A private ACME compatible CA with HSM

May contain traces of nuts, buzzwords and AI generated images

seism0saurus

2. Oktober 2024

SEISMOSAURUS

Table of contents

Introduction

Who am I and why am I here?

At day

Ulrich Viefhaus

- DevOps Engineer and Information Security Officer at BRANDAD Solutions GmbH
- Certified Professional for Software Architecture Foundation Level
- Certified Application Security Engineer (Java)
- Certified Cloud Security Engineer
- Certified Ethical Hacker

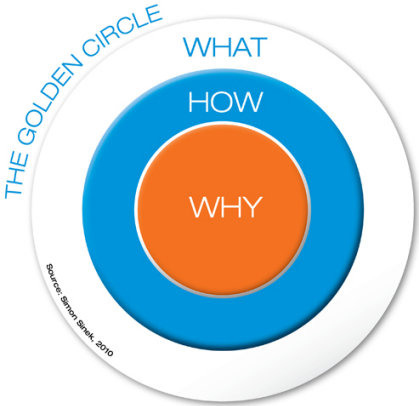
At night

seism0saurus

- Fighter against the imposter syndrom
- Blogger: seism0saurus.de
- Writer of Mastodon toots: @seism0saurus@infosec.exchange
- Vegetarian cook
- Father of three children
- Builder of OpenSource stuff
- Coorganisator of the Open Security Conference

Start with why

From Simon Sinek



[1]

Secure your servers

Simple nginx webserver

```
$ cat /etc/nginx/sites-enabled/default
server {
        listen 80 default_server;
        listen [::]:80 default_server;
        listen 443 ssl default_server;
        listen [::]:443 ssl default_server;
        ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
        ssl_certificate_key /etc/ssl/private/nginx-selfsigned.key;

        root /var/www/html;
        ...
}
```

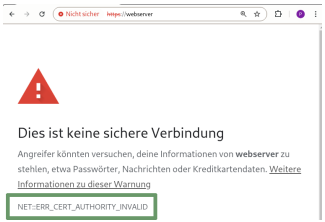
Untrusted

Webserver security issue

The webserver is reachable through https. Yeah!
But you get an error about the security. The cert authority is invalid, since it is a self-signed certificate and you can't trust every single one of them. Oh No!

Bad Practice

Don't teach your coworkers to accept invalid certificates! They will be unable to recognize MITM attacks.



PKI

Private Certificate Authority and PKI

Benefits of a private ACME compatible PKI

- Your only have to trust your private Root CA
- Automatic certificates for all your systems via ACME
- Omnipresent encryption for defense in depth
- Cost reduction

Definition

Buzzwords, bullshitbingo or important stuff?

Definitions

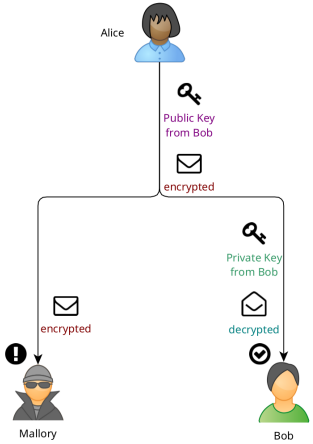
Assymetric Cryptography

public key cryptography

Public-key cryptography, or asymmetric cryptography, is the field of cryptographic systems that use pairs of related keys. Each key pair consists of a public key and a corresponding private key[2]

public key infrastructure

A public key infrastructure (**PKI**) is a set of roles, policies, hardware, software and procedures needed to create, manage, distribute, use, store and revoke digital certificates and manage public-key encryption[3]



Definitions

X.509

X.509 certificate

X.509 certificates bind an identity to a public key using a digital signature.[4]

X.509 CA certificate

A CA certificate can issue other certificates. The top level, self-signed CA certificate is sometimes called the Root CA certificate. Other CA certificates are called intermediate CA or subordinate CA certificates.[4]

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            04:83:c1:90:61:93:70:55:dd:88:99:b5:e2:ea:ab:13:02:2c
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = US, O = Let's Encrypt, CN = R11
        Validity
            Not Before: Aug 13 01:02:50 2024 GMT
            Not After : Nov 11 01:02:49 2024 GMT
        Subject: CN = seism@saurus.de
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (4096 bit)
                Modulus:
                    00:ec:3c:38:66:a4:eb:12:8f:ab:13:3d:67:cf:0e:
                    6b:01:26:ca:40:67:82:71:e0:48:c1:fb:7e:48:15:
                    51:dc:62:ff:df:73:e6:9c:75:e6:aa:68:2c:fb:14:
                    12:f8:51:0b:e4:3d:ce:9f:5b:94:f1:cd:9c:27:c9:
                    ec:5f:98:40:9e:0a:1c:8d:7f:7a:52:8f:11:aa:80:
                    44:00:cf:be:9a:48:1e:10:97:bf:94:ce:99:e9:2d:
                    5d:42:4d:49:ac:a9:5d:c6:49:20:be:53:9f:72:54:
                    23:b0:ef:70:ae:2e:7a:60:40:40:38:2e:db:c9:d2:
                    85:e6:7b:90:1c:81:24:06:30:5f:54:3d:c6:6f:2e:
                    e9:5f:5d:44:bd:07:a6:89:cc:9e:e1:f0:2e:83:6c:
                    0c:93:25:da:23:57:c9:33:58:0f:13:d0:54:46:9f:
                    ad:7a:19:55:9b:b3:c4:78:91:34:c6:45:ac:67:63:
                    3c:34:d6:c7:a6:1d:e1:89:cb:f8:7e:7c:3a:3f:4d:
```

Example

X.509 certificate

```
$ echo | openssl s_client -connect seism0saurus.de:443 -showcerts | openssl
    x509 -text -noout
...
Serial Number:
    04:83:c1:9b:61:93:70:55:dd:88:99:b5:e2:ea:ab:13:02:2c
Signature Algorithm: sha256WithRSAEncryption
Issuer: C = US, O = Lets Encrypt, CN = R11
Subject: CN = seism0saurus.de
Validity
    Not Before: Aug 13 01:02:50 2024 GMT
    Not After : Nov 11 01:02:49 2024 GMT
...
```

Definitions

X.509 continued

X.509 end-entity certificate

An end-entity certificate identifies the user, like a person, organization or business. An end-entity certificate *cannot* issue other certificates. An end-entity certificate is sometimes called a leaf certificate since no other certificates can be issued below it.[4]

Example

X.509 end-entity certificate

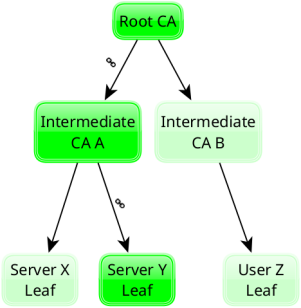
```
$ echo | openssl s_client -connect seism0saurus.de:443 -showcerts | openssl
    x509 -text -noout
...
X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature, Key Encipherment
            X509v3 Extended Key Usage:
                TLS Web Server Authentication, TLS Web Client Authentication
            X509v3 Basic Constraints: critical
                CA:FALSE
...
```

Definitions

X.509 continued

X.509 certificate chain

A certificate chain is a list of certificates (starting with an end-entity certificate) followed by one or more CA certificates (usually the last one being a self-signed certificate). Each certificate is signed by the next in the chain until you reach the trust anchor[5]



Example

X.509 certificate chain

```
$ echo | openssl s_client -showcerts -connect seism0saurus.de:443
 ...
depth=2 C = US, O = Internet Security Research Group, CN = ISRG Root X1
 verify return:1
depth=1 C = US, O = Lets Encrypt, CN = R11
 verify return:1
depth=0 CN = seism0saurus.de
 verify return:1
 ...
```

Definitions

Trust anchor

Trust anchor

In cryptographic systems with hierarchical structure, a trust anchor is an authoritative entity for which trust is assumed and not derived[6]

Automatic Certificate Management Environment

The ACME protocol is a communications protocol for automating interactions between certificate authorities and their users' servers, allowing the automated deployment of public key infrastructure at very low cost[7]

Lab environment

What systems do we need for our PKI?

Lab environment

VMs

Air Gap System

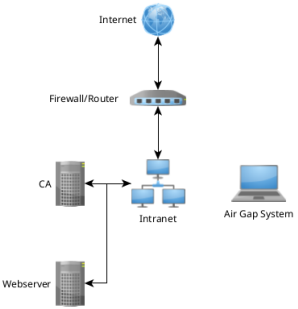
In reality this is a hardened and disk encrypted laptop you keep securely stored. On this system the Root CA is created

CA

This is our intermediate CA with smallstep ca and an HSM

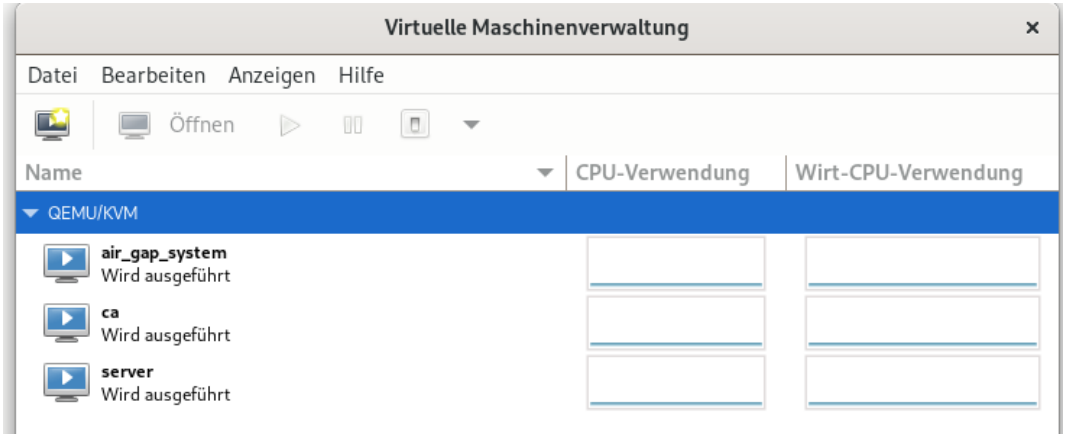
Server

This is an example http server, who wants a leaf certificate



Lab environment

Demo



Root CA

How do we create the root of our public key infrastructure in a secure way?

Offline

The Root CA has to be as secure as possible

Reasons

For an offline Root CA with a HSM

- Eliminating online attack vectors[8]
- Reducing site channel attack vectors
- Reducing risk of configuration errors
- Secure offsite backup[9]
- n-of-m schemes[10]

Air Gap System

Demo inspired by NitroKey docs[11]

```
osco@air-gap-system:~$ tree /opt/certificate-authority
/opt/certificate-authority
├── certs
│   └── root.crt
├── config
│   ├── create_intermediate_csr.ini
│   ├── create_root_cert.ini
│   └── sign_intermediate_csr.ini
├── crl
├── index.txt
├── index.txt.attr
```

Intermediate CA

Why do we need the indirection of an intermediate CA?

Offline

The root CA is offline for security reasons. Someone has to sign the requests

Reasons

For an intermediate CA with a HSM

- Root CA can stay offline
- Damage done by a break of the intermediate CA is less severe
- Reducing site channel attack vectors
- step ca has HA mode available[12]
- step ca has ACME support[13]

Intermediate CA

Demo

```
osco@ca:~$ systemctl status step-ca.service
● step-ca.service - step-ca service
     Loaded: loaded (/etc/systemd/system/step-ca.service; enabled; preset: enabled)
     Active: active (running) since Tue 2024-10-01 07:56:27 BST; 6h ago
       Docs: https://smallstep.com/docs/step-ca
             https://smallstep.com/docs/step-ca/certificate-authority-server-production
   Main PID: 722 (step-ca)
      Tasks: 8 (limit: 1098)
     Memory: 23.4M
        CPU: 1min 7ms
     CGroup: /system.slice/step-ca.service
             └─722 /usr/local/bin/step-ca config/ca.json
```

Leaf system

How does this protect our servers?

Private PKI with ACME

Not only Let's Encrypt

- Enables provisioning of server certificates
- Authentication through Challenges
- Supported by Smallstep CA as one of the provisioners
- Many tools for different operating systems like acme.sh[14]
- Only the root certificate is needed on clients

Webserver

Demo

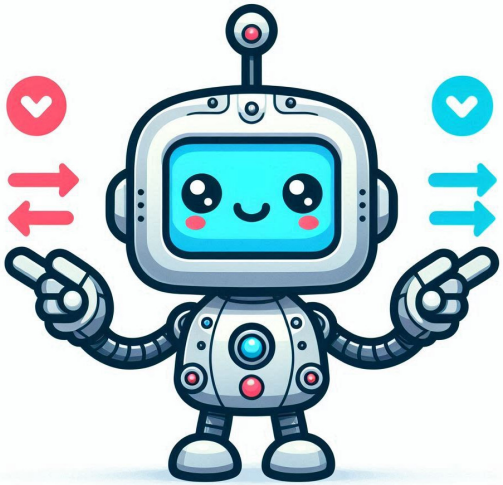
```
osco@webserver:~$ cat /etc/nginx/sites-enabled/default
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        listen 443 ssl default_server;
        listen [::]:443 ssl default_server;

        ssl_certificate /etc/ssl/certs/webserver.crt;
        ssl_certificate_key /etc/ssl/private/webserver.key;
```

Alternative use cases

What else can you do with your PKI?



PKI

Other use cases

- JWK provisioner for getting certificates programmatically[15]. You can provision client certificates for mutual TLS
- OIDC provisioner for certificates for users[16]. You can provision user certificates for email or authentication
- SSH provisioner for ssh certificates for clients[17]. Grant short lived certificates and authenticate servers and clients

Thank you!

Ressources available at
https://seism0saurus.de/posts/private-acme-ca-hsm/

References I

[1] *Graphic from Simon Sinek - CC BY 2.0*. 2024. URL: https://creativecommons.org/licenses/by/2.0/ (visited on 09/30/2024).

[2] *Public-key cryptography*. 2024. URL: https://en.wikipedia.org/wiki/Public-key_cryptography (visited on 09/12/2024).

[3] *Public key infrastructure*. 2024. URL: https://en.wikipedia.org/wiki/Public_key_infrastructure (visited on 09/12/2024).

[4] *X509 - Certificates*. 2024. URL: https://en.wikipedia.org/wiki/X.509#Certificates (visited on 09/12/2024).

References II

[5] *X509 - Certificate chains and cross-certification*. 2024. URL: https://en.wikipedia.org/wiki/X.509#Certificate_chains_and_cross-certification (visited on 09/12/2024).

[6] *Trust anchor*. 2024. URL: https://en.wikipedia.org/wiki/Trust_anchor (visited on 10/01/2024).

[7] *Automatic Certificate Management Environment*. 2024. URL: https://en.wikipedia.org/wiki/Automatic_Certificate_Management_Environment (visited on 10/01/2024).

[8] *Offline root certificate authority*. 2024. URL: https://en.wikipedia.org/wiki/Offline_root_certificate_authority (visited on 10/01/2024).

References III

[9] *NitroKey - Importing Keys And Certificates*. 2024. URL: https://docs.nitrokey.com/hsm/linux/import-keys-certs (visited on 10/02/2024).

[10] *N-of-m Schemes*. 2024. URL: https://docs.nitrokey.com/hsm/linux/n-of-m-schemes (visited on 10/02/2024).

[11] *Creating a Certificate Authority*. 2024. URL: https://docs.nitrokey.com/hsm/linux/certificate-authority (visited on 09/30/2024).

References IV

[12] *Step v0.8.3: Federation and Root Rotation for step Certificates*. 2024. URL: https://smallstep.com/blog/step-v0.8.3-federation-root-rotation/ (visited on 09/30/2024).

[13] *ACME Basics*. 2024. URL: https://smallstep.com/docs/step-ca/acme-basics/ (visited on 10/02/2024).

[14] *acme.sh*. 2024. URL: https://github.com/acmesh-official/acme.sh (visited on 10/02/2024).

[15] *Step CA - Provisioners - JWK*. 2024. URL: https://smallstep.com/docs/step-ca/provisioners/#jwk (visited on 09/30/2024).

References V

[16] *Step CA - Provisioners - OAuth/OIDC Single Sign-on*. 2024. URL: https://smallstep.com/docs/step-ca/provisioners/#oauthoidc-single-sign-on (visited on 09/30/2024).

[17] *Step CA - Provisioners - SSHPOP - SSH Certificate*. 2024. URL: https://smallstep.com/docs/step-ca/provisioners/#sshpop---ssh-certificate (visited on 09/30/2024).