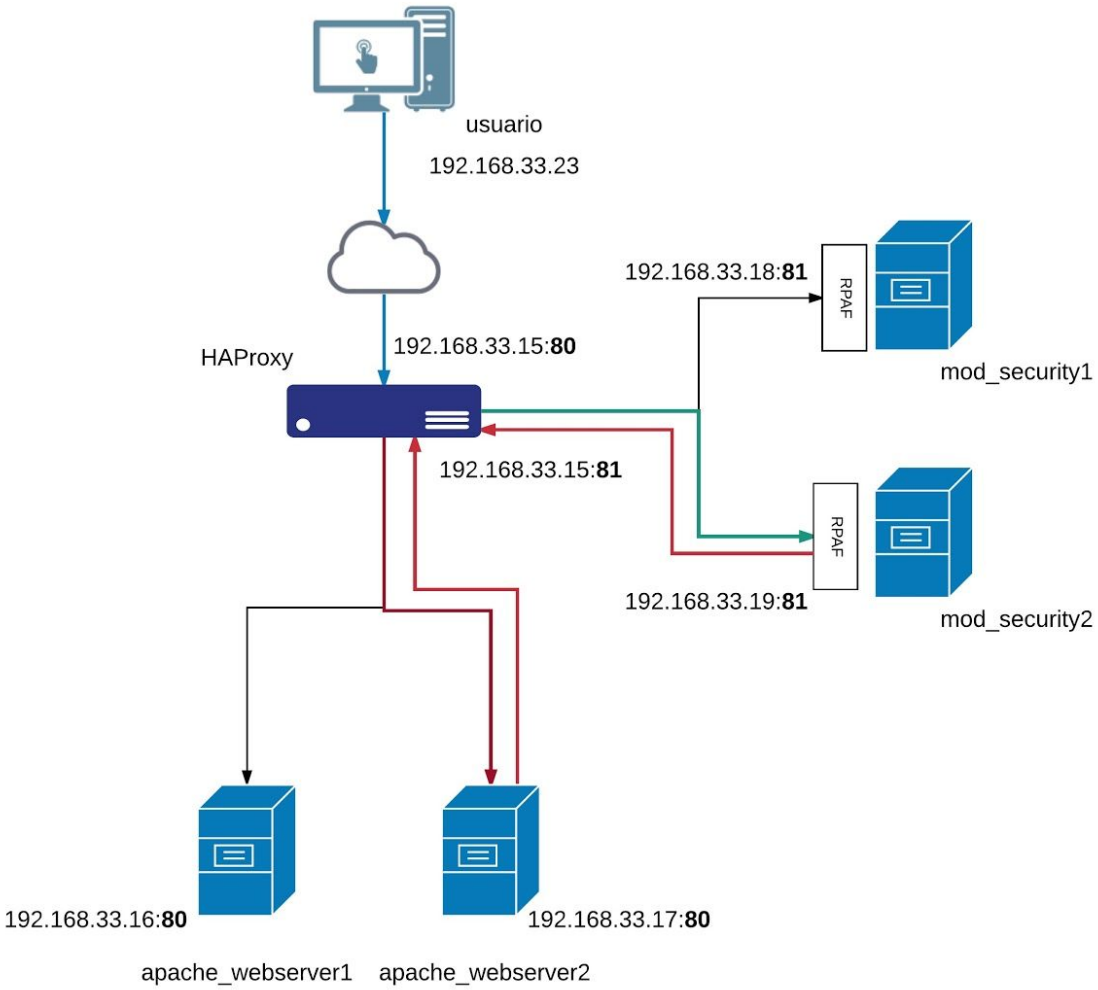
Instructivo de Instalación

Índice

Explicación general del sistema



Cuando un usuario realiza un http request el mismo será recibido por HAProxy en la interfaz 192.168.33.15 en el puerto 80. Se realizarán chequeos respecto a la cantidad de conexiones concurrentes y así como el ratio de apertura de conexiones por segundo por parte del cliente. Si se pasan los chequeos se forwardeará el http request a un servidor WAF, en este caso tendremos dos proxy reversos con ModSecurity con interfaces 192.168.33.18:81 y 192.168.33.18:81.

Los proxy reversos con ModSecurity verificarán si las request del cliente es parte de un ataque de tipo DOS, SQL Injection o XSS. Luego del análisis se comunicaran con HAProxy pero en la interfaz 192.168.33.15:81. Si el http request no paso los análisis de ModSecurity, HAProxy recibirá un http reply con un status code 403 que será forwardeado al cliente. En caso de haber pasado las verificaciones de los servidores WAF, HAProxy forwardeará el http request a los web servers.

Los web servers se encuentran implementan LAMP (Utilizan Linux, Apache, MySQL y PHP) y servirán los recursos que se les pidan. Asumen que cualquier request que se les haga es seguro. Los web server se encuentran en 192.168.33.16:80 y 192.168.33.17:80.

A continuación se comenta cómo realizar la instalación de un proxy reverso HAProxy, así como un proxy reverso con ModSecurity y un servidor web Apache.

En todos los casos se utilizó como sistema operativo Ubuntu 12.04 LTS.

Servidor Apache

Instalamos un servidor web apache

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

Instalamos MySQL. En la instalación se nos pedirá una contraseña para el administrador de la base de datos.

```
apt-get -y install mysql-server
```

Agregamos un módulo para *comunicar* php con MySQL

```
apt-get install -y libapache2-mod-auth-mysql php5-mysql
```

Ejecutamos un script que viene incluido en la instalación de MySQL para inicializar la base de datos

```
mysql_install_db
```

Segurizamos la base de datos

```
mysql_secure_installation
```

Para realizar la segurización se realizarán ciertas preguntas:

- Desea cambiar la contraseña?
- Eliminar usuarios anónimos? (Se recomienda eliminarlos)
- No permitir acceso del admin remotamente? (Se recomienda no permitirlo)
- Eliminar la base de datos de pruebas y acceder a ella? (Se recomienda hacerlo)
- Recargar las tablas de privilegios ahora? (Se recomienda hacerlo)

Instalamos php

```
apt-get install -y php5 libapache2-mod-php5 php5-mcrypt
```

Reiniciamos el servidor para que todas las configuraciones se apliquen.

```
service apache2 restart
```

Proxy reverso con ModSecurity

Instalamos un servidor web apache

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

Instalamos el módulo RPAF Apache, usado para crear un proxy reverso.

```
apt-get install libapache2-mod-rpaf
```

Configuración de Proxy Reverso con ModSecurity

Modificamos el archivo /etc/apache2/httpd.conf

Configuramos las IP y puertos donde va a escuchar el servidor.

```
Listen 192.168.33.18:81
NameVirtualHost 192.168.33.18:81
```

Dentro de las configuración del host.

```
<VirtualHost 192.168.33.18:81>
 ServerName *
 AddDefaultCharset UTF-8
```

Preservamos el header host.

```
ProxyPreserveHost On
```

No permitimos la funcionalidad de forwardear paquetes, debido a que el WAF funciona como un proxy reverso del HAProxy.

```
ProxyRequests off
```

No incluimos el header Via en la respuesta, pues no se utilizará en los servidores internos.

```
ProxyVia Off
```

Configuramos las páginas de web server, incluyendo una página default para el error 403.

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

Si buscamos un recurso en /error/ no forwardeamos al HAProxy.

```
ProxyPass /error/ !
```

Para el resto de los recursos, forwardeamos a la interfaz ft_web del HAProxy.

```
ProxyPass / http://192.168.33.15:81/
ProxyPassReverse / http://192.168.33.15:81/
```

Modificamos el archivo /etc/apache2/conf.d/mod_rpaf.conf, de esta forma configuramos un proxy reverso para ModSecurity, que solo acepta conexiones del HAProxy (RPAFproxy_ips) y reconstruye el paquete utilizando como IP origen lo que tenga el header X-Client-IP (RPAFheader).

```
<IfModule rpaf_module>
      RPAFenable On
      RPAFsethostname On
      RPAFproxy_ips 192.168.33.15 192.168.33.15
      RPAFheader X-Client-IP
</IfModule>
```

Instalamos los módulos de proxy a apache.

```
a2enmod proxy
a2enmod proxy_http
```

Instalamos el módulo de ModSecurity a apache.

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

Configuración de reglas para ModSecurity

Modificamos el archivo /etc/apache2/mods-enabled/mod-security.conf, que incluye todos los archivos .conf los cuales contienen las reglas a ser aplicadas.

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

En el archivo /etc/modsecurity/modsecurity.conf solo modificamos la línea *“SecRuleEngine DetectionOnly”* por *“SecRuleEngine On”*. Ya que con *“DetectionOnly”* sólo detecta y logea las amenazas, y con *“On”* las detecta, las logea y las detiene.

Agregamos el archivo /usr/share/modsecurity-crs/activated_rules/aloha.conf con una regla para que si se busca el recurso “/waf_health_check” responda con un 200. Esto es lo esperado por el HAProxy para considerar que el servidor está activo.

```
SecRule REQUEST_FILENAME "/waf_health_check"
                         "nolog,phase:1,status:200"
```

Agregamos el archivo /usr/share/modsecurity-crs/activated_rules/xss.conf con reglas para detectar cross-site scripting, el archivo se encuentra en:
HAProxy-Apache-ModSecurity/architecture/modsecurity/config/rules/xss.conf
Estas reglas lo que hacen es buscar patrones que se consideren amenazas en los parámetros que se reciben.

Estamos usando ModSecurity versión 2.6, y las reglas para detectar el cross-site scripting son para una versión superior, para que este sea compatible ejecutamos.

```
        perl HAProxy-Apache-ModSecurity/architecture/modsecurity/config/remove-2.7-actions.pl -t
2.6 -f /usr/share/modsecurity-crs/activated_rules/xss.conf
```

Para agregar las reglas que detecten el SQL Injection, utilizamos las provistas por OWASP al instalar el ModSecurity.

```
        ln -fs /usr/share/modsecurity-crs/base_rules/modsecurity_crs_41_sql_injection_attacks.conf
/usr/share/modsecurity-crs/activated_rules/
        ln -fs /usr/share/modsecurity-crs/base_rules/modsecurity_41_sql_injection_attacks.data
/usr/share/modsecurity-crs/activated_rules/
```

Estas reglas, al igual que las anteriores busca patrones que se consideren amenazas.

Reglas para ataques DOS

Agregamos el archivo /usr/share/modsecurity-crs/activated_rules/dox.conf con reglas para detectar un ataque de deny of service (DOS).

Contamos la cantidad de requests por ip por segundo

```
	SecRule REQUEST_BASENAME "!(\.avi$|\.bmp$|\.css$|\.doc$|\.flv$|\.gif$|\
\.htm$|\.html$|\.ico$|\.jpg$|\.js$|\.mp3$|\.mpeg$|\.pdf$|\.png$|\.pps$|\.ppt$|\.swf$|\
\.txt$|\.wmv$|\.xls$|\.xml$|\.zip$)"
"id:1000,phase:1,nolog,pass,initcol:ip=%{REMOTE_ADDR},setvar:ip.requests=+1,expirevar:ip.requests=1
"
```

Si hubo 5 requests o más en el último segundo desde dicha IP, se modifica la variable *block* a 1 (que expira en 5 segundos) y se aumenta la variable *blocks* (que expira en 120 segundos)

```
	SecRule ip:requests "@ge 5"
"id:1001,phase:1,pass,nolog,setvar:ip.block=1,expirevar:ip.block=5,setva
r:ip.blocks=+1,expirevar:ip.blocks=120"
```

Si la variable *block* es 1 (esto quiere decir que se realizaron 5 request de la misma IP en el último segundo), se le niega el acceso y retorna con un status 403.

```
SecRule ip:block "@eq 1" "id:1003,phase:1,deny,status:403"
```

Si la variable *blocks* es mayor o igual a 5 (esto quiere decir que hubo 5 bloqueos en los últimos 2 minutos), se logea la IP del request para posteriormente poder realizar algún tipo de bloqueo de dicha IP, se niega el acceso y se devuelve un status 403.

```
	SecRule ip:blocks "@ge 5"
"id:1002,phase:1,deny,log,logdata:'req/sec: %{ip.requests}, blocks:
%{ip.blocks}',status:403"
```

Finalmente, recargamos el servidor para guardar todas las configuraciones.

```
	service apache2 reload
```

HAProxy

Instalación HAProxy

Agregamos el repositorio con la versión 1.6 de HAProxy

```
apt-get install -y python-software-properties
add-apt-repository -y ppa:vbernat/haproxy-1.6
```

Actualizamos la lista de paquetes disponibles para posibilitar la instalación de HAProxy 1.6

```
apt-get update
```

Instalamos HAProxy 1.6

```
apt-get install -y haproxy
```

Modificamos el archivo /etc/default/haproxy para que quede con la siguiente configuración.

```
ENABLED=1
```

Esto permite que HAProxy se inicie automáticamente al iniciarse la computadora.

Configuración de HAProxy

Modificamos el archivo /etc/haproxy/haproxy.cfg para que quede con la siguiente configuración.

global define el servidor de Rsyslog que recibirá los logs de haproxy

```
global
      log 127.0.0.1 local1 notice
      user haproxy
      group haproxy
```

defaults específica definiciones que de no ser sobreescritas en las interfaces quedaran con los valores seteados en esta sección.

```
defaults
      log         global
      mode        http
```

Enable logging of HTTP request, session state and timers

```
option httplog
```

Si el servidor designado por la cookie está caído los clientes seguirán accediendo al mismo por las sticky sessions. Al habilitar la opción de redispatch el proxy romperá la persistencia de la cookie y enviará al cliente a un server que se encuentre en funcionamiento.

```
option redispatch
```

El cliente tiene un máximo de 10 segundo para realizar el acknowledge o enviar información.

```
timeout client  10s
```

Un ataque común consiste en que el cliente no se vuelve inactivo sino que lee muy lentamente la respuesta de servidor, obligándolo a mantener recursos para poder comunicarse con el cliente. Este ataque es conocido como SlowLoris y puede ser evitado obligado a que todo http-request hecho por el cliente debe terminar en 5 segundos, es decir, que hacer el request más leer la respuesta debe completarse en 5 segundos.

```
 timeout http-request 5s
```

Establece cual es el máximo tiempo que HAProxy esperara para que el intento de conexión con el servidor tenga éxito. Es una buena práctica pues el server podría caerse mientras HAProxy está tratando de conectarse al mismo.

```
timeout connect  5s
```

Establecemos el máximo tiempo que tiene un servidor WAF para responder el *Health Check*.

```
timeout server 25s
```

La interface listen stats habilita la recopilación de estadísticas por parte de HAProxy. En este caso particulas podrá ser accedido por 192.168.33.15:9000

```
listen stats
      bind 192.168.33.15:9000
      stats enable
```

Especificamos la URL de la pagina de stats a 192.168.33.15:9000/haproxy_stats

```
stats uri /haproxy_stats
```

Muestra el texto 'HAProxy Statistics' en la pantalla de autenticación

```
stats realm HAProxy Statistics
```

Define el user:password para acceder a la página de stats

```
stats auth admin:admin
```

Definimos la interfaz pública de HAProxy a través de la cual todos los clientes se conectaran.

```
frontend ft_waf
      bind 192.168.33.15:80
```

Creamos una tabla que almacenará una tupla por cada IPv4 que haga un request. Por cada tupla almacenaremos el número de de conecciones concurrente que el cliente tiene y la frecuencia con la que el cliente abre sockets en un periodo de 3 segundos.

```
stick-table type ip size 1m expire 5m store conn_cur,conn_rate(3s)
```

Si la cantidad de conexiones concurrentes que usuario tiene en mayor o igual a 10, se cortaran nuevas conexiones que se realicen. Los browsers generalmente abren muchas conexiones para descargar recursos estáticos por lo que la cantidad máxima debe basarse en el sitio que se está protegiendo para no empeorar la experiencia del usuario.

```
tcp-request connection reject if { src_conn_cur ge 10 }
```

Un ataque DOS simple consiste en abrir muchos sockets contra un servidor y cerrarlos rápidamente, con la siguiente instrucción podemos podemos limitar a que como máximo un cliente puede abrir 9 sockets en un periodo de 3 segundos. El periodo fue definido en la instrucción stick-table de la que se habló anteriormente.

```
tcp-request connection reject if { src_conn_rate ge 9 }
```

Incrementa los contadores que se almacenan en la stick table.

```
tcp-request connection track-sc1 src
```

Si una conexión pasa todas las condiciones anteriormente establecidas, se la redirige al backend WAF donde un servidor con ModSecurity se encargará de verificar si el cliente está realizando un ataque o no.

```
default_backend bk_waf
```

Configuración de backend de servidores WAF

Definimos el backend de servidores WAF que verificarán cada una de las conexiones para determinar si se realiza un ataque o no.

```
backend bk_waf
```

El balanceo roundrobin especifica que las conexiones se irán distribuyendo entre los servidores especificados en esta sección con un algoritmo *round robin*.

```
balance roundrobin
```

Cuando HAProxy forwardea un http request de un cliente a uno de los servidores WAF reemplaza la IP origen por la suya y agrega un header al http request con la IP del cliente.

```
option forwardfor header X-Client-IP
```

Para chequear si los servidores se encuentran activos se realiza un http request HEAD a la IP_SERVIDOR_WAF/waf_health_check.

```
option httpchk HEAD /waf_health_check HTTP/1.0
```

Especifica que la respuesta esperada para el chequeo de estado es un status code 400. Si se recibe un status code distinto de 200 se asume que el servidor WAF se encuentra inactivo.

```
http-check expect status 200
```

Con la siguiente instrucción establecemos que se realizan health checks cada 3s (inter 3s), si el servidor responde a dos chequeos correctamente entonces se considerará que está operando correctamente (rise 2), si falla no responde correctamente a tres chequeos consecutivos entonces se considerará que el servidor está caído.

```
default-server inter 3s rise 2 fall 3
```

Describimos la lista de servidores. Establecemos que un servidor WAF escuchará en la IP 192.168.33.18:81 en el puerto 81, que soportará como maximo 100 conexiones simulataneas. También establece un peso de 10, si esto se agrega el algoritmo round robin se modificará para que reparta las conexiones usando una distribución basada en los pesos.

El parámetro check especifica que este server debe ser chequeado de la forma que se describió anteriormente.

```
server waf1 192.168.33.18:81 maxconn 100 weight 10 check
```

Configuración de frontend para request chequeados por ModSecurity

Definimos la interfaz ft_web que será a la cual se conectan los servidores ModSecurity. En caso de que el request del cliente haya pasado todos los chequeos realizados se recibirá una request que será forwardeado al backend bk_web. Si no se hubieran pasados los chequeos ModSecurity enviará un 403 a HAProxy y este será forwardeado al cliente.

```
frontend ft_web
      bind 192.168.33.15:81

      default_backend bk_web
```

Configuración de backend de servidores web

Definimos la interfaz bk_web que funciona a modo de proxy reverso para los servidores web. Toda http request que llegue a este punto se la considera segura.

```
backend bk_web
      balance roundrobin
```

Por default HAProxy no mantiene sticky sessions. Con cookie logramos sticky sessions escribiendo en un header con el nombre del servidor al cual se redirigió el request http del cliente. De esta manera un cliente es forwardeado siempre al mismo servidor y no pierde su sesión.

```
cookie SRVNAME insert
```

Se listan los servidores Apache que responden a request http. El parámetro cookie <nombre de servidor> especifica el nombre que se insertará en el header antes descrito.

```
server node1 192.168.33.16:80 cookie server-1 check
```

Reiniciamos HAProxy para que utilice los cambios que realizamos.

service haproxy restart

Configuración de logs de HAProxy

Modificamos el archivo /etc/rsyslog.d/49-haproxy.conf para tenga la siguiente configuración.

```
$ModLoad imudp
```

El servidor Rsyslog para HAproxy se ejecutará localmente y escuchara en el puerto 514. Solo recibirá paquetes UDP.

```
$UDPServerAddress 127.0.0.1
$UDPServerRun 514
```

Definimos a *local1* como un alias del archivo donde se guardarán los logs de HAProxy. Por eso en la configuración de HAProxy se hace referencia a *local1*.

Reiniciamos el servidor Rsyslog para que utilice los cambios que realizamos.

```
service rsyslog restart
```