sethwebster / **GoDaddySSLHAProxy.md**

Created 3 years ago

Creating a PEM for HaProxy from GoDaddy SSL Certificate

GoDaddySSLHAProxy.md

GoDaddy SSL Certificates PEM Creation for HaProxy (Ubuntu 14.04)

1 Acquire your SSL Certificate

Generate your CSR This generates a unique private key, skip this if you already have one.

```
sudo openssl genrsa -out  etc/ssl/yourdomain.com/yourdomain.com.key 1024
```

Next generate your CSR (Certificate Signing Request), required by GoDaddy:

```
sudo openssl req -new -key /etc/ssl/yourdomain.com/yourdomain.com.key \
                  -out /etc/ssl/yourdomain.com/yourdomain.com.csr
```

note: Save all of these files and make sure to keep the *.key* file secure.

Send this to GoDaddy In the GoDaddy certificate management flow, there is a place where you give them the CSR. To get the contents of the CSR, open the CSR file in your favorite editor or:

```
cat /etc/ssl/yourdomain.com/yourdomain.com.csr
```

Once GoDaddy verifies the signing request, they will allow you to download the certificate.

Download this file, extract, and rename the file which is a series of letters and numbers followed by a *.crt* extension (eg. 5a3bc0b2842be632.crt) to *yourdomain.com.crt*. Send these files to your server.

2 Create Requried PEM for HAProxy**

HaProxy requires a *.pem* file formatted as follows:

1. Private Key (generated earlier)
2. SSL Certificate (the file that will be a series of numbers and letters followed by .crt, included in the zip you downloaded from GoDaddy)
3. CA-Bundle (gd_bundle-g2-g1.crt)

```
sudo cat yourdomain.key cat yourdomain.com.crt gd_bundle-g2-g1.crt > /etc/ssl/private/yourdomain.com.combined.pem
```

Configure HAProxy to use this new PEM

Example:

```
frontend www-https
   bind *:443 ssl crt /etc/ssl/private/yourdomain.com.combined.pem
   reqadd X-Forwarded-Proto:\ https
   default_backend www-backend
```

note: The values on the bind line should be correct for most use cases, but make sure the other lines are correctly configured for yours.

ransikafs commented on Mar 27, 2017 • edited

Hi

I have a concern regarding ssl being implemented. I tried to run the above implementation inside a docker container. My haproxy.cfg worked for http. After configuring for https, I am getting a 408 - timeout error. My config for frontend is as follows,

```
frontend haproxy_in
    bind *:443 ssl crt /etc/ssl/private/domain.pem
    reqadd X-Forwarded-Proto:\ https
    acl url_api path_beg /api
    use_backend api-backend if url_api
    acl url_login path_beg /login
    use_backend login-backend if url_login
```

Would you be able to tell me what i should look into in order to fix that.

Thank you.

matthieu-honel-v... commented on Apr 7, 2017

> sudo cat yourdomain.key **cat** yourdomain.com.crt gd_bundle-g2-g1.crt > /etc/ssl/private/yourdomain.com.combined.pem

I think this `cat` shouldn't be there.

thosuperman commented on Jan 19

I think the correct one is

```
sudo cat yourdomain.key yourdomain.com.crt gd_bundle-g2-g1.crt > /etc/ssl/private/yourdomain.com.combined.pem
```

baughj commented on Mar 2

Regarding the first step - nobody should be using 1024-bit keys for any purpose, ever, and a lot of CAs won't even sign requests using them. It should be 2048-bit or higher.

coolaj86 commented on Jul 12 • edited

> It should be 2048-bit or higher.

It should be 2048 RSA or 256 ECDSA. Some platforms won't allow 4096 RSA (i.e. Google App Engine).

Since 2048-bit keys are not 2x 1024-bit keys but rather 2^1024x, there is no need to use a higher key bit value and any fundamental break in RSA encryption will likely make all variants of the algorithm equally vulnerable. (i.e. P=NP is solved, an algorithm is discovered to generated prime numbers in O(n) time, or quantum computers are developed that can do actual math - as opposed to simulating beryllium hydride molecules)

Most likely someone will come up with another efficient encryption algorithm on par with ECDSA before any of that happens.