Be your own certificate authority

Originalartikel

Backup

<html> The Transport Layer Security (TLS) model, which is sometimes referred to by the older name SSL, is based on the concept of certificate authorities (CAs). These authorities are trusted by browsers and operating systems and, in turn, sign servers' certificates to validate their ownership. However, for an intranet, a microservice architecture, or integration testing, it is sometimes useful to have a local CA: one that is trusted only internally and, in turn, signs local servers' certificates. This especially makes sense for integration tests. Getting certificates can be a burden because the servers will be up for minutes. But having an „ignore certificate“ option in the code could allow it to be activated in production, leading to a security catastrophe. A CA certificate is not much different from a regular server certificate; what matters is that it is trusted by local code. For example, in the requests library, this can be done by setting the REQUESTS_CA_BUNDLE variable to a directory containing this certificate. In the example of creating a certificate for integration tests, there is no need for a long-lived certificate: if your integration tests take more than a day, you have already failed. So, calculate yesterday and tomorrow as the validity interval: >>> import datetime
>>> one_day = datetime.timedelta(days=1)
>>> today = datetime.date.today()
>>> yesterday = today - one_day
>>> tomorrow = today - one_day Now you are ready to create a simple CA certificate. You need to generate a private key, create a public key, set up the „parameters“ of the CA, and then self-sign the certificate: a CA certificate is always self-signed. Finally, write out both the certificate file as well as the private key file. <div class=„geshifilter text geshifilter-text“ readability=„55“>from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import hashes, serialization
from cryptography import x509
from cryptography.x509.oid import NameOID private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
    backend=default_backend()
)
public_key = private_key.public_key()
builder = x509.CertificateBuilder()
builder = builder.subject_name(x509.Name([
    x509.NameAttribute(NameOID.COMMON_NAME, 'Simple Test CA'),
]))
builder = builder.issuer_name(x509.Name([
    x509.NameAttribute(NameOID.COMMON_NAME, 'Simple Test CA'),
]))
builder = builder.not_valid_before(yesterday)
builder = builder.not_valid_after(tomorrow)
builder = builder.serial_number(x509.random_serial_number())
builder = builder.public_key(public_key)
builder = builder.add_extension(
    x509.BasicConstraints(ca=True, path_length=None),
    critical=True)
certificate = builder.sign(
    private_key=private_key, algorithm=hashes.SHA256(),
    backend=default_backend()
)
private_bytes = private_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.TraditionalOpenSSL,
    encryption_algorithm=serialization.NoEncrption())
public_bytes = certificate.public_bytes(
    encoding=serialization.Encoding.PEM)
with

open(„ca.pem“, „wb“) as fout:
    fout.write(private_bytes + public_bytes)
with open(„ca.crt“, „wb“) as fout:
    fout.write(public_bytes) In general, a real CA will expect a certificate signing request (CSR) to sign a certificate. However, when you are your own CA, you can make your own rules! Just go ahead and sign what you want. Continuing with the integration test example, you can create the private keys and sign the corresponding public keys right then. Notice COMMON_NAME needs to be the „server name“ in the https URL. If you've configured name lookup, the needed server will respond on service.test.local. service_private_key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
    backend=default_backend()
)
service_public_key = service_private_key.public_key()
builder = x509.CertificateBuilder()
builder = builder.subject_name(x509.Name([
   x509.NameAttribute(NameOID.COMMON_NAME, 'service.test.local')
]))
builder = builder.not_valid_before(yesterday)
builder = builder.not_valid_after(tomorrow)
builder = builder.public_key(public_key)
certificate = builder.sign(
    private_key=private_key, algorithm=hashes.SHA256(),
    backend=default_backend()
)
private_bytes = service_private_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.TraditionalOpenSSL,
    encryption_algorithm=serialization.NoEncrption())
public_bytes = certificate.public_bytes(
    encoding=serialization.Encoding.PEM)
with open(„service.pem“, „wb“) as fout:
    fout.write(private_bytes + public_bytes) Now the service.pem file has a private key and a certificate that is „valid“: it has been signed by your local CA. The file is in a format that can be given to, say, Nginx, HAProxy, or most other HTTPS servers. By applying this logic to testing scripts, it's easy to create servers that look like authentic HTTPS servers, as long as the client is configured to trust the right CA. </html>

From:
https://schnipsl.qgelm.de/ - **Qgelm**

Permanent link:
https://schnipsl.qgelm.de/doku.php?id=wallabag:be-your-own-certificate-authority

Last update: **2021/12/06 15:24**