Self-Hosted TRUST with your own Certificate Authority!

2023-06-14

#networking #security #homelab

TRUST. It’s what certificates are all about. How do we know that we can trust a server? We verify that the server has a certificate, and that the certificate is signed by someone we trust. That can be a well-known third party like Let’s Encrypt, or our own certificate authority. In this video, I’m going to cover the basics of setting up a root private key and signing certificates using OpenSSL, and running a certificate authority server. As a bonus, I’m using a Yubikey to store the certiicate authorities private keys, so they can’t be compromised without stealing the physical dongle (they CAN however be used to generate leaf certificates if the certificate authority is compromised). So follow along for a fun journey into the basics of setting up your public key infrastructure!

Contents

Video

A Bit About PKI

Public key encryption provides a secure method of transmitting data over insecure networks. It allows for secure communication by using a pair of keys: a public key for encryption and a private key for decryption. TLS (Transport Layer Security) certificates, which are based on public key encryption, ensure the authenticity and integrity of data exchanged between a server and a client. They provide trust and verification, protecting against unauthorized access, data tampering, and eavesdropping, thus establishing secure and encrypted connections.

2-layer vs 3-layer CA

- 2-layer
  - Root CA public trusted by users, private used to sign servers
  - Leaf CA generated via ACME ~daily from servers
- 3-layer
  - Root CA public trusted by users, private kept entirely offline
  - Intermediate CA public unused, private used to sign servers
  - Leaf CA generated via ACME ~daily from servers

Advantages to 2-layer:

- Simple to setup
- Single private key can be kept in multiple places (backups, HSMs, …)

Advantages of 3-layer:

- Root CA can issue a CRL which can revoke intermediate certificates (although we will not do that in this tutorial!)
- Intermediate CA can be issued for less time than root, so you can manually renew intermediate CA periodically so if it’s lost the time for exposure isn’t as high
- Root CA private key can be kept entirely offline, and reissue intermediate CA certs without keeping the private key in multiple palces

We will be setting up a 3-layer CA where the root keys are generated by OpenSSL and kept entirely offline (how you do that is up to you, this tutorial is already long enough) and the intermediate certificates are kepy on the Yubikey and used to sign server and user certificates.

Certificate Authority in OpenSSL

First, we are going to setup the certificate authority and generate our most precious private keys. To do this, we can use any Linux system with OpenSSL, such as Debian or Alpine. We also need the yubikey manager package installed. On Debian you can install this through apt with `apt install yubikey-manager` .

You do NOT need to use the same system as your eventual Certificate authority to generate these private keys! You can use an ephemeral system like a live image, as long as you can copy off the resulting certificates (public keys) for the CA and the root public key somewhere safe for later. For ease, I'm going to create a new directory in `/root/ca` on my eventual CA system to house the certificates and then delete them once all is done and backed up safe. Make sure you update any paths on your own system if you're using a USB drive for your private keys, or copy them out later.

This would also be a good time to set your Yubikey PINs. The defaults are what an idiot would use on their luggage.

Setup the OpenSSL CA Config

We don't need an intermediate CA config file since we are just generating the private key to be used by Smallstep, so certificates won't be signed by OpenSSL using the intermedate key.

Put the config file in `/root/ca/root.cnf` .

```
# OpenSSL root CA configuration file.
```

```
[ ca ]
# `man ca`
default_ca = CA_root

[ CA_root ]
# Directory and file locations.
dir               = /root/ca
certs             = $dir/certs
crl_dir           = $dir/crl
new_certs_dir     = $dir/newcerts
database          = $dir/index.txt
serial            = $dir/serial
RANDFILE          = $dir/private/.rand

# The root key and root certificate.
# Match names with Smallstep naming convention
private_key       = $dir/root_ca_key
certificate       = $dir/root_ca.crt

# For certificate revocation lists.
crlnumber         = $dir/crlnumber
crl               = $dir/crl/ca.crl.pem
crl_extensions    = crl_ext
default_crl_days  = 30

# SHA-1 is deprecated, so use SHA-2 instead.
default_md        = sha256

name_opt          = ca_default
cert_opt          = ca_default
default_days      = 25202
```

```
preserve          = no
policy            = policy_strict

[ policy_strict ]
# The root CA should only sign intermediate certificates that match.
# See the POLICY FORMAT section of `man ca`.
countryName             = match
organizationName        = match
commonName              = supplied

[ req ]
# Options for the `req` tool (`man req`).
default_bits        = 4096
distinguished_name  = req_distinguished_name
string_mask         = utf8only

# SHA-1 is deprecated, so use SHA-2 instead.
default_md          = sha256

# Extension to add when the -x509 option is used.
x509_extensions     = v3_ca

[ req_distinguished_name ]
# See <https://en.wikipedia.org/wiki/Certificate_signing_request>.
commonName                      = Common Name
countryName                     = Country Name (2 letter code)
0.organizationName              = Organization Name

[ v3_ca ]
# Extensions for a typical CA (`man x509v3_config`).
subjectKeyIdentifier = hash
```

```
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[ v3_intermediate_ca ]
# Extensions for a typical intermediate CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true, pathlen:0
keyUsage = critical, digitalSignature, cRLSign, keyCertSign
```

And also don't forget to create those directories:

```
mkdir -p /root/ca /root/ca/certs /root/ca/crl /root/ca/newcerts /root/ca/private
touch /root/ca/index.txt
echo 1420 > serial
```

Generate Root Certificates

Keep this *VERY SAFE*, preferably offline. Compromising the `root_ca_key` means any of your servers (and users with mutual TLS) can be impersonated.

```
# Generate a private key (needs a passphrase, don't forget the passphrase)
openssl genrsa -aes256 -out /root/ca/root_ca_key 4096
```

```
# Sign a 69-year (nice) certificate, to be used by clients mostly
openssl req -config /root/ca/root.cnf -key /root/ca/root_ca.key -days 25202 -new -x509 -sha256 -extensions v3_ca -
```

Congrats, we have a private key and 69-year certificate for it! Now we need a short(er) lived intermediate certificate. For this example, I'm going to use 10 years, although in an enterprise environment you'll probably want to go a lot shorter (2-5 years, or as often as you feel comfortable bringing out the root private key to offline-sign new certs and update your Yubikeys).

If you want to examine the key, you can use `openssl x509 -noout -text -in /root/ca/root_ca.crt`

Generate Intermediate Certificate

Similar to above, but we sign the public key with the root private key. We need to use a 2048 bit key for this to fit in the Yubikey's PIV storage (which does not support 4096 bit keys with the PIV app, but does with PGP)

```
# Generate a private key (needs a passphrase)
openssl genrsa -aes256 -out /root/ca/intermediate_ca_key 2048
# Generate a certificate-signing-request (CSR) for the intermediate CA key
openssl req -config /root/ca/root.cnf -new -sha256 -key /root/ca/intermediate_ca_key -out /root/ca/intermediate_ca.
# Sign the CSR with the root key
openssl ca -config /root/ca/root.cnf -keyfile /root/ca/root_ca_key -cert /root/ca/root_ca.crt -extensions v3_interr
```

If you want to examine the key, you can use `openssl x509 -noout -text -in /root/ca/intermediate_ca.crt`

Add Intermediate Key to Yubikey

In this phase, we take the intermediate CA key and import it to the Yubikey so we can use it. We also need to start the Yubikey service.

So, the commands:

```
# Instll Yubikey manager tools
apt install yubikey-manager

# Start Yubikey PCS service and enable it on boot
systemctl enable pcscd --now

# Add the intermediate CA keys in slot 9C
# You will need the passphrase for the intermediate private key
# There are other slots available, including all of the 'retired' slots if you want
# to use your Yubikey for other things and not overwrite slot 9C
ykman piv certificates import 9c /root/ca/intermediate_ca.crt
ykman piv keys import 9c /root/ca/intermediate_ca_key
```

If you are curious about the results, run `ykman piv info`

Install Smallstep

Setup Debian

Install Debian Bullseye (I know Bookworm just released), no GUI, with SSH although we can disable that later. Then login, make sure you are root - either login as root, or `sudo su` to get to a root prompt. Then `cd /root` so we can begin.

I'm using "tempest" as the name of my CA, located at `tempest.palnet.net` . DNS resolves to the IP address of my system locally, not externally, I don't own that domain name. Make sure you update the names in any configuration / examples I list with your own names.

As usual, once you're done installing run `apt update` and `apt full-upgrade -y` to make sure everything is up to date before continuing.

Install Smallstep CLI

Unfortunately their deb package doesn't include Yubikey support, so we have to compile that from source, but we can install the CLI from deb packages. If you don't want to use a Yubikey for your private keys, you can also instal the CA from deb packages and skip the whole compiling step below.

```
# Download step-cli from github (latest release is 0.24.2)
# We need to build step-ca from source to use Yubikey, but if you aren't using Yubikey you can go ahead
# and install the ca from deb packages as well.
#wget https://dl.smallstep.com/gh-release/certificates/gh-release-header/v0.24.2/step-ca_0.24.2_amd64.deb
wget https://dl.smallstep.com/gh-release/cli/gh-release-header/v0.24.4/step-cli_0.24.4_amd64.deb

# Install using apt
apt install ./*.deb -y
rm ./*.deb
```

Build Step-CA from Source

To build Step-CA, we need to install a recent version of the Go compiler, a few other tools and libraries we need to build Step-CA, to actually git clone and build step-ca, and install it. Here's the process:

```
# Add Backports to sources.list
cat >> /etc/apt/sources.list << EOF

#Backports repository
deb http://deb.debian.org/debian/ bullseye-backports main
deb-src http://deb.debian.org/debian/ bullseye-backports main
EOF

# And apt-update
apt update

# Install golang (this is the same version in Bookworm, fyi, 1.20 is in Sid still)
# And Git, so we can clone stuff
# And also some other packages we need to compile Smallstep
apt install golang-1.19-go git libpcsclite-dev gcc make pkg-config curl -y

# Add Go 1.19 to PATH for the next operations
export PATH="/usr/lib/go-1.19/bin:$PATH"

# Clone the git repo, move to it, and checkout the latest release (as of this writing, 0.24.2, but check Github)
cd /root
git clone https://github.com/smallstep/certificates.git
cd certificates
git checkout v0.24.2

# Build process
# Once you execute these, go get a coffee and come back, or maybe drive to get a snack, it'll be awhile
make bootstrap
# I know this sounds odd but I promise 'install' also does 'build', since it depends on the binary
# GOFLAGS are clear so it does a CGO build, which is required for Yubikey support
```

```
make install GOFLAGS=""

# This tells the kernel that step-ca can bind to service ports
setcap CAP_NET_BIND_SERVICE=+eip /usr/bin/step-ca
```

Now check version: `step version` and `step-ca version` to make sure they both run. `step-ca` should show a release time/date of now the time you actually built it, which should be now() but in UTC and not your local timezone.

Setup Step-CA

Step-CA usually wants to do its own PKI, so we are going to let it generate a new set of private keys and sign them, then delete them. So, that's fun. There aren't any options to *not* generate keys.

Anyway, commands for this phase:

```
# Create a new user for step-ca process, and a home in etc for it
mkdir -p /etc/step
export STEPPATH=/etc/step
useradd step
passwd -l step

# Run Step Init to create its folder structure and config
# Enter the password for your admin user ('provisioner')
step ca init --name="TEMPEST" --dns="tempest.palnet.net" --address=":443" --provisioner="apalrd" --deployment-type

# Copy the certificates (but not the private keys) from the root location to /etc/step and own them to step
```

```
cp /root/ca/root_ca.crt /root/ca/intermediate_ca.crt /etc/step/certs/
chown -R step:step /etc/step/
```

Next up, we need to edit `/etc/step/config/ca.json` to configure our Yubikey instead of internal PKI for this Step instance. Specifically, replace the `key` directive which points to a private key with a bit about it being a yubikey and selecting the parameters for the key management system (KMS). The diff looks like this:

```
    "root": "/etc/step/certs/root_ca.crt",
    "federatedRoots": null,
    "crt": "/etc/step/certs/intermediate_ca.crt",
-   "key": "/etc/step/secrets/intermediate_ca_key",
+   "key": "yubikey:slot-id=9c",
+   "kms": {
+           "type": "yubikey",
+           "pin": "123456"
+   },
    "address": ":443",
    "insecureAddress": "",
```

And now we can test it! `sudo -u step step-ca /etc/step/config/ca.json` , shouldn't give any errors and should sit and wait for requests to come in.

Add SystemD Service

Since we presumably want this to run all the time, this should be a service, and systemd is the thing that does services.

So, write out this service file, or copy and paste this all into the terminal to do it for you:

```
# The service script
cat > /etc/systemd/system/step-ca.service  << EOF
[Unit]
Description=Smallstep Certificate Authority

[Service]
User=step
Group=step
Environment="STEPPATH=/etc/step"
ExecStart=/usr/bin/step-ca /etc/step/config/ca.json

[Install]
WantedBy=multi-user.target
EOF

# Reload daemons and start now
systemctl daemon-reload
systemctl enable --now step-ca
```

Enable ACME Challenges

Now that our `step-ca` is up and running, we can enable the ACME provisioner for it. We are, finally, almost ready to issue certificates to our infrastructure! For this, we also need the `fingerprint` , which the ca prints when it starts up (get it from `systemctl status step-ca` and look for `X.509 Root Fingerprint`).

And the commands for this step:

```
# Make sure STEPATH is still set, since we need it for this phase
export STEPPATH=/etc/step
# Add the acme provisioner using our admin account
step ca provisioner add acme --type ACME --admin-name apalrd
# Restart the service
systemctl restart step-ca
```

Using your CA

Trust your Root

Here are some instructions on trusting your root certificate in Debian. Remember that the certificate is hosted by our ACME server, so we can just download it (although it's certificate won't yet be trusted, so we need to ignore certificate errors for now).

```
# As root, or prepend with sudo of course
wget --no-check-certificate https://tempest.palnet.net/roots.pem -O /usr/local/share/ca-certificates/tempest.crt
update-ca-certificates
```

You'll need to do this for every single computer which you use to access your sites, or you'll get a certificate error. Fun, right? But once you add the root certificate, then you can continue to add homelab services without needing to individually trust each one on each user's system.

Using your CA in Caddy

Here's an example Caddyfile which uses my local CA to issue a signed certificate. Also note that I first added the root certificate to the trust store on the local system, so I don't need to separately tell Caddy to trust the root certificate.

```
#Global options
{
        #Our local ACME server
        acme_ca https://tempest.palnet.net/acme/acme/directory
}

#A single server which will get a TLS certificate automatically
ca-testsvr.palnet.net {
        #All of the options here are left as defaults
        #But just say hello world for now
        respond "Hello, World!"
}
```

References

Here are some guides I used when writing this script, they might also be useful to you.

- Smallstep ACME Server on Raspberry Pi
- Smallstep ACME Provisoiner Documentation

- Smallstep Yubikey PIV Documentation
- OpenSSL Documentation