Ignition Guide

Let's Encrypt Guide for Ignition!

How the capabilities introduced in Ignition 8.0.3 could be used to integrate with the Let's Encrypt service in order to automate SSL certificate management.

(800) 266-7798
inductiveautomation.com

Guide written by: Joel Specht, Software Developer, Inductive Automation

Introduction

Ignition 8.0.3 introduces support for hot-reloading the Gateway's SSL key store. This capability enables Ignition to play well with services such as Let's Encrypt which provide for automatic SSL certificate management. This document walks through one example for integrating Ignition with Let's Encrypt.

Background

Before we move any further, it is important to have a basic understanding of SSL certificates, the ACME framework, and the Let's Encrypt Service.

In order to perform sensitive transactions on the web securely, we rely on Transport Layer Security (TLS), which supersedes Secure Sockets Layer (SSL). TLS allows for secure communications over an insecure network such as the Internet. SSL certificates are used as a part of TLS in order for a web server (such as Ignition's Gateway) to identify itself to a web client (such as your web browser) and to provide a key for encrypted communications. By default, your web browser will not trust any SSL certificate presented by a web server. Your web browser maintains a list of root certificates issued by third parties called Certificate Authorities (CAs). Your browser trusts these CAs and it is these CAs' responsibility to verify that an organization controls a domain.

When an organization wishes for web clients to trust their web server at their domain, they issue a Certificate Signing Request (CSR) and send this to a CA. This is like a person applying for a driver's license. The CA may perform background checks to verify that the organization is legit and truly controls the domain associated with their CSR. If everything checks out, the CA issues the SSL certificate binding the organization to their domain, just like a government agency issuing a person a valid driver's license certifying the person's identity and their right to use the roads.

Once this CA-signed SSL certificate is installed in the organization's web server, it is presented to any web client which connects to it. Since the web clients trust the CA and the CA signed the SSL certificate presented by the web server, the web client can trust the web server and will proceed to securely transact with it.

ACME is an acronym: **A**utomatic **C**ertificate **M**anagement **E**nvironment. ACME is detailed in RFC 8555. At a high level: ACME is an automated framework for obtaining and renewing SSL certificates for your domain so that you may enable SSL / TLS in your web server. This framework greatly simplifies the traditional SSL certificate lifecycle, which is cumbersome and varies from CA to CA.

Let's Encrypt is "a free, automated, and open certificate authority (CA), run for the public's benefit." Let's Encrypt accomplishes this by running an ACME server. Any domain administrator wishing to obtain free SSL certificates trusted by almost all platforms may stand up an ACME client which points at Let's Encrypt's ACME server.

Approach

Starting with Ignition 8.0.3, the Gateway fetches the SSL private key and certificates from the PKCS12 key store file located at **$IGNITION/webserver/ssl.pfx**. While the Gateway is running, it is possible to replace this key store file with one which contains new certificates. By default, the Gateway will check the file every 15 minutes and apply any changes which may occur, or you can use the following GCU command to reload the key store from disk instantly (assuming a Linux environment):

```
gwcmd.sh --reloadks
```

With this in mind, in order to automate the certificate renewal process, we have the following approach to automate the SSL certificate renewal process for Ignition:

1. Use an ACME client to obtain an SSL certificate from Let's Encrypt
2. Use a set of tools to convert the certificates and keys into a PKCS12 key store
3. Copy the new key store file to **$IGNITION/webserver/ssl.pfx**
4. Invoke the GCU's reload key store command to apply the new key store from disk

The general approach above may be scripted and scheduled using any set of tools you prefer.

ACME Client

I chose to use certbot as my ACME client. If you choose to do the same, consult the certbot documentation for information regarding how to install the software on your host. The examples below leverage certbot version 0.31.0 on Ubuntu 18.04.2 LTS.

In order for Let's Encrypt's ACME server to issue a new certificate, we must be able to demonstrate control over our domain name. The ACME server may perform either a DNS challenge or an HTTP challenge. In each type of challenge, the ACME server generates a random token. The ACME client takes this token and signs it in order to prove possession of the private key associated with the public key of the SSL certificate. In the case of the DNS challenge, the signed token is placed as a DNS TXT record. In the case of the HTTP challenge, the signed token is exposed as a resource on an HTTP server (port 80) accessible from a public IP pointed to by your domain name.

In my case:

1. I wish to perform an HTTP challenge
2. Ignition is bound to port 80
3. I do not wish to stop Ignition when renewing the SSL certificate

Ignition has direct support for exposing the ACME HTTP challenge. Simply create a file with the token value as the file name and the signed token as the file's contents in `$IGNITION/.well-known/acme-challenge/<token>`. The Gateway will return the file's contents when accessed from the HTTP resource `/.well-known/acme-challenge/<token>`, which is exactly what the ACME server needs to verify your control over the domain.

To accomplish the above, I ran the following certbot command:

```
certbot certonly --webroot
```

Follow the on-screen prompts to enter your email address, accept the terms, and enter your domain. When certbot asks you for the path to your web root, enter the path to your **$IGNITION** (root directory where the Gateway is installed). Certbot will automatically create the token file in `$IGNITION/.well-known/acme-challenge/<token>` for you and will remove the file once the ACME server verifies it or a failure occurs.

If you want to use a DNS challenge instead, there are plenty of plugins available for certbot. For example, there is a plugin which enables you to write the challenge using AWS's Route 53 services if you use AWS Route 53 as your DNS server.

If Ignition is not running on port 80 and you have no other service running on port 80, you could use certbot's `--standalone` option instead of `--webroot`. This will make certbot stand up its own HTTP server bound to port 80 for the duration of the HTTP challenge.

By default, certbot points to the live production Let's Encrypt services. You may want to play around with the `--test-cert` flag while learning / experimenting. This flag will point certbot to Let's Encrypt's staging services which are more lenient regarding resource limits. However, you will need to temporarily trust the fake staging root CA certificate while testing with staging certs. See Let's Encrypt's website for more details.

There are any number of other ways to perform the ACME challenge not mentioned here. The best option depends on your network architecture, requirements, and personal preference.

Creating the Key Store

At this point, it is assumed that you have obtained a CA-signed SSL certificate. If you used certbot in Ubuntu Linux as I did, the default location for the certificate chain and private key are as follows:

Certificate Chain: `/etc/letsencrypt/live/<your domain name>/fullchain.pem`

Private Key: `/etc/letsencrypt/live/<your domain name>/privkey.pem`

Note: While the above are the default locations for these artifacts, they may be overridden using command line arguments or a config file. See the certbot documentation for more details.

Let's reference the certificate chain in this document as **$CERT_CHAIN** and the private key as **$PRIV_KEY**.

You will also need the root CA certificate in order to create a valid SSL key store for Ignition as the Let's Encrypt certificate chain only contains the server cert and the intermediary CA cert. As of the time of this writing, Let's Encrypt is using DST Root CA X3 as the root CA cert. Copy its contents to a .pem file in a well known location on your server and let's reference it as **$ROOT_CA_CERT** in this document.

Next, we will build the full certificate bundle from the server cert to the intermediary CA cert to the root CA cert. To do this, I ran the following:

```
cat $CERT_CHAIN $ROOT_CA_CERT > $CERT_BUNDLE
```

$CERT_BUNDLE is the location of the certificate bundle .pem file we wish to create by concatenating the root CA cert at the end of the Let's Encrypt certificate chain.

Finally, we can build the PKCS12 key store using OpenSSL:

```
openssl pkcs12 \
     -export \
     -out $TEMP_KEYSTORE \
     -inkey $PRIV_KEY \
     -in $CERT_BUNDLE \
     -passout pass:$KEYSTORE_PWD \
     -name $KEYSTORE_ALIAS
```

$TEMP_KEYSTORE is the path to a temporary key store file where openssl will write the new key store

$PRIV_KEY and **$CERT_BUNDLE** are mentioned above.

$KEYSTORE_PWD is the password which will protect the key store and the private key. By default, Ignition uses the value "ignition". Note: it is generally not advised to pass plaintext passwords into a command line argument. Consult the openssl documentation for other options such as using an environment variable or file containing the password.

$KEYSTORE_ALIAS is the friendly name for the key store entry. By default, Ignition uses the value "ignition".

Copying the Key Store

At this point, you should now have the PKCS12 key store file with the certificate chain and private key intact located at **$TEMP_KEYSTORE**.

Next, if you have an existing SSL key store used by the Gateway, take a backup of it:
`cp $IGNITION/webserver/ssl.pfx $IGNITION/webserver/ssl.pfx.bk`

Now copy the new key store into Ignition:
`cp $TEMP_KEYSTORE $IGNITION/webserver/ssl.pfx`

Finally, clean up temporary files:
`rm $CERT_BUNDLE $TEMP_KEYSTORE`

Hot-Reloading the Key Store

At this point, your new SSL certificate is installed in a new key store located in **$IGNITION/webserver/ssl.pfx**. Now it is time for Ignition to start using the new key store:

```
$IGNITION/gwcmd.sh --reloadks
```

If you now visit your Gateway using https, you should see the new certificate and a lock or green bar indicating that the browser trusts the Gateway.

Automated Renewal

SSL certificates have a validity period just like a driver's license. Once an SSL certificate expires, a web browser will no longer trust the certificate and the web server presenting it. Therefore, administrators must renew their web server's SSL certificate before it expires. Let's Encrypt certificates in particular expire in around 90 days from the time they are issued.

Certbot may be run non-interactively using the -n command line argument. Many of the values entered in the interactive prompt may be passed by command line argument for automation. Here is an example:

```
certbot certonly \
     --webroot
     --webroot-path $IGNITION
     --domains $DOMAIN
     --agree-tos
     --deploy-hook $DEPLOY_SCRIPT
```

In the command above, **$IGNITION** is the Gateway's root installation directory. **$DOMAIN** is your domain name (such as example.com). **$DEPLOY_SCRIPT** is the path to a script which can be run once the new SSL certificate is generated.

You could then create a bash script at **$DEPLOY_SCRIPT** which does the following:

	$DEPLOY_SCRIPT
1	`#! /bin/bash`
2	
3	`# create cert chain bundle from server cert to root ca cert:`
4	`cat $CERT_CHAIN $ROOT_CA_CERT > $CERT_BUNDLE`
5	`# create the PKCS12 key store:`
6	`openssl pkcs12 -export -out $TEMP_KEYSTORE -inkey $PRIV_KEY -in $CERT_BUNDLE -passout pass:$KEYSTORE_PWD -name $KEYSTORE_ALIAS`
7	`# backup existing SSL key store if it exists:`
8	`[[ -f $IGNITION/webserver/ssl.pfx ]] && cp $IGNITION/webserver/ssl.pfx $IGNITION/webserver/ssl.pfx.bk`
9	`# copy the new SSL key store:`
10	`cp $TEMP_KEYSTORE $IGNITION/webserver/ssl.pfx`
11	`# clean up temp files:`
12	`rm $CERT_BUNDLE $TEMP_KEYSTORE`
13	`# hot-reload the new key store into the running Gateway:`
14	`$IGNITION/gwcmd.sh --reloadks`

Finally, here is a sample cron job which runs the certbot command with deploy hook every day at 5:13 p.m.:

```
13 5 * * * certbot certonly --webroot --webroot-path $IGNITION --domains $DOMAIN --agree-tos --deploy-hook $DEPLOY_SCRIPT >> $RENEW_LOG
```

$RENEW_LOG is the location of a log file where the output from the cron job will be written.

See the certbot documentation or consult certbot `--help` all for all certbot command line argument details.

Conclusion

My hope is that this document gives users an idea for how they may automate their SSL certificate management using Let's Encrypt and the new features in Ignition 8.0.3. The above approach is not the only way to integrate Ignition with Let's Encrypt and certainly may not be the "best" way to accomplish this, but it should illustrate the process any solution must take in order to achieve our goal.