Client Certificate Authentication with HAProxy

Using client certificates for security is a pretty cool idea! You can protect an entire application or even just a specific URI for only those that provide a valid client certificate. HAProxy will not only confirm the certificate is valid but also supports revoking certificates when compromised.

A brilliant and common use for this would be a corporate intranet or other internal application such as SharePoint. You can get an additional level of security via common methods like obfuscating the URL or adding additional passwords to the site. But if you want to avoid these, you can make use of SSL certificates to allow only clients who provide a non-expired or revoked certificate that was issued by your server access. Users will need to add this client certificate to their web browser's personal certificate store so that they can get access to either the whole site or specific parts of it.

The Certificate Revocation List (CRL) is key to making this security approach work with many users. Without the CRL, should a certificate become compromised you would need to re-issue the Certificate Authority (CA) and any client certificates. With a CRL, however, you can revoke a certificate - allowing sane user management for your backend application.

Locking an application

Here's a config example (reduced for simplicity) for locking down an entire application:

```
listen VIP
    bind 172.16.200.85:443 transparent ssl crt /etc/haproxy/cert/server.pem ca-file /etc/haproxy/cert/ca.crt verify required crl-file /etc/haproxy/cert/root_crl.pem
    ...
    server RIP 172.16.203.35:80  weight 100  cookie RIP  check port 80 inter 4000  rise 2  fall 2  minconn 0  maxconn 0  on-marked-down shutdown-sessions
```

With the above config, only a valid client certificate will gain you access to the site(s) behind "listen VIP". You'll notice I am using the statement “verify required” on the bind line. If you want to allow users without a client certificate to use this service you'll need to change that to “verify optional”.

verify options:

```
required - A client cert must be provided.
optional - Prompt for client cert but allow access without one.
none - Don't prompt for client cert, useless for web browsers but good for curl...
```

People with the client certificate can use the site as before - but the problem now is that anyone else can too! So, what if you need to allow those with the client certificate to access a specific URI, keeping it private from those without the certificate? Then you need a config like this instead:

```
listen VIP
    bind 172.16.200.85:443 transparent ssl crt /etc/haproxy/cert/server.pem ca-file /etc/haproxy/cert/ca.crt verify optional crl-file /etc/haproxy/cert/root_crl.pem
    ...
    acl restricted_path path_beg,url_dec -m beg -i /secure/
    http-request deny if restricted_path !{ ssl_c_used 1 } || restricted_path !{ ssl_c_verify 0 }
    server RIP 172.16.203.35:80  weight 100  cookie RIP  check port 80 inter 4000  rise 2  fall 2  minconn 0  maxconn 0  on-marked-down shutdown-sessions
```

Separating the configuration

The above config allows all users to access the whole site except the protected area “/secure”. Requests to `“https://<My-Site>/secure/<whatever>”` will result in a “403 Forbidden” error. You may consider using "verify optional" on your public-facing website - but be warned that users will be prompted to select a certificate when first accessing the site.

Most users wouldn't notice this as web browsers tend not to show the dialogue if you have no personal certificates installed. However, some users may consider seperating the configuration. This means that reqests to the secure part of the site can use "verify required" on one frontend/listen section while the unsecure part is on another.

The easiest way to handle this is by using a different IP, port or subdomain for the secure part of your site. For a more advanced method (using SNI so the secure and unsecure site can both be behind the same IP and port without users being prompted for a certificate) check out this excellent example by Lukas Tribus.

Custom error pages

You could also add custom error pages for revoked or expired certificates like so:

```
listen VIP
    bind 172.16.200.85:443 transparent ssl crt /etc/haproxy/cert/server.pem ca-file /etc/haproxy/cert/ca.crt verify optional crt-ignore-err all crl-file /etc/haproxy/cert/ro
    ...
    acl restricted_path path_beg,url_dec -m beg -i /secure/
    redirect location /certmissing.html if restricted_path !{ ssl_c_used 1 }
    redirect location /certexpired.html if restricted_path { ssl_c_verify 10 }
    redirect location /certrevoked.html if restricted_path { ssl_c_verify 23 }
    redirect location /othererrors.html if restricted_path !{ ssl_c_verify 0 }
    server RIP 172.16.203.35:80  weight 100  cookie RIP  check port 80 inter 4000  rise 2  fall 2  minconn 0  maxconn 0  on-marked-down shutdown-sessions
```

You'll note that I've added "crt-ignore-err all" to the bind line to allow invalid certificates access. You can then provide different error pages depending on the "ssl_c_verify" status. Client verify 10 represents an expired certificate while 23 is revoked. I had a 26 once during testing which I think meant the certificate had an invalid purpose, while 21 seems to return when you supply an entirely unrelated certificate.

Adding headers

Next, you can add in various headers containing client certificate information:

```
listen VIP
    bind 172.16.200.85:443 transparent ssl crt /etc/haproxy/cert/server.pem ca-file /etc/haproxy/cert/ca.crt verify required crl-file /etc/haproxy/cert/root_crl.pem
    ...
    http-request set-header X-SSL                       %[ssl_fc]
    http-request set-header X-SSL-Client-Verify         %[ssl_c_verify]
    http-request set-header X-SSL-Client-SHA1           %{+Q}[ssl_c_sha1]
    http-request set-header X-SSL-Client-DN             %{+Q}[ssl_c_s_dn]
    http-request set-header X-SSL-Client-CN             %{+Q}[ssl_c_s_dn(cn)]
    http-request set-header X-SSL-Issuer                %{+Q}[ssl_c_i_dn]
    http-request set-header X-SSL-Client-Not-Before     %{+Q}[ssl_c_notbefore]
    http-request set-header X-SSL-Client-Serial         %{+Q}[ssl_c_serial,hex]
    http-request set-header X-SSL-Client-Version        %{+Q}[ssl_c_version]
    server RIP 172.16.203.35:80  weight 100  cookie RIP  check port 80 inter 4000  rise 2  fall 2  minconn 0  maxconn 0  on-marked-down shutdown-sessions
```

This will add the following fetches to the traffic headers so it can be logged or even used by your backend application:

```
X-SSL: 1
X-SSL-Client-Verify: 0
X-SSL-Client-SHA1: .+......W....%0By%7F...d
```

```
X-SSL-Client-DN: /C=AU/ST=Some-State/O=Internet Widgits Pty Ltd
X-SSL-Client-CN: Bob Roberts
X-SSL-Issuer: /C=AU/ST=Some-State/O=Internet Widgits Pty Ltd
X-SSL-Client-Not-Before: 170531090028Z
X-SSL-Client-Serial: 02
X-SSL-Client-Version: 1
```

There's a lot more information that we can get if needed. The possible fetches are mentioned here in the HAProxy manual.

You can also use a certificate revocation list should a cert become compromised. For example, when employees move on, you can simply revoke their certificate, re-create the CRL PEM file and reload HAProxy to remove user access.

So we've discussed *what* you can do and *why* you might do it. Let's get to the *how*.

Creating certificates

First, you need to create the CA and client certificates followed by the certificate revocation list. I'm doing this on CentOS 6 in a fairly generic way, so it should work on most systems, including the Loadbalancer.org Enterprise appliance!

Create a directory for your CA and other certificate files under the HAProxy directory:

```
mkdir /etc/haproxy/cert
cd /etc/haproxy/cert
```

Create the CA which will be used for signing the client certificate:

```
openssl genrsa -out ca.key 4096
```

```
openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
```

Example answers:

```
Country Name (2 letter code) [AU]:UK
State or Province Name (full name) [Some-State]:HANTS
Locality Name (eg, city) []:Portsmouth
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Loadbalancer.org Ltd
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:Loadbalancer.org CA
Email Address []:
```

Create the OpenSSL configuration file:

```
nano -w ca.conf
```

Next paste the prepared and simplified config file:

```
[ ca ]
default_ca=     dft_ca  # Configuration files may have more than one CA
                        # section for different scenarios.

[ crl_ext ]
# issuerAltName=issuer:copy  #this would copy the issuer name to altname
authorityKeyIdentifier=keyid:always

[ dft_ca ]
DIR= ./                                 # Default directory.
unique_subject= no

certificate=    ${DIR}/ca.crt           # The CA certificate.
database=       ${DIR}/index.txt        # Keeps tracks of valid/revoked certs.
                                        # administrative purposes.

new_certs_dir=  ${DIR}/                 # Copies of signed certificates, for
                                        # administrative purposes.

private_key=    ${DIR}/ca.key           # The CA key.
serial=         ${DIR}/serial           # Should be populated with the next
                                        # cert hex serial.

default_md = sha1
```

```
# These govern the way certificates are displayed while confirming
# the signing process.
name_opt=       ca_default
cert_opt=       ca_default

default_days= 730       # How long to sign certificates for.
default_crl_days= 730   # The same, but for CRL.

policy=         dft_policy      # The default policy should be lenient.
x509_extensions= cert_v3        # For v3 certificates.

[ dft_policy ]
# A value of 'supplied' means the field must be present in the certificate,
# whereas 'match' means the field must be populated with the same contents as
# the CA certificate. 'optional' dictates that the field is entirely optional.

C=      supplied        # Country
ST=     supplied        # State or province
L=      optional        # Locality
O=      supplied        # Organization
OU=     optional        # Organizational unit
CN=     supplied        # Common name

[ cert_v3 ]
# With the exception of 'CA:FALSE', there are PKIX recommendations for end-user
# certificates that should not be able to sign other certificates.
# 'CA:FALSE' is explicitely set because some software will malfunction without.

subjectKeyIdentifier=   hash
basicConstraints=       CA:FALSE
keyUsage=               nonRepudiation, digitalSignature, keyEncipherment
basicConstraints=       CA:false
subjectKeyIdentifier=   hash
authorityKeyIdentifier= keyid:always
```

Now for a few extra requirements:

```
touch index.txt
echo 01 > serial
```

Next you create your first client certificate:

```
openssl genrsa -des3 -out client1.key 1024
openssl req -new -key client1.key -out client1.csr
openssl ca -batch -config ca.conf -notext -in client1.csr -out client1.crt
```

Example answers:

```
Country Name (2 letter code) [AU]:UK
State or Province Name (full name) [Some-State]:HANTS
Locality Name (eg, city) []:Portsmouth
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Loadbalancer.org Ltd
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:Bob Roberts
Email Address []:
```

Finally you can generate the CRL PEM file for HAProxy to use:

```
openssl ca -config ca.conf -gencrl -keyfile ca.key -cert ca.crt -out root_crl.pem
```

Assuming you've done that all correctly you should now be able to start HAProxy and test your setup:

```
curl --insecure --cert-type pem --cert client1.pem "https://172.16.200.85/secure/index.html"
```

Accessing the secure area without the client cert:

```
[root@lbmaster cert]# curl --insecure "https://172.16.200.85/secure/index.html"
Enter PEM pass phrase:
<html><body><h1>403 Forbidden</h1>
Request forbidden by administrative rules.
</body></html>
```

Accessing the secure area when providing the certificate you now gain access:

```
[root@lbmaster cert]# curl --insecure --cert-type pem --cert client1.pem "https://172.16.200.85/secure/index.html"
Enter PEM pass phrase:
TEST SECURE
```

Revoking a certificate:

```
openssl ca -config ca.conf -revoke client1.crt -keyfile ca.key -cert ca.crt
openssl ca -config ca.conf -gencrl -keyfile ca.key -cert ca.crt -out root_crl.pem
service haproxy reload
```

Let's try again with the newly revoked certificate:

```
[root@lbmaster cert]# curl --insecure --cert-type pem --cert client1.pem "https://172.16.200.85/secure/index.html"
Enter PEM pass phrase:
<html><body><h1>403 Forbidden</h1>
Request forbidden by administrative rules.
</body></html>
```

Any questions?

Finally, you may ask: how do I remove the certificate password, how do I make a PEM file as used in the examples, and how do I use this in a client's web browser?

The password was set when you created the private key, and can be removed, although it may be useful as an extra security layer in some circumstances. But I'll remove it for your use:

```
openssl rsa -in client1.key -out client1_nopass.key
```

The PEM file is also nice and simple to generate, as it's combined form of the .key and .crt files we created earlier. To combine these files into a PEM file, issue the following two commands:

```
cat client1_nopass.key > client1.pem
cat client1.crt >> client1.pem
```

Other than for testing, it may actually be more useful to create a PFX file that can be imported into your web browser:

```
openssl pkcs12 -export -clcerts -in client1.crt -inkey client1.key -out client1.p12
```

When it comes to using the client certificate in your web browser - many other bloggers have covered this. It's best done with lots of screenshots that I can't be bothered to make, so here's some links instead:

http://www.binarytides.com/client-side-ssl-certificates-firefox-chrome/
http://www.jscape.com/blog/firefox-client-certificate

After importing the certificate you will usually need to restart your web browser so that it will prompt you for the certificate. Equally, if you choose not to use a certificate when prompted, you often need to do the same to get the browser to ask you again!

Here are some links for further reading:

https://www.haproxy.com/blog/ssl-client-certificate-management-at-application-level/
https://raymii.org/s/tutorials/haproxy_client_side_ssl_certificates.html
https://www.haproxy.com/blog/ssl-client-certificate-information-in-http-headers-and-logs/