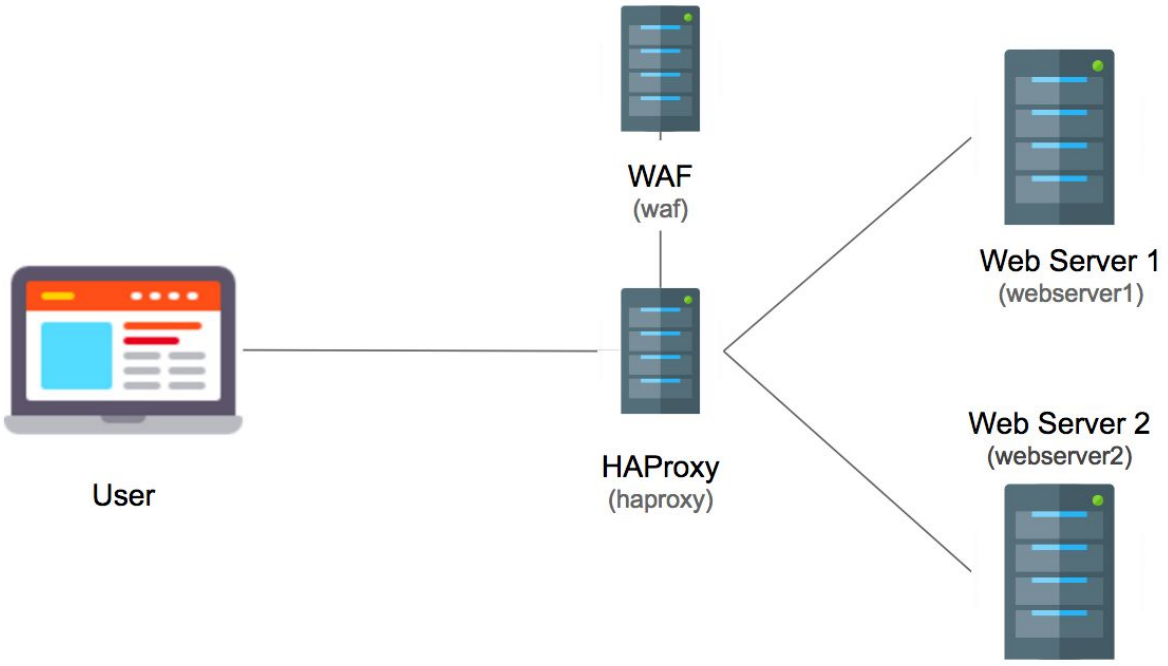
Trabajo Práctico - WAF

Integrantes:

- Melisa Rossi
- Nicolás Di Nucci
- María Victoria Cavo
- Franco Michel Zannini

Índice

Arquitectura General



La arquitectura general se compone de un server que funciona como load balancer (HAProxy), un server que funciona como firewall y proxy reverso (WAF) y por último los servers propiamente dichos. Las IP utilizadas para cada uno de los servers son las siguientes:

- HAProxy: 192.168.17.70 escuchando al cliente en el puerto 80
- WAF: 192.168.17.15 escuchando HAProxy en el puerto 80
- Web Server 1: 192.168.17.80 escuchando al HAProxy en el puerto 80
- Web Server 2: 192.168.17.90 escuchando al HAProxy en el puerto 80

Cada vez que un cliente realice una petición al HAProxy, este redirigirá la misma hacia el WAF, el cual aplicará una serie de filtros para detectar si dicha petición contiene código malicioso (XXS, SQL-Injection) o si se está realizando algún tipo de ataque (por ej: DoS). Si el WAF se encuentra en alguno de estos escenarios, entonces responderá al HAProxy con un código 403, el cual lo redirigirá al cliente.

Si dicha request es aprobada por el WAF, entonces se redirigirá la misma hacia uno de los servers (determinado por el HAProxy en su función de Load Balancer). Estos atenderán dicha request y enviarán la response correspondiente al HAProxy, el cual redirigirá la misma al cliente. Cabe aclarar que el web server 1 y 2, asumen que las requests que les llegan son seguras.

A fines prácticos, y para simplicidad de las pruebas, se decidió modelar esta arquitectura con máquinas virtuales utilizando Vagrant con Ubuntu Trusty 64.

Configuración de un servidor Node.js

Para la implementación de los servidores, se decidió utilizar Node.js.

Primero se debe instalar node. (Nota: Asegurarse de tener instalada la última versión de Python)

```
curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash -
sudo apt-get install nodejs
```

Luego, se debe levantar un servidor en el puerto correspondiente.

```
node server.js
```

Adicionalmente, se puede instalar un módulo para conectar una base de datos local al servidor. En el presente trabajo se utilizó una base de datos Postgres en conjunto con Sequelize como ORM, para poder realizar las pruebas de SQL Injection.

Primero instalamos postgres

```
sudo apt-get install libpq-dev postgresql-common postgresql -y
```

Creamos un nuevo rol dentro del esquema de la Base de Datos

```
sql_command="CREATE ROLE \"postgres\" WITH LOGIN SUPERUSER PASSWORD 'postgres';"
sudo sql_command="$sql_command" -Eu postgres bash -c 'psql -c "$sql_command"'
```

Creamos una base de datos a utilizar

```
sudo db_command="CREATE DATABASE mydb;" -Eu postgres bash -c 'psql -c "$db_command"'
```

Ahora ya estamos listos para crear las tablas dentro de nuestro esquema de base de datos

```
sudo db_command="CREATE TABLE mytable(...);" -Eu postgres bash -c 'psql -d mydb -c
"$db_command"'
```

Opcionalmente se puede instalar Sequelize

```
npm i sequelize
```

Al finalizar, se debería correr el servidor usando:

```
node server.js
```

Implementación de ModSecurity

Instalación de servidor apache con ModSecurity

ModSecurity funciona con un servidores apache. Por lo tanto, primero debemos poder instalar un servidor con estas características.

```
sudo apt-get install apache2 -y
```

Luego, instalamos ModSecurity con el siguiente comando y seteamos la configuración recomendada

```
apt-get install libapache2-modsecurity -y
```

Agregamos las reglas específicas que queremos que nuestro WAF aplique (en este caso, sql_injection, XSS y HTTP Policies)

```
+ln -s /usr/share/modsecurity-crs/base_rules/modsecurity_crs_30_http_policy.conf
/usr/share/modsecurity-crs/activated_rules/
+ln -s /usr/share/modsecurity-crs/base_rules/modsecurity_crs_41_xss_attacks.conf
/usr/share/modsecurity-crs/activated_rules/
+ln -s /usr/share/modsecurity-crs/base_rules/modsecurity_crs_41_sql_injection_attacks.conf
/usr/share/modsecurity-crs/activated_rules/
```

Hacemos que modsecurity actúe como proxy http reverso

```
sudo a2enmod proxy
sudo a2enmod proxy_http
```

Reiniciamos el server apache

```
service apache2 reload
```

Configuración de proxy reverso con ModSecurity

Configuramos el archivos httpd.conf ubicado en nuestra carpeta waf (donde se encuentra instalado HAProxy).

Definimos la IP del host

```
<VirtualHost 192.168.17.60:80>
```

Conservamos el header host

```
<IfModule mod_proxy.c>
        ProxyPreserveHost On
```

Deshabilitamos la opción de redirección de paquetes, ya que queremos que el WAF funcione como un proxy reverso.

```
ProxyRequests off
```

Deshabilitamos el header “Via”, ya que no será necesario en nuestros servidores

```
ProxyVia Off
```

Configuramos las páginas de los servidores y las páginas de error. Se define el error 403 como respuesta ante un potencial ataque

```
        DocumentRoot /var/www
        ErrorDocument 403 /error/error.html
```

No redireccionamos a los servers si identificamos un ataque

```
        ProxyPass /error/ !
```

Si no hay amenazas detectadas, redireccionamos hacia la interfaz de webservers_front

```
        ProxyPass / http://192.168.17.70:81/
        ProxyPassReverse / http://192.168.17.70:81/
</IfModule>
</VirtualHost>
```

Configuración de reglas para ModSecurity

Dentro del archivo modsecurity.conf definimos las reglas que se aplicarán sobre cada request. Importamos las reglas default de modsecurity

```
<IfModule security2_module>
        SecDataDir /var/cache/modsecurity
        IncludeOptional /etc/modsecurity/*.conf
        Include "/usr/share/modsecurity-crs/*.conf"
        IncludeOptional "/usr/share/modsecurity-crs/activated_rules/*.conf
</IfModule>
```

Dentro de mod_security.conf se encuentran definidas las reglas que aplican sobre cada request para evitar amenazas. Si se desea, dentro de este archivo se pueden incluir reglas propias.

Luego, debemos configurar nuestro proxy para que detenga las amenazas que le llegan (la configuración default implica solamente el log de la detección de la amenaza). En el archivo mod_security.conf, cambiamos el siguiente parámetro

```
        SecRuleEngine DetectionOnly
```

por

```
        SecRuleEngine On
```

Implementación de un servidor con HAProxy

HAProxy (High Availability Proxy) es un software gratuito que facilita y agiliza la distribución de carga de un servidor que ofrece servicios HTTP con conexiones TCP. Entre sus funciones, está la implementación de un proxy.

Instalación

En primer lugar, instalamos la última versión de haproxy
`/usr/bin/apt-get -y install haproxy`

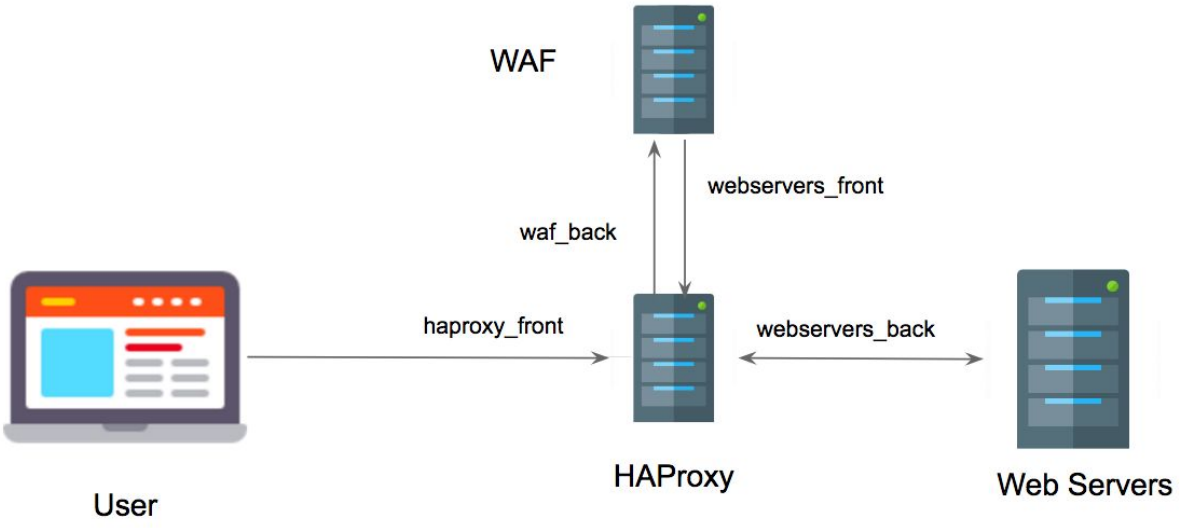
Luego, en el archivo `/etc/default/haproxy` agregamos la siguiente línea
`ENABLED=1`

Esto permite que HAProxy se inicie automáticamente cuando se enciende la computadora.

Configuración

Según la arquitectura planteada, se identifican 4 partes principales:

- **haproxy_front**: socket que atiende las conexiones de los clientes.
- **waf_back**: servidor WAF al que HAProxy se va a conectar para redirigir las conexiones de los clientes.
- **webservers_front**: atiende las conexiones que son aceptadas por el firewall.
- **webservers_back**: socket por el cual se re-dirigirán las requests de los clientes y los responses de los web servers.



Cada una de ellas contará con una configuración particular que se explica a continuación.

La configuración de HAProxy se realiza dentro del archivo `/etc/haproxy/haproxy.cfg`

En esta sección se define la configuración default de cada uno de los servidores. Estas reglas pueden ser sobreescritas, ya que la configuración más específica es la que predomina.

```
defaults
        mode http
```

Se limita el máximo tiempo de espera que HAProxy espera para que la conexión con el servidor resulte exitosa. Esta limitación es recomendable ya que previene que HAProxy esté intentando conectarse con un servidor que se encuentra caído

```
        timeout connect 5000ms
```

Se delimita a un máximo de 50000ms (50s), el tiempo que tiene un cliente para enviar un ACK o enviar información, luego de establecida la conexión.

```
        timeout client 50000ms
```

Configuración del frontend de HAProxy

Esta conexión será la que utilizarán los clientes para realizar requests a los servidores. Se hace un bind a una dirección IP y puerto específico.

```
frontend haproxy_front
        bind 192.168.17.70:80
```

Se re-dirigirán las conexiones a un backend que contiene un servidor configurado con Modsecurity, el cual analizará si dicha request contiene alguna amenaza para los servidores.

```
        default_backend waf_back
```

Configuración del backend del servidor WAF

Esta conexión será la que utilizará el HAProxy para redirigir los requests que atienda de los clientes. Se puede especificar la política de balanceo de carga.

```
backend waf_back
        balance roundrobin
```

Se suscribe al servicio waf a la IP y puerto indicado, con un límite a la cantidad de conexiones disponibles.

```
        server waf 192.168.17.60:80 maxconn 32 check
```

Configuración del frontend para requests aprobados por el WAF

Esta conexión será la que utilizará Modsecurity para redirigir los requests de los clientes que resulten seguros para nuestro WAF. Si una requests resultara segura, entonces será redirigida a los servidores. En caso contrario, enviará un código de error 403 (Forbidden) al cliente.

```
frontend webservices_front
```

```
        bind 192.168.17.70:81
        default_backend webservices_back
```

Configuración del backend para responder a los clientes

Esta conexión será la que utilizarán los servidores para enviar la respuesta hacia los clientes.

```
backend webservices_back
        balance roundrobin
        server web1 192.168.17.80:80 maxconn 32 check
        server web2 192.168.17.90:80 maxconn 32 check
```

Por último, hacemos un restart a HAProxy para que se aplique la configuración antes mencionada.

```
/usr/sbin/service haproxy restart
```

Ejemplo

En el siguiente repositorio se encuentra un ejemplo de implementación.

Para correrlo es necesario clonar el repositorio e instalar Vagrant.

Luego, abrir 4 terminales (una por cada servidor) y correr:
`vagrant up webserver1` en la terminal 1
`vagrant up webserver2` en la terminal 2
`vagrant up waf` en la terminal 3
`vagrant up haproxy` en la terminal 4