Activity V : Public Key Infrastructure

Created by : Krerk Piromsopa, Ph.D

Overview

In this activity, you will learn the fundamentals of Public Key Infrastructure. We will need the following tools:

A. OpenSSL. On Linux and Mac OS X, the OpenSSL is installed by default. For Windows, you may download it from https://wiki.openssl.org/index.php/Binaries .

B. Python with PyOpenSSL and pem to do our exercise. If you python does not come with PyOpenSSL and pem, you may install it with pip
$ pip install pyopenssl
$ pip install pem

C. You also need ca-certificates.crt from your OS (e.g. /etc/cacerts/ca-certificates.crt in Linux) or take it from the course web.

Exercise

Issuing the following command. **openssl s_client -connect twitter.com:443**

Once connected, you may try
GET / HTTP/1.0
[Enter twice]

(Note that the server may return HTTP 404. This is completely normal since we did not send a request for a valid resource.)

Repeat the same step again, now with

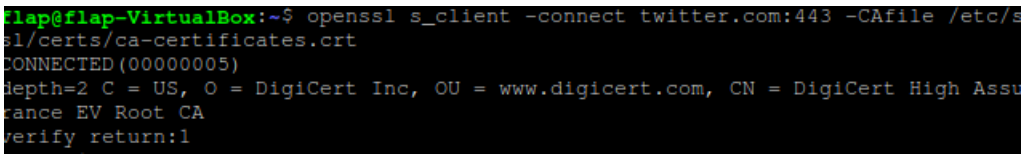
This command basically connects to port 443 (HTTPS) with the TLS/SSL. This is like a standard telnet command, but with openssl performing the encryption for you.

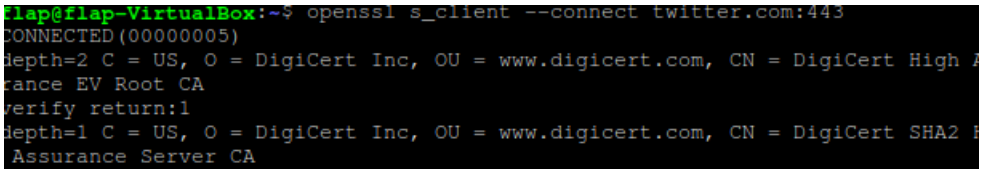
1. From the two given openssl commands, what is the difference?

   Note: If your operating system does not show any error in the first command, try **openssl s_client -connect twitter.com:443 -CApath /dev/null** . If the results are still the same, your system is not reliable. You may ignore this exercise. (Modern

versions of Mac OS X will always read CA from keychains. There is no intuitive way to turn it off.)

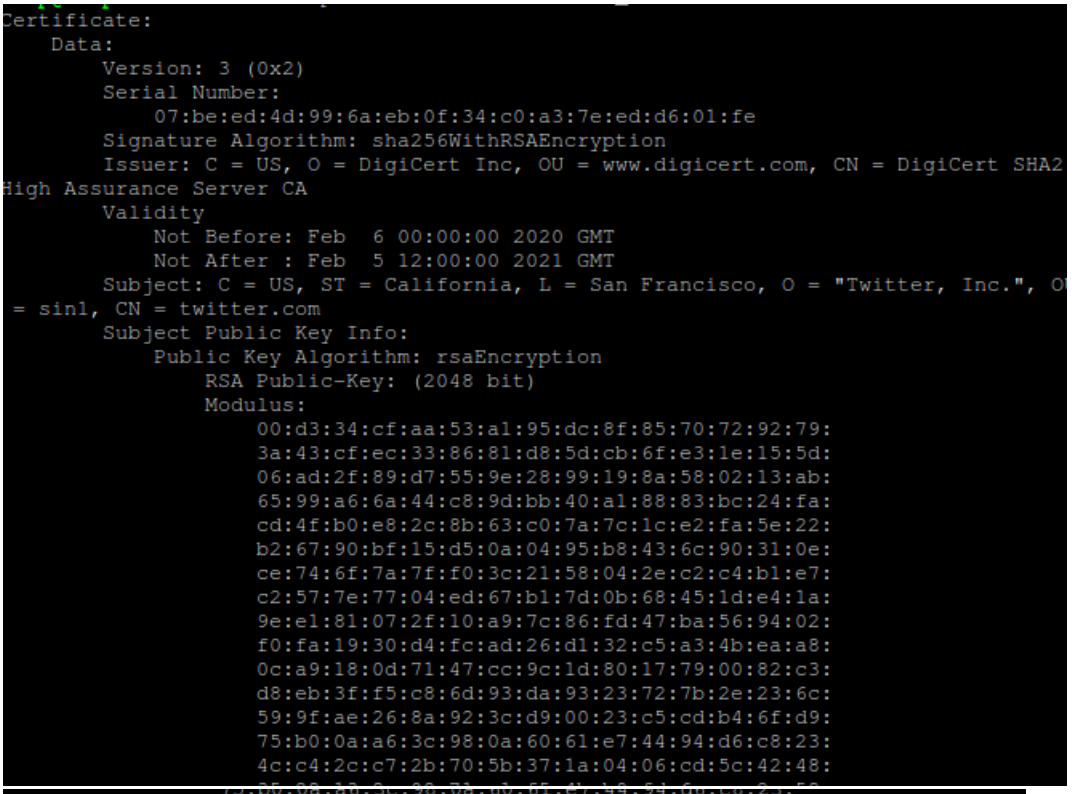
Same output

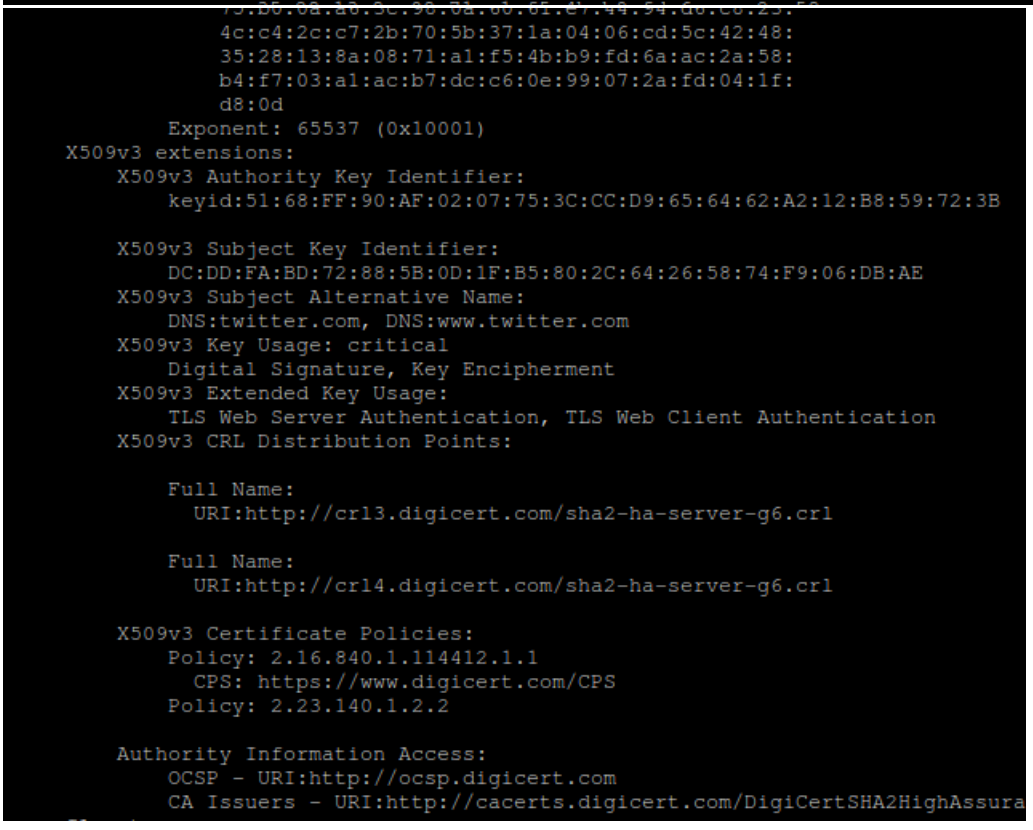




2. What does the error (**verify error**) in the first command mean? Please explain.
   I get no error

3. Copy the server certificate (beginning with -----BEGIN CERTIFICATE----- and ending with -----END CERTIFICATE-----) and store it as twitter_com.cert. Use the command **openssl x509 -in twitter_com.cert -text** to show a text representation of the certificate content. Briefly explain what is stored in an X.509 certificate (i.e. data in each field).





Version number: certificates version

Serial number: unique number identify cert

Signature algorithm: Hash algorithm

Issuer information

Company file for issue information
Date range that the certificate valid
Pub key details

4. From the information in exercise 3, is there an intermediate certificate? If yes, what purpose does it serve?

   Hint: Look for an issuer and download the intermediate certificate. You may use the command **openssl x509 -inform der -in intermediate.cert -text** to show the details of the intermediate certificate. (Note that the -inform der is for reading the DER file. The default file format for x509 is the PEM file.)

```
        Issuer: C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert High Assurance EV
Root CA
        Validity
            Not Before: Oct 22 12:00:00 2013 GMT
            Not After : Oct 22 12:00:00 2028 GMT
        Subject: C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert SHA2 High Assura
ce Server CA
```

   Yes the purpose are for providing an added level of security and compromise root PK

5. Is there an intermediate CA, i.e. is there more than one organization involved in the certification? Say why you think so.
   Yes, in the picture above Common Name is difference
6. What is the role of ca-certificates.crt?
   Store trusted certificate
7. Explore the ca-certificates.crt. How many certificates are in there? Give the command/method you have used to count.
   127

```
flap@flap-VirtualBox:~$ grep -o -i BEGIN /etc/ssl/certs/ca-certificates.crt | wc -l
```

8. Extract a root certificate from ca-certificates.crt. Use the openssl command to explore the details. Do you see any Issuer information? Please compare it to the details of twitter's certificate and the details of the intermediate certificate.
   There are issuer information but it's value is in gibberish compare to twitter

```
flap@flap-VirtualBox:~$ openssl x509 -in /etc/ssl/certs/ca-certificates.crt -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 6828503384748696800 (0x5ec3b7a6437fa4e0)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: CN = ACCVRAIZ1, OU = PKIACCV, O = ACCV, C = ES
```

9. If the intermediate certificate is not in a PEM format (text readable), use the command to convert a DER file (.crt .cer .der) to PEM file. **openssl x509 -inform der -in certificate.cer -out certificate.pem.**
   (You need the pem file for exercise 10.)

10. From the given python code,[1] implement the certificate validation.

```
from OpenSSL import crypto
import pem

def verify():
    with open('./target.cert', 'r') as cert_file:
        cert = cert_file.read()

    with open('./intermediate.cert', 'r') as int_cert_file:
        int_cert = int_cert_file.read()

    pems=pem.parse_file('./ca-certificates.cert');
    trusted_certs = []
    for mypem in pems:
        trusted_certs.append(str(mypem));

    trusted_certs.append(int_cert);

    verified = verify_chain_of_trust(cert, trusted_certs)

    if verified:
        print('Certificate verified')



def verify_chain_of_trust(cert_pem, trusted_cert_pems):

    certificate = crypto.load_certificate(crypto.FILETYPE_PEM, cert_pem)

    # Create and fill a X509Store with trusted certs
    store = crypto.X509Store()
    for trusted_cert_pem in trusted_cert_pems:
```

[1] Code taken from http://www.yothenberg.com/validate-x509-certificate-in-python/ . It has been modified for this exercise.

```
        trusted_cert = crypto.load_certificate(crypto.FILETYPE_PEM,
trusted_cert_pem)
        store.add_cert(trusted_cert)

    # Create a X590StoreContext with the cert and trusted certs
    # and verify the the chain of trust
    store_ctx = crypto.X509StoreContext(store, certificate)
    # Returns None if certificate can be validated
    result = store_ctx.verify_certificate()

    if result is None:
        return True
    else:
        return False
```

Use your program to verify the certificates of:

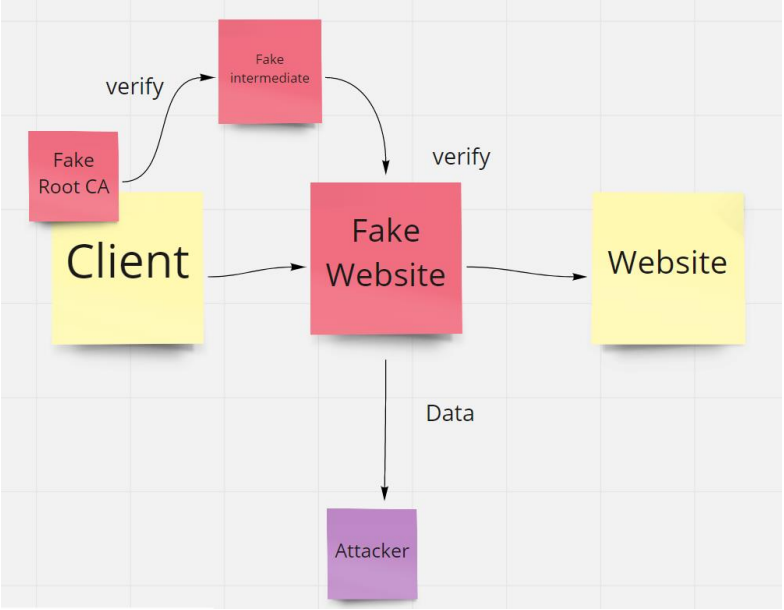
Twitter, google, www.chula.ac.th, classdeedee.cloud.cp.eng.chula.ac.th

They are all verified. Especially consider www.chula.ac.th which have 3 intermediate which are also all verified.

```
flap9@Avalon MINGW64 ~
$ python cert-verify.py google_com.cert
google_com
Certificate verified

flap9@Avalon MINGW64 ~
$ python cert-verify.py classdd_com.cert
lassdd_com
Certificate verified

flap9@Avalon MINGW64 ~
$ python cert-verify.py twitter_com.cert
witter_com
Certificate verified

flap9@Avalon MINGW64 ~
$ python cert-verify.py chula_int1_com.cert
hula_int1_com
Certificate verified

flap9@Avalon MINGW64 ~
$ python cert-verify.py chula_int2_com.cert
hula_int2_com
Certificate verified

flap9@Avalon MINGW64 ~
$ python cert-verify.py chula_int3_com.cert
hula_int3_com
Certificate verified
```

11. Nowaday, there are root certificates for class 1 and class 3. What uses would a class 1 signed certificate have that a class 3 doesn't, and vice versa?
    Class3 is specialization of class1 which contain only high-security certificates which check organization and geolocation of server. On the other hand class1 only check domain name.
12. Assuming that a Root CA in your root store is hacked and under the control of an attacker, and this is not noticed by anyone for months.
    a. What further attacks can the attacker stage? Draw a possible attack setup.
    Attacker can do man-in-the-middle sproofing by let client connect to fake

website that verify by fake root CA that attacker inject to root store.



b. In the attack you have described above, can we rely on CRLs or OCSP for protection? Please explain

CRLs can't help in this case since fake-root is newly created and not in CRLs.
OCSP can't help either since it's rely on root which got faked.