Developer: Hernan Espinosa Reboredo.
E-mail of contact: hespinosar@gmail.com.
Year 2020/21.

MANUAL FOR RUNNING WAF_v2 ARECHITECTURE

The following manual's intention is to help initialise WAF_v2 architecture, which stands for the second version of Hernan Espinosa's web application firewall design. The design can be found in the following repository: https://github.com/HernanEspinosa/Web_Application_Firewall.git. The manual is intended to work with Linux distributions and with the previous install of docker and docker-compose.

To build an image for the load balancer:

Containers are based in images, the images built for the load balancer are based in a DockerFile which can be edited to suit anybody's needs.

1. Clone the following GIT repository to your local machine:
   - https://github.com/HernanEspinosa/Web_Application_Firewall.git
2. After cloning the Git repository, go to /WAF_v2_Progression/WAF_v2.2/docker_Images/loadbalancer/WAF_v6
3. Type the following command in the terminal:
   - `docker build -f docker/lb/Dockerfile -t waf/loadbalancer .`

   This command will build a fresh image with the loadbalancing configuration in `docker/lb/Dockerfile.`
4. Typing `docker images ls`, will display all docker images in your local machine. The new docker image will show named as `waf/loadbalancer.`

To build and launch an image for the two Backend servers:

1. Clone the following GIT repository to your local machine:
   - https://github.com/HernanEspinosa/Web_Application_Firewall.git
2. After cloning the Git repository, go to /WAF_v2_Progression/WAF_v2.2/docker_Images/Backendservers
3. Type the following command in the terminal:
   - `docker compose up`

   This command will build two fresh images with the BackendServers configuration in `docker-compose.yml` file.
4. Typing `docker images ls`, will display all docker images in your local machine. The new docker image will show named as `httpd:2.4.`
5. Note that `docker compose up` will build the image and will run the docker containers, in this case, our backend-servers.

To build and launch an image for the two backend servers with PHP do the following:

```
docker run -d --name backenderver_x --hostname backendserver_x --
net=waf_public_net -p xx:xx --volume
/home/hernan/Documentos/Web_Application_firewall/WAF_v2_Progression/WAF_v2
.2/backendservers/backendserverXX/htdocs:/var/www/html php:apache
```

Replace the XX for your own requirements, as well as with the path for persistence.

Launching the load balancers:

To run the load balancers, follow these steps:

1. Create the directories in your local filesystem where the docker will mount it's volumes in order to make the architecture persistent.
2. Create the docker network, if already not created with `docker network create`, with the network specifications needed.

Developer: Hernan Espinosa Reboredo.
E-mail of contact: hespinosar@gmail.com.
Year 2020/21.

3. Run the following command:
   ```
   -  docker run -it --volume
      /home/hernan/Documentos/Web_Application_firewall/WAF_v2_Progressi
      on/WAF_v2.2/loadbalancers/loadbalancer1/haproxy:/etc/haproxy --
      volume
      /home/hernan/Documentos/Web_Application_firewall/WAF_v2_Progressi
      on/WAF_v2.2/loadbalancers/loadbalancer1/keepalived:/etc/keepalive
      d --name loadbalancer1 --hostname loadbalancer1 --
      net=waf_public_net -p 28080:8080 -p 28090:8090 --sysctl
      net.ipv4.ip_nonlocal_bind=1 --privileged waf/loadbalancer
   ```

   Beware to adjust the ports correctly so in the future, when starting the containers port conflict is avoided and volume persistence is enabled.
   In our configuration we have the following ports allocated.

   An example would be the following: a set up with 2 loadbalancers and 2 backendservers.

   LoadBalancer 1:
   -p 28080:8080 -p 28090:8090
   LoadBalancer 2:
   -p 29080:8080 -p 29090:8090
   BackendServer 1:
   -p 18080:80
   BackendServer 2:
   -p 19090:80

At this point, both load balancers and backend servers are running as expected.
Configure and Run HAProxy

After the load balancer container is launched, it will take you into the container.
1. Make sure haproxy has not been started with the following command:
   `service haproxy status`
   It should return something as `haproxy not running`
2. To start haproxy as a service run the following command:
   `service haproxy start`
3. Verify haproxy started successfully by monitoring logs:
   `tail -f /var/log/haproxy.log`
4. Input below URLs in browser to open the haproxy statistics report view:
   `http://localhost:28090/haproxy/stats`
   `http://localhost:29090/haproxy/stats`
   Input the predefined username and password, haproxy/passw0rd, when promted.
   Repeat the same steps in the other load balancer container.
   Note:
5. If you forget which load balancer container you are in, type `hostname` in the container.
6. If you exit the container for some reason, the container will be stopped as expected. To go back, e.g. `loadbalancer1`, run below command:
   `docker start -i loadbalancer1`

Configure and Run Keepalived for an ACTIVE – PASSIVE configuration

1. Go to the directory `/etc/keepalived`, where there are two sample configuration files for keepalived. One is for master node, and the other one is for backup node.
2. Run ping command in container, or below command on host machine to get all the IP addresses for the containers involved in the current network:
   ```
   -  docker inspect -f '{{.Name}} - {{range
      .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' $(docker ps -q)
   ```
   Then use a new IP address as the virtual IP address that does not conflict with others, e.g.:

Node	IP Address
Backendserver1	172.34.1.4
Backendserver2	172.34.1.5

Developer: Hernan Espinosa Reboredo.
E-mail of contact: hespinosar@gmail.com.
Year 2020/21.

Node	IP Address
Loadbalancer1	172.34.1.3
Loadbalancer2	172.34.1.2
virtual*	172.34.1.1

Here we use 172.34.1.1 as the virtual IP address. Replace `<your_virtual_ip>` with the actual value in both keepalived-master.conf and keepalived-backup.conf.
Note:
The value of interface defined in sample configuration files is eth0. It could be different depending on your system. To figure out the right value, you can run ip addr show.
Choose one load balancer container as master, e.g. mylb1.

3. Launch keepalived on master using master configuration:
```
keepalived --dump-conf --log-console --log-detail --log-facility 7 --vrrp -f /etc/keepalived/keepalived-master.conf
```
   In backup container, launch keepalived using backup configuration:
```
keepalived --dump-conf --log-console --log-detail --log-facility 7 --vrrp -f /etc/keepalived/keepalived-backup.conf
```
4. Verify keepalived started successfully by monitoring logs:
```
tail -f /var/log/syslog
```
5. Verify the virtual IP address is assigned successfully, run below command on master:
```
ip addr show eth0
```
   If configured correctly, you will see the virtual IP address appeared in the output.
   Run curl in either of the two load balancer containers.
6. Send request to the virutal IP and see if it returns the content retrieved from web servers, e.g. in the case where ssl is not enabled:
   curl -XGET http://<virtual_ip>:8080/index.html
   And ssl is enabled:
```
curl --insecure --cert /etc/ssl/certs/my.crt --key /etc/ssl/private/my.key -XGET https://<virtual_ip>:8443/healthz
```

Configure and Run Keepalived for an ACTIVE – ACTIVE configuration

Testing the environment

Backend servers

In this section we are going to perform some tests to see if the architecture designed works ok.
First, we are going to test if the statistics reports from the HAProxy servers work correctly. For any of the backend servers.

1. try to stop it of the with the command below:
```
docker stop backendserver1
```
2. Wait for a moment then check the haproxy statistics report in browser, e.g. use the below URL, to see if `backendserver1` is down:
```
http://localhost:28090/haproxy/stats
```
   Hit the `/index.html` endpoint exposed by load balancer either in browser or using `curl`. Make sure `backendserver1` will never be hit.
3. Start `backendserver1` again:
```
docker start backendserver1
```
4. Wait for a moment then check the haproxy statistics report in browser to see if `backendserver1` is up.
   Hit the `/index.html` endpoint again. Make sure both `backendserver1` and `backendserver2` will be hit.

Load balancers

Developer: Hernan Espinosa Reboredo.
E-mail of contact: hespinosar@gmail.com.
Year 2020/21.

In this section we are going to perform some tests to see if the architecture designed works ok.

1. Try to stop the master haproxy service within the container, e.g. `loadbalancer1`.
   `service haproxy stop`
2. Wait for a moment then check the keepalived logs by monitoring `/var/log/syslog` in both load balancer containers. See if `backserver1` entered into `BACKUP` state, and `backendserver2` transitioned to `MASTER` state.
   You can also verify it using `ip` command in both containers:
   `ip addr show eth0`
   If it works correctly, you will see the virtual IP address appeared in the output in container `backendserver2` instead of container `backendserver1`.
3. Run `curl` in either of the two load balancer containers. Send request to the virutal IP and see if it still returns the content retrieved from web servers.
4. Start `loadbalancer1` again:
   `service haproxy start`
5. Wait for a moment then check the keepalived logs. See if `backendserver1` gained `MASTER` state, and `backendserver2` returned to `BACKUP` state.
   You can also verify it using `ip` command in both containers, to see if the virtual IP address appeared in the output in container `backendservel` instead of container `backendserver2`.
6. Run `curl` in either of the two load balancer containers. Send request to the virutal IP and see if it still returns the content retrieved from web servers.

Log information

To view the logs, form the Docker containers direct to /var/lib/docker/containers and so on to overview the logs from each container. In this directory, you will find log in format .log or in format .json.

Security

1. RATE LIMITING

Rate limiting can be achieved by using the combination of ACL's, access control lists, stick tables and maps.

- A) Limiting the number of requests.
- B) Limiting the number of connections.
- C) Limiting the number of bytes flowing in.
- D) Limiting the number of bytes flowing out.
- E) Limiting the maximum amount of errors.

What happens if a client is exceeding the rate limit?

- Tarpit the client.
- Send the client to a different pool of servers.
- Ban the client for an extended period of time.

All rate limiting configuration is done in HAProxy_RateLimiting.cfg in both loadbalancers.

First, an HTTP Check is used when we want to verify that a specific website URL is healthy. If it returns a status an HTTP status 200 or 300 response, everything is working as it should. If not, HAProxy will consider the request as a bad health one and will mark the backend server with the status OFFLINE.

A) Setting the maximum number of requests:

In this scenario, we want to limit the number of requests that a user can make within a certain period of time. The period is a sliding window. So, if we set it to allow no more than 20 requests per client during the last 10 seconds, HAProxy will count the *last* 10 seconds. Since we are dealing with HTTP requests, and HTTP requests are stateless by design we are going to use stick tables to persist information from the client's behaviour. Stick tables are in-memory storage. Allow us to track client activities across requests, enable, server persistence, and to collect real time metrics. Stick tables can be used for:

- Server Persistence: Since each HTTP request is executed independently, without any knowledge of the requests that were executed before it we need some type of way to store pieces of information, such as IP address, cookie or ranges of bytes in the requested boy to associate it with a server. Using stick tables helps to forward the requests that share the same information of the request on to the same server. This way it can help in tracking user activities between one request and add a mechanism for storing events and categorizing them by client IP or other keys.

- Bot Detection: Stick tables can be used to prevent certain types of bot threads, brute force attacks, vulnerability scanners and many more threats.

- Collecting metrics: Without logging or having to parse logs, stick tables collect information from the activity of the load balancer

What does is keep track of whether an IP is abusive as well as its current request rate.We have their histori cal track record, as well as real-time behaviour. The directive creates a key-value store for storing counters, the key is configured by the type parameter, which will be used to store and aggregate the number of client requests by the IP address, we classify the data we'll be capturing by the IP. The store argument declares the values that we'll be saving. A stick table record expires and is removed after a period of inactivity by the client, as set by the expire parameter. That's just a way of freeing up space. Without an expire parameter, oldest records are evicted when the storage becomes full. Here, we're allowing 100,000 records.

```
backend Abuse
stick -Table type ip size 100K expire 30m store gpc0,http_req_rate(10s)
```

In the project we have created an Abuse stick table, which the rest of the configuration will refer to by the name of Abuse. The stick table is a lookup table for request data. Since we are using the IP as key, all requests following the same IP will refer to the same record, we keep track of the IP and its data. After 30 minutes of inactivity the table entries will expire. The table records store the general-purpose counter gpc0 and the IP's request rate of amount of times an IP has been marked as abusive. It will be called abuse indicator. What the abuse table does is to keep track of whether an IP is abusive as well as it's request rate.

```
frontend http-in
        bind *:8080
        # ACL function declarations
        acl is_abuse src_http_req_rate(Abuse) ge 10
        acl inc_abuse_cnt src_inc_gpc0(Abuse) gt 0
        acl abuse_cnt src_get_gpc0(Abuse) gt 0
        # Rules
        tcp-request connection track-sc0 src table Abuse
        tcp-request connection reject if abuse_cnt
        http-request deny if abuse_cnt
        http-request deny if is_abuse inc_abuse_cnt
        use_backend backendservers
```

An ACL(Access Control List) is a function declaration. The function is only invoked when used by a rule. In and of itself an ACL is nothing more than a declaration. Let's see all 3 of them in detail. Keep in mind that since all explicitly refer to the Abuse table which uses the IP as key, the functions are applied on the request's IP.

Developer: Hernan Espinosa Reboredo.
E-mail of contact: hespinosar@gmail.com.
Year 2020/21.

`acl is_abuse src_http_req_rate(Abuse) ge 10:` Returns true if the IP rate is equal to 10.
`acl inc_abuse_cnt src_inc_gpc0(Abuse) gt 0:` Returns true if the incremented value of gpc0 is greater that 0. The initial value of gpc0 is 0, the function always returns True. It increments the value of the abuse indicator, marking the IP as abusive.
`acl abuse_cnt src_get_gpc0(Abuse) gt 0:` ReturnsTrue if the value of gpc0 is greater than 0. Tells if the IP has already been marked as abusive.

The rules are not applied on incoming requests unless invoked by some rule.

It makes sense to take a look at the rules defined in the same `frontend` section. The rules are applied in turn on every incoming request and make use of the ACLs that we just defined. Let's see what each one does.

`tcp-request connection track-sc0 src table Abuse:` This rule adds the request IP to the table Abuse.
`tcp-request connection reject if abuse_cnt:` This rule will reject any new connection if tha IP has been marked as abusive. It is a way to ban a IP.
`http-request deny if abuse_cnt:` This rule denies access to a request if the IP has been marked as abusive, this rule will work for already established connections, but correspond to and IP that has been marked as abusive.
`http-request deny if is_abuse inc_abuse_cnt:` This rule denies acces to a request if a IP has a high request rate and is abusive.
The main difference between http-request and tcp-request is that http-request defines the rules which apply for level 7 processing and tcp request performs an action on an increasing connection depending on a layer 4 condition.

B) Setting the maximum number of connections:

```
Server WAF_backendserver1 172.34.1.4:80 ckeck inter 10s fall 3 rise 2
maxconn 30
```

We use the maxconn parameter on a server line to cap the number of concurrent connections that will be sent. Here's an example that sends up to 30 connections at a time to each server. After all servers reach their maximum, the connections queue up in HAProxy. This helps to keep the backndservers from overloading. We can also manage how long clients should be queued with the line `timeout queue 10 s`.
Establishing a window will prevent our servers from being buried under load, clients will reieve a 503 Service Unavailable error rather than waiting for the overloaded servers to respond. It's better to prevent any errors from our servers because of their overload.

Making Decisions based on stick tables

How to persist a client to a particular server, extracting the client's session ID from a cookie and storing it as the key in the table, the client will continue to be directed to the same server.

While on the topic of persistence, let us say we have a cluster of MySQL servers participating in master-master replication and we are worried that writing to one might cause a duplicate primary key if, at that moment, the primary master goes down and then comes back up. Normally this can make a rather complicated situation wherein both MySQL servers have some queries that the other doesn't and it requires a lot of fighting to get them back in sync. Suppose that instead we added the following to our MySQL backend?

Developer: Hernan Espinosa Reboredo.
E-mail of contact: hespinosar@gmail.com.
Year 2020/21.

Collecting information about traffic on our web applications so we can make decisions based on the information. If we want to know if perhaps disabling TLS 1.1 protocol is it safe, we could build a stick table that tracks the TLS versions that people are using.

To block requests that have made more than XX amount of requests over a time period defined on the stick table definition and by the key defined in the track line. Maybe we want to check if a request should be blocked without increasing the request counter by tracking it (so that a client can make 10 requests a second and everything above that gets blocked, rather than making 10 requests in a second and having future blocked requests keep them getting blocked until they cool down)