FEBRUARY 27, 2012 | BAPTISTE ASSMANN | 41 COMMENTS

We've seen recently more and more DOS and DDOS attacks. Some of them were very big, requiring thousands of computers…
But in most cases, this kind of attacks are made by a few computers aiming to make a service or website unavailable, either by sending it too many requests or by taking all its available resources, preventing regular users to use the service.
Some attacks targets known vulnerabilities of widely used applications.

In the present article, we'll explain how to take advantage of an **application delivery controller** to protect your website and application against DOS, DDOS and vulnerability scans.

Why using a LB for such protection since a firewall and a Web Application Firewall (aka WAF) could already do the job?
Well, the Firewall is not aware of the application layer but would be useful to pretect against SYN flood attacks. That's why we saw recently application layer firewalls: Web Application Firewalls, also known as WAF.
Well, since the load balancer is in front of the platform, it can be a good partner for the WAF, filtering out 99% of the attacks, which are managed by script kiddies. The WAF can then happily clean up the remaining attacks.
Well, maybe you don't need a WAF and you want to take advantage of your **Aloha** and save some money ;).

Note that **you need an application layer load-balancer**, like **Aloha** or OpenSource **HAProxy** to be efficient.

TCP syn flood attacks

The syn flood attacks consist in sending as many TCP syn packets as possible to a single server trying to saturate it or at least, saturating its uplink bandwith.

```
# Protection SYN flood
net.ipv4.tcp_syncookies = 1
net.ipv4.conf.all.rp_filter = 1
net.ipv4.tcp_max_syn_backlog = 1024
```

Note: If the attack is very big and saturates your internet bandwith, the only solution is to ask your internet access provider to null route the attackers IPs on its core network.

Slowloris like attacks

For this kind of attack, the clients will **send very slowly their requests to a server**: header by header, or even worst character by character, waiting long time between each of them.
The server have to wait until the end of the request to process it and send back its response.
The purpose of this attack is to prevent regular users to use the service, since the attacker would be using all the available resources with very slow queries.

In order to protect your website against this kind of attack, just setup the **HAProxy** option "**timeout http-request**".
You can set it up to 5s, which is long enough.
It tells **HAProxy** to let five seconds to a client to send its whole HTTP request, otherwise **HAProxy** would shut the connection with an error.

For example:

```
# On Aloha, the global section is already setup for you
# and the haproxy stats socket is available at /var/run/haproxy.stats
global
  stats socket ./haproxy.stats level admin

defaults
  option http-server-close
  mode http
  timeout http-request 5s
```

```
  stats uri     /
  stats realm   HAProxy Statistics
  stats auth    admin:admin

frontend ft_web
  bind 0.0.0.0:8080

  # Spalreadylit static and dynamic traffic since these requests have diffe
  use_backend bk_web_static if { path_end .jpg .png .gif .css .js }

  default_backend bk_web

# Dynamic part of the application
backend bk_web
  balance roundrobin
  cookie MYSRV insert indirect nocache
  server srv1 192.168.1.2:80 check cookie srv1 maxconn 100
  server srv2 192.168.1.3:80 check cookie srv2 maxconn 100

# Static objects
backend bk_web_static
  balance roundrobin
  server srv1 192.168.1.2:80 check maxconn 1000
  server srv2 192.168.1.3:80 check maxconn 1000
```

To test this configuration, simply open a **telnet** to the frontend port and wait for 5 seconds:

```
telnet 127.0.0.1 8080
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
HTTP/1.0 408 Request Time-out
Cache-Control: no-cache
Connection: close
Content-Type: text/html

<h1>408 Request Time-out</h1>
Your browser didn't send a complete request in time.

Connection closed by foreign host.
```

- too many connections opened
- new connection rate too high
- http request rate too high
- bandwith usage too high
- client not respecting RFCs (IE for SMTP)

How does a regular browser works?

Before trying to protect your website from weird behavior, we have to define what a “normal” behavior is!
This paragraphs gives the main lines of how a browser works and there may be some differences between browsers.

So, when one wants to browse a website, we use a browser: Chrome, Firefox, Internet Explorer, Opera are the most famous ones.
After typing the website name in the URL bar, the browser will look like for the IP address of your website.
Then it will establish a tcp connection to the server, downloading the main page, analyze its content and follow its links from the HTML code to get the objects required to build the page: javascript, css, images, etc…
To get the objects, **it may open up to 6 or 7 TCP connections per domain name**.
Once it has finished to download the objects, it starts aggregating everything then print out the page.

Limiting the number of connections per users

As seen before, a browser opens up 5 to 7 TCP connections to a website when it wants to download objetcs and they are opened quite quickly.
One can consider that somebody having more than 10 connections opened is not a regular user.
The configuration below shows how to do this limitation in the **Aloha** and **HAProxy**:

This configuration also applies to any kind of TCP based application.

```
5
6  defaults
7    option http-server-close
8    mode http
9    timeout http-request 5s
10   timeout connect 5s
11   timeout server 10s
12   timeout client 30s
13
14 listen stats
15   bind 0.0.0.0:8880
16   stats enable
17   stats hide-version
18   stats uri     /
19   stats realm   HAProxy Statistics
20   stats auth    admin:admin
21
22 frontend ft_web
23   bind 0.0.0.0:8080
24
25   # Table definition
26   stick-table type ip size 100k expire 30s store conn_cur
27
28   # Allow clean known IPs to bypass the filter
29   tcp-request connection accept if { src -f /etc/haproxy/whitelist.lst }
30   # Shut the new connection as long as the client has already 10 opened
31   tcp-request connection reject if { src_conn_cur ge 10 }
32   tcp-request connection track-sc1 src
33
34   # Split static and dynamic traffic since these requests have different im
35   use_backend bk_web_static if { path_end .jpg .png .gif .css .js }
36
37   default_backend bk_web
38
39 # Dynamic part of the application
40 backend bk_web
41   balance roundrobin
42   cookie MYSRV insert indirect nocache
43   server srv1 192.168.1.2:80 check cookie srv1 maxconn 100
44   server srv2 192.168.1.3:80 check cookie srv2 maxconn 100
45
46 # Static objects
47 backend bk_web_static
48   balance roundrobin
49   server srv1 192.168.1.2:80 check maxconn 1000
50   server srv2 192.168.1.3:80 check maxconn 1000
```

Testing the configuration

run an apache bench to open 10 connections and doing request on these connections:

```
ab -n 50000000 -c 10 http://127.0.0.1:8080/
```

Watch the table content on the haproxy stats socket:

```
echo "show table ft_web" | socat unix:./haproxy.stats -
# table: ft_web, type: ip, size:102400, used:1
0x7afa34: key=127.0.0.1 use=10 exp=29994 conn_cur=10
```

Let's try to open an eleventh connection using telnet:

```
telnet 127.0.0.1 8080
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Connection closed by foreign host.
```

Basically, opened connections can keep on working, while a new one can't be established.

Limiting the connection rate per user

In the previous chapter, we've seen how to protect ourselves from somebody who wants to open more than X connections at the same time.
Well, this is good, but something which may kill performance would to allow somebody to open and close a lot of tcp connections over a short period of time.
As we've seen previously, a browser will open up to 7 TCP connections in a very short period of time (a few seconds). One can consider that somebody having more than 20 connections opened over a

```
# On Aloha, the global section is already setup for you
# and the haproxy stats socket is available at /var/run/haproxy.stats
global
  stats socket ./haproxy.stats level admin

defaults
  option http-server-close
  mode http
  timeout http-request 5s
  timeout connect 5s
  timeout server 10s
  timeout client 30s

listen stats
  bind 0.0.0.0:8880
  stats enable
  stats hide-version
  stats uri     /
  stats realm   HAProxy Statistics
  stats auth    admin:admin

frontend ft_web
  bind 0.0.0.0:8080

  # Table definition
  stick-table type ip size 100k expire 30s store conn_rate(3s)

  # Allow clean known IPs to bypass the filter
  tcp-request connection accept if { src -f /etc/haproxy/whitelist.lst }
  # Shut the new connection as long as the client has already 10 opened
  tcp-request connection reject if { src_conn_rate ge 10 }
  tcp-request connection track-sc1 src

  # Split static and dynamic traffic since these requests have different im
  use_backend bk_web_static if { path_end .jpg .png .gif .css .js }

  default_backend bk_web

# Dynamic part of the application
backend bk_web
  balance roundrobin
  cookie MYSRV insert indirect nocache
  server srv1 192.168.1.2:80 check cookie srv1 maxconn 100
  server srv2 192.168.1.3:80 check cookie srv2 maxconn 100

# Static objects
```

have a negative impact for them. You can whitelist these IPs.

Testing the configuration

run 10 requests with ApacheBench, everything may be fine:

```
ab -n 10 -c 1 -r http://127.0.0.1:8080/
```

Using socat we can watch this traffic in the stick-table:

```
# table: ft_web, type: ip, size:102400, used:1
0x11faa3c: key=127.0.0.1 use=0 exp=28395 conn_rate(3000)=10
```

Running a telnet to run a eleventh request and the connections get closed:

```
telnet 127.0.0.1 8080
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Connection closed by foreign host.
```

Limiting the HTTP request rate

Even if in the previous examples, we were using HTTP as the protocol, we based our protection on layer 4 information: number or opening rate of TCP connections.
An attacker could respect the number of connection we would set by emulating the behavior of a regular browser.
Now, let's go deeper and see what we can do on HTTP protocol.

```
  stats socket ./haproxy.stats level admin

defaults
  option http-server-close
  mode http
  timeout http-request 5s
  timeout connect 5s
  timeout server 10s
  timeout client 30s

listen stats
  bind 0.0.0.0:8880
  stats enable
  stats hide-version
  stats uri     /
  stats realm   HAProxy Statistics
  stats auth    admin:admin

frontend ft_web
  bind 0.0.0.0:8080

  # Use General Purpose Couter (gpc) 0 in SC1 as a global abuse counter
  # Monitors the number of request sent by an IP over a period of 10 second
  stick-table type ip size 1m expire 10s store gpc0,http_req_rate(10s)
  tcp-request connection track-sc1 src
  tcp-request connection reject if { src_get_gpc0 gt 0 }

  # Split static and dynamic traffic since these requests have different im
  use_backend bk_web_static if { path_end .jpg .png .gif .css .js }

  default_backend bk_web

# Dynamic part of the application
backend bk_web
  balance roundrobin
  cookie MYSRV insert indirect nocache

  # If the source IP sent 10 or more http request over the defined period,
  # flag the IP as abuser on the frontend
  acl abuse src_http_req_rate(ft_web) ge 10
  acl flag_abuser src_inc_gpc0(ft_web)
  tcp-request content reject if abuse flag_abuser

  server srv1 192.168.1.2:80 check cookie srv1 maxconn 100
  server srv2 192.168.1.3:80 check cookie srv2 maxconn 100

# Static objects
backend bk_web_static
  balance roundrobin
```

Testing the configuration

run 10 requests with ApacheBench, everything may be fine:

```
ab -n 10 -c 1 -r http://127.0.0.1:8080/
```

Using socat we can watch this traffic in the stick-table:

```
# table: ft_web, type: ip, size:1048576, used:1
0xbebbb0: key=127.0.0.1 use=0 exp=8169 gpc0=1 http_req_rate(10000)=10
```

Running a telnet to run a eleventh request and the connections get closed:

```
telnet 127.0.0.1 8080
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Connection closed by foreign host.
```

Detecting vulnerability scans

Vulnerability scans could generate different kind of errors which can be tracked by **Aloha** and **HAProxy**:

- invalid and truncated requests
- denied or tarpitted requests
- failed authentications
- 4xx error pages

HAProxy is able to monitor an error rate per user then can take decision based on it.

```
    mode http
    timeout http-request 5s
    timeout connect 5s
    timeout server 10s
    timeout client 30s

listen stats
    bind 0.0.0.0:8880
    stats enable
    stats hide-version
    stats uri     /
    stats realm   HAProxy Statistics
    stats auth    admin:admin

frontend ft_web
    bind 0.0.0.0:8080

    # Use General Purpose Couter 0 in SC1 as a global abuse counter
    # Monitors the number of errors generated by an IP over a period of 10 se
    stick-table type ip size 1m expire 10s store gpc0,http_err_rate(10s)
    tcp-request connection track-sc1 src
    tcp-request connection reject if { src_get_gpc0 gt 0 }

    # Split static and dynamic traffic since these requests have different im
    use_backend bk_web_static if { path_end .jpg .png .gif .css .js }

    default_backend bk_web

# Dynamic part of the application
backend bk_web
    balance roundrobin
    cookie MYSRV insert indirect nocache

    # If the source IP generated 10 or more http request over the defined per
    # flag the IP as abuser on the frontend
    acl abuse src_http_err_rate(ft_web) ge 10
    acl flag_abuser src_inc_gpc0(ft_web)
    tcp-request content reject if abuse flag_abuser

    server srv1 192.168.1.2:80 check cookie srv1 maxconn 100
    server srv2 192.168.1.3:80 check cookie srv2 maxconn 100

# Static objects
backend bk_web_static
    balance roundrobin
    server srv1 192.168.1.2:80 check maxconn 1000
    server srv2 192.168.1.3:80 check maxconn 1000
```

```
ab -n 10 http://127.0.0.1:8080/diskfjkdsjlkfdsj
```

Watch the table content on the haproxy stats socket:

```
echo "show table ft_web" | socat unix:./haproxy.stats -
# table: ft_web, type: ip, size:1048576, used:1
0x8a9770: key=127.0.0.1 use=0 exp=5866 gpc0=1 http_err_rate(10000)=11
```

Let's try to run the same ab command and let's get the error:

```
apr_socket_recv: Connection reset by peer (104)
```

which means that HAProxy has blocked the IP address

Notes

- We could combine configuration example above together to improve protection. This will be described later in an other article
- The numbers provided in the examples may be different for your application and architecture. Bench your configuration properly before applying in production.

Related articles

- Fight spam with early talking detection
- Protect Apache against Apache-killer script
- Protect your web server against slowloris

Links

- HAProxy Technologies
- Aloha load balancer: HAProxy based LB appliance

41 THOUGHTS ON “USE A LOAD-BALANCER AS A FIRST ROW OF DEFENSE AGAINST DDOS”

dirigeant

FEBRUARY 27, 2012 AT 16:45

Does “timeout http-request” affect POST requeststst? 5 second may be too short for post requests. If it affects, is there any way to set timeout for only GET requests?

Baptiste Assmann

FEBRUARY 27, 2012 AT 16:53

Hi,

“timeout http-request” only applies to the header part of the request, and not to any data.
As soon as the empty line is received, this timeout is not used anymore.

Cheers

DmZ

DECEMBER 12, 2012 AT 20:44

And how I could force timeout on data part of request? My goal to timeout slow http post attack (http://www.darkreading.com/vulnerability-management/167901026/security/attacks-breaches/228000532/index.html)

Baptiste Assmann

DECEMBER 13, 2012 AT 09:26

Hi DmZ,

Latest HAProxy version (1.5-dev15) allows layer 7 tracking so there may be some things to do with it to protect against this type of attack.
That said I’m not sure HAProxy has yet all the features required for this type of protection.

dud

OCTOBER 23, 2015 AT 19:08

Hi.

Looks like it is no longer true today as per your article :
http://blog.haproxy.com/2015/10/14/whats-new-in-haproxy-1-6/
“Once enabled, the timeout http-request parameters also apply to the POSTED data.”

BTW I did not find anything regarding this change in the latest changelog nor the latest documentation does point this out :
“Note that this timeout only applies to the header part of the request, and not to any data.”

Could you please clarify this change and possibly point me to the relevant message in the changelog ?

Thanks.

Baptiste Assmann

OCTOBER 27, 2015 AT 23:46

I confirm the documentation has not been updated accordingly. It will be updated to match real HAProxy’s behavior.
Thanks for reporting.

Patrick Mézard

FEBRUARY 27, 2012 AT 22:51

“NOTE: if several users are hidden behind the same IP (NAT or proxy), this configuration may have a negative impact for them. You can whitelist these IPs.”

Not sure this is really practical for public web sites or mobile services. Or you have someone dedicated to whitelisting. He better be fast.

How do i get around this error

Baptiste Assmann

FEBRUARY 29, 2012 AT 10:22

Hi mandm,

You have to setup properly your stats socket in haproxy and point your socat to the socket path.

In the examples, the config file and the socket were in the same dir, which is not recommanded in produtcion. We usually configure the stat socket in /var/run.

Cheers

Ivan Skyz

FEBRUARY 29, 2012 AT 22:32

This is awesome, but is it possible to combine them all into one beautiful config under the same backend? I think I may speak for others when I find the syntax around the stick-table counters (gpc0) somewhat confusing.

Baptiste Assmann

MARCH 1, 2012 AT 07:27

Hi Ivan,

Yes it is possible.
I'll write this kind of conf in an other article, a bit later.
You can subscribe to the RSS stream or our tweeter account to get updated.

cheers

protection

MARCH 8, 2012 AT 23:17

Thanks very interesting blog!

Thomas

JUNE 13, 2012 AT 12:03

Running 1.5dev11p20120604 i need to specify the sticktable in the frontend to make the reject work using the example of “Limiting the HTTP request rate”, instead of line 29:
tcp-request connection reject if { src_get_gpc0(ft_web) gt 0 }

Vido

JUNE 27, 2012 AT 15:18

With which version of haproxy is this possible? latest 1.5, or is it possible in version 1.4 as well? I kinda hate using development versions on production servers.

Thanks

Baptiste Assmann

JUNE 27, 2012 AT 22:01

Hi,

All the examples are related to 1.5 (dev) branch.
You’re right, there are some 1.5 versions you should not use, like 1.5-dev9 and 1.5-dev10 😉
To be honest, 1.5-dev7 is very stable and we 1.5-dev11 looks quite stable but is still young.

cheers

Vido

JUNE 28, 2012 AT 10:46

I see… When can we expect stable version?

keepalive on the server side, which requires huge modification on HAProxy's core.
You can use 1.5-dev7 which is quite stable, I heard that the latest one, 1.5-dev11 is good as well.

cheers

Pingback: HTTP request flood mitigation | Exceliance – Aloha Load Balancer

Pingback: Scalable WAF protection with HAProxy and Apache with modsecurity | Exceliance – Aloha Load Balancer

Pingback: high performance WAF platform with Naxsi and HAProxy | Exceliance – Aloha Load Balancer

Smana

MAY 2, 2013 AT 18:03

Hello, first of all Good job 🙂 !
I just wanted to know how is it possible to reduce the number of lines if we want to use each of the configuration you proposed.
For example is it possible to have one line for the sticky-table like the following one ? (i know that one doesn't seem to work :p)

stick-table type ip size 1m expire 30s store gpc0,http_req_rate(10s),http_err_rate(10s) store conn_cur store conn_rate(3s)

Thanks,
Smana

Baptiste Assmann

LRaikhman (@lraikhman)

MAY 13, 2013 AT 19:05

Hi,

I want to implement the vulnerability scan detection (your last example) but want to exclude one IP address from the detection.

Can you help me to do that – is that possible?

Baptiste Assmann

MAY 14, 2013 AT 11:22

Hi,

Yes it is possible.
Look for the whitelisting options proposed in some of the configuration example.

Baptiste

Rfraile

JULY 17, 2013 AT 11:48

Hello Baptiste,
Why you define a “tcp_max_syn_backlog” with syncookies enabled?
In this situation, the backlog is not used because the aren’t any entry in that table, isn’t it?

Thanks,

Rfraile

JULY 17, 2013 AT 12:30

Nathan

MARCH 3, 2014 AT 11:16

Hello,

Thanks for your topic.
I have a warning in HAPROXY 1.5.-dev21 :

Starting HAproxy: [WARNING] 061/100656 (6315) : parsing acl keyword 'src_inc_gpc0(HTTP_FR_PHP55)' :
no pattern to match against were provided, so this ACL will never match.
If this is what you intended, please add '–' to get rid of this warning.
If you intended to match only for existence, please use '-m found'.
If you wanted to force an int to match as a bool, please use '-m bool'.

I don't understand because on a 1.5-dev19, i don't have this warning.

Thanks for your help

Baptiste Assmann

MARCH 15, 2014 AT 22:13

Hi,

Please send your question to the ML, including your configuration.

Baptiste

Pingback: Scalable WAF protection with HAProxy and Apache with modsecurity | HAProxy Technologies – Aloha Load Balancer

Pingback: high performance WAF platform with Naxsi and HAProxy | HAProxy Technologies – Aloha Load Balancer

For keep-alive clients it is also convenient to rate limit session rate.
Very nice writeup, thank you!

om

MAY 21, 2014 AT 07:51

Hi Can it be possible to throttle limit the HTTP Post ad Get method at HAProxy layer ?

Baptiste Assmann

MAY 21, 2014 AT 08:17

Yes of course.
With HAProxy, you have ACLs to match HTTP methods.

Baptiste

om

MAY 21, 2014 AT 08:56

Sorry, i am not getting the complete command. could you please provide me. i found following sample which block the http request if it not belongs to GET/POST/OPTIONS method:

```
acl missing_cl hdr_cnt(Content-length) eq 0
block if HTTP_URL_STAR !METH_OPTIONS || METH_POST missing_cl
block if METH_GET HTTP_CONTENT
block unless METH_GET or METH_POST or METH_OPTIONS
```

Best Regards
-Om

Tunisie annonces

NOVEMBER 28, 2015 AT 19:56

Baptiste Assmann

NOVEMBER 30, 2015 AT 07:54

Hi,

Yes, you can do this, using req.hdr(Referrer).

Alexey

APRIL 28, 2016 AT 09:43

Hi folks,
As far as i understand, src_http_err_rate does NOT counts 5xx http errors, right?
Currently I am under DOS attack which causes a lot of 500 errors from the web application, and error counter is incrementing only on 4xx.
Is there any option how can i track 5xx errors and do “tcp-request reject ” if greater than some value?
Thanks in advance!

Baptiste Assmann

MAY 2, 2016 AT 07:52

You’re right. This counter does not count HTTP 500 errors.
You may want to use gpc0, increase gpc0 on responses where status is greater or equal than 500.
and then decide to deny if the gpc0 inc rate is greater than some threshold.

Sean

AUGUST 4, 2016 AT 15:43

I have an HAProxy for a mail setup. Unfortunately sometimes accounts get hacked and we get a lot of spam and smtp connections coming through.

Usually, they are all from the same IP address. Would this config help me control it?

```
tcp-request connection reject if { sc1_conn_rate ge 20 }
tcp-request connection reject if { sc1_conn_cur ge 20 }
tcp-request connection track-sc1 src
acl local_ips src -f /etc/haproxy/trusted-ips.txt
use_backend smtp-be-local if local_ips
default_backend smtp-be-foreign

backend smtp-be-foreign
option smtpchk
source 0.0.0.0 usesrc clientip
server xxxx01 x.x.x.x:25 maxconn 1000 check port 25

backend smtp-be-local
option smtpchk
source 0.0.0.0 usesrc clientip
server xxxx02 x.x.x.x:25 maxconn 1000 check port 25
server xxxx03 x.x.x.x:25 maxconn 1000 check port 25
```

Any other suggestions?