- The Comprehension of how the different software tools, that are used in this project, work and how are going to be for use to make the project possible.
- The Development of a scalable client-server web application architecture.
- The Development of high availability, data persistency, and load balancig to the web application firewall.
- The architecture justification: justification of the structure proposed, explaining the decisions made and the reasons why the structure is suitable.
- Initial project tests: For every stage of our project, a test is being performed to validate that the project satisfies the initial requirements.
- The analysis and monitoring of the data-flow:
  From different client's requests, we want to analyse all the client requests that are done in real-time and we will want to establish rules to grant or deny the access from the requests to the backend servers.
- The Improvement of backend server's security and the reduction of security vulnerabilities from web application services.
- Deployment of the Web Application Firewall rules to the service.
- Log management and analysis.
- Comprehension of data real time data flow monitorisation and benchmarking.

III. Methodology and Planning

This project follows a software development model based in incremental prototyping [2]. Each prototype will fulfil a set of requirements. The final prototype will accomplish the objectives from this work. Each prototype built will serve as a mechanism for the definition of requirements, since requirements can change from the initial requirements definition.

This work is divided into three main phases, without keeping in mind the delivery of the reports and the presentation.

The methodology that will be used to follow the development of this project is the Gantt chart, which is a useful way of scheduling a project and defining the different dependencies between tasks. In figure 1 of the appendix there is a Gantt diagram, in which it you can see the tasks and the planning in weeks of the work.

The actual state of evolution from the project is that the initial definition of objectives and requirements, and the analysis of the current building tools to develop the project have been accomplished. A running scalable client-server web application architecture has been deployed. Data persistence, network design, traffic load balancing, front end high availability and traffic analysis have been deployed to the architecture. The initial project tests and the final architecture's development and testing are in progress. In parallel with the development and analysis of the software,

the documentation and justification of achievements are in progress as well.

IV. TOOLS FOR DEVELOPMENT

In this section of the paper, are described, the set of software that is going to be used to develop this work.
This project will be developed using a Linux operating system, Ubuntu.

A. *Docker: The development environment*

Docker [4] is a tool designed to make it easier to create, deploy, and run applications by using containers [5]. Containerization allow us to package up an application with all the parts it needs, such as libraries and other dependencies, and run it all out as one package. Thanks to its environment management, makes it a suitable environment for this project.

Separating the different components of our web application service into different containers will have security benefits, because if one container is compromised the others remain unaffected. Separating the different components of our web application into different containers will avoid conflicts with dependencies between the containers since each container is isolated from each other.

B. *HAProxy: Building tool*

Since in this work a high availability web application will be built, we are going to use the HAProxy software to build the high availability load balancers. HAProxy is an open source, very fast and reliable solution offering high availability, load balancing, and proxying for TCP and HTTP-based applications. It is particularly suited for web application services.

C. *ModSecurity: Building tool*

For the monitorisation of traffic load, we will use ModSecurity software for analysis to be able to make conclusions. ModSecurity is a toolkit that will be used for real-time web application monitoring and logging.

D. *Elastic Stack: Data Management tool*

Elastic Stack [6] is a powerful search engine which will allow us to process logs generated from the architecture. Elastic Stack is a complete end-to-end log analysis solution which helps in deep searching, analysing and visualizing the log generated from different machines.

Elastic stack will let us search through the multiple logs at a single place and identify the issues spanning through multiple servers by correlating their logs within a specific time frame found in our environment.

E. *Security Onion: Testing tool*

Security Onion is a free and open source Linux distribution for intrusion detection, monitoring, and log management. It includes Elasticsearch, and many other security tools.
Security Onion collects many tools for forensic analysis, both networks and systems, so that we can guarantee the proper functioning of all of them and the absence of all kinds of intruders in the net.

F. *Github: Version Control tool*

A versioning tool is being used, in order to keep copies and record of the work done over the course of this project. A repository has been created in GitHub [3] where the project is updated, and version controlled.

V. PHASE 1: WAF_V1

a) Analysis

The first prototype consists in building a base client-server architecture and to test its functionality to corroborate that it works. The architecture will serve an index.hml stored in backend server. The aim of this phase is to develop a simulation of a basic web server architecture so in the following phases, different resources and processes can be deployed to suit the different objectives from the work. In this phase of the project, data persistence, data consistency, the management of architecture Networking and logs have been developed.

b) Design

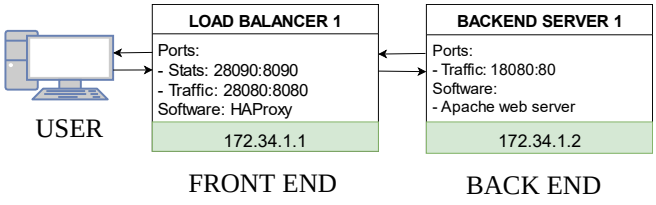


Figure 1 shows the design structure proposed for phase one of this work.

This phase of the architecture is built with Docker containers. The backend of our architecture consists of an Apache web server, which stores a html file, that will be requested by the user client. The frontend of the architecture consists of a HAProxy load balancer node, in order to distribute requests from the client to the backend server. The need of transparency and privacy of the backend server justifies why we have opted to use the HAProxy as the software used for the front-end node. Persistence, log management, consistence, networking and container communication have been integrated to WAF_v1 architecture.

c) Implementation

Since we are using containerization to develop our work, data doesn't persist when a container no longer exists, and it can be difficult to get the data out of the container if another process needs it, to manage that problem, we are using Host-based Persistence among containers, in order to store files in the host machine, and the files are persisted even after the container stops. By using host-based persistence [7], data is persisted outside of the container, which means it will be available when a container is removed. In phase 1, data persistence is important since we need our architecture's containers to centrally store logs to the same directory, making it easier to process the logs, so we can test the architecture's performance.

For log management, we are going to use the software Rsyslog.

Rsyslog is a powerful, secure and high-performance log processing system which accepts data from different types of source and outputs it into multiple formats. Redirecting all the logs from rsyslog to the standard out device makes logs play nice with docker default logging.
The front-end node uses the software HAProxy to load balance traffic to the backend servers. HAProxy software, doesn't log to stout by default, we need to have a configuration which will take logs generated by the software and ship them to a specific local directory.

Since the design is scalable, in a future situation we may have the situation where each backend server is connected to different databases to serve client requests. Data is generally replicated to enhance reliability or to improve performance. One major problem is to keep replicas consistent. Data Base servers must be configured to perform Master-Master replication as load balancing involves both reading and writing to all backends.

When configurating the network for the architecture environment in this phase of the project, a private network has been used for security reasons since sensitive data is travelling through the private network. If a public network configuration is used, the sensitive information would face security vulnerabilities, because of the information not being encrypted. The HAProxy load balancer node, is a basic load balancing setup where the front-end listens on a specific IP address and port, then forwards the incoming traffic to a specified group of webservers.

d) Test

In order to test the design, by typing to any web browser the following URL address: 172.43.1.1:28080/index.html, the html file stored in the Backend Server node will be requested. The HAProxy software will handle the request and will forward it to the backend server. The backend server will deal with the response, which will be delivered to the client by the HAProxy node.

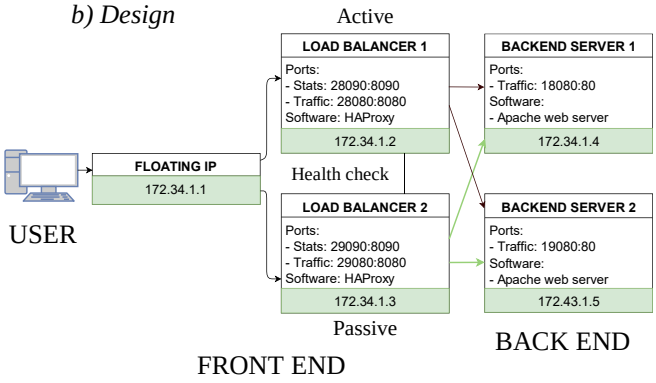
Thanks to data persistency, and Rsyslog software the logs from the http request will be stored to the host's computer and we are able to trace the client request and response.
Thanks to the Loadbalancer 1 node, the IP address from the backend server will not be exposed to the public network, therefore making the deign more secure by not exposing the internal design and implementation.

VI. PHASE 2: WAF_V2

a) Analysis

The Second prototype from this project consists in upgrading the basic client-server architecture built in phase 1 and to test its functionality to corroborate that it works. The aim of this phase of the project is to upgrade the simulation of a basic web server architecture by increasing the number of frontend and backend nodes which are involved in serving user client requests, in order to develop traffic load balancing and high availability.

b) Design



This phase of the project takes advantage of containerization as well as phase 1. The backend of our architecture consists of two Apache web servers, which store an html file that will be requested by the user client. The frontend of the architecture consists of two HAProxy load balancer nodes, in order to distribute requests from the client to the backend server. The need of high availability and traffic load balancing justifies why we have opted to upgrade both the frontend and the backend from our architecture.

The main difference between the prototype from phase one and the prototype from phase two is that high availability is introduced in the architecture, making the design fault tolerant.

To achieve high availability, the front end from Phase 1 has been upgraded. Now the front end is composed by two load balancers, a Master node, which works as an active server, and the other one, the Backup load balancer, working as a passive server. Whenever one of the load balancers is unavailable, the one that is available takes the Master role and can redirect and balance client request throughout the architecture, making the design more secure and reliable.

c) Implementation

In this version of the project we have set up a two-node front end load balancer in an active/passive configuration with HAProxy and keepalived. The load balancer sits between the user and two backend Apache web servers that hold the same content. Not only does the load balancer distribute the requests to the two backend Apache servers, it also checks the health of the backend servers. If one of them is down, all requests will automatically be redirected to the remaining backend servers. In addition to that, the two load balancer nodes monitor each other using the software keepalived, and if the master fails, the slave becomes the master, which means the users will not notice any disruption of the service.

Keepalived is the software that manages the configuration of the virtual floating IP from the architecture frontend. Thanks

to Keepalived, we can scale our architecture by adding more front end load balancer nodes in the future if it's needed.

Keepalived uses virtual floating IPs to perform load balancing and failover tasks on the active and passive routers, while HAProxy performs load balancing and high-availability services to TCP and HTTP applications. All nodes running keepalived use the Virtual Redundancy Routing Protocol (VRRP) [8]. The active node sends VRRP advertisements at periodic intervals; if the backup nodes fail to receive these advertisements, a new active node is elected.

Keepalived performs failover on layer 4, upon which TCP conducts connection-based data transmissions. When a real server fails to reply to simple timeout TCP connection, keepalived detects that the server has failed and removes it from the server pool.

HAProxy offers load balanced services on layer 7, HTTP and TCP-based services, such as Internet-connected services and web-based applications. Depending on the load balancer scheduling algorithm chosen, HAProxy is able to process several events on thousands of connections across a pool of multiple real servers acting as one virtual server.

The scheduler determines the volume of connections and either assigns them equally in non-weighted schedules or given higher connection volume to servers that can handle higher capacity in weighted algorithms.

d) Test

In order to test the design, we are going to perform a simulation of client requests in order to test that the design works as designed. We are going to request the different html files that are stored in our Backend servers, by typing to any web browser the following URL address: 172.43.1.1, we are accessing through the virtual IP to the Master Front end node, thanks to the Keepalived software, the software in the master node sends periodic advertisements to the other Front end node. On the backup node, the VRRP instance determines the running status of the active node. If the active node fails to advertise after a previously configured interval, Keepalived initiates failover. In the frontend node, the HAProxy software will handle the request and will forward it to the backend server, performing layer 7 traffic load balancing. The backend server will deal with the response, which will be delivered to the client by the HAProxy node. The algorithm that the HAProxy is using to balance the traffic to the backend servers, in the current in the current configuration, is a Rounrobin algorithm.

By checking the logs, we can overview that the traffic from the client request and response goes through all phases anticipated, and we can corroborate that the implementation of the design works.

Thanks to the upgrades from the Front end from phase one, the internal implementation of our architecture is not exposed to the public internet. By having multiple nodes for both the front end and backend, high availability is developed, making the design more robust and reliable to overtake large amounts of client requests.

REFERENCES

[1] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[2] «Prototyping Model in Software Engineering: Methodology, Process, Approach». Reviewed 16th of September of 2019. https://www.guru99.com/software-engineering-prototyping-model.html.

[3] Repository of Hernan's Espinosa Work. https://github.com/HernanEspinosa/Web_Application_Firewall.git. Last Acces: October 2, 2019.

[4] Hat, R. (2019). *¿Qué es Docker?*. [online] Redhat.com. Available at: https://www.redhat.com/es/topics/containers/what-is-docker [Accessed 22 Sept. 2019].

[5] "What is a Container? | Docker", *Docker*, 2019. [Online]. Available: https://www.docker.com/resources/what-container. [Accessed: 15 Sept- 2019]

[6] «Productos de Elastic: Búsqueda, analíticas, logging y seguridad | Elastic». Reviewed 10th of September of 2019. https://www.elastic.co/es/products/.

[7] Docker Documentation. «Manage Data in Docker», Reviewed the 12th of October 2019 https://docs.docker.com/storage/.

[8] «7210-vrrp.pdf». Reveiwed the 7th of November of 2019. https://www.cisco.com/c/es_mx/support/docs/security/vpn-3000-series-concentrators/7210-vrrp.pdf.

APPENDIX Figure 1