DOTA: - Laboratory #4

1 Lab 4, part 1: Creating a certificate authority

There is a tool installed on your machine that allows you to perform all kinds of cryptographic tasks. It is called `openssl` and you can find out about its large variety of functions by typing "`man openssl`". Before beginning this lab, it may be useful to create a directory in which to store things:

```
cd
mkdir  LAB4; cd LAB4
```

In this directory, create[1] a configuration file for `openssl`. Lets call it `localhost.cnf` and it is something like this:

```
[req]
default_bits = 2048
default_md = sha256
prompt = no
distinguished_name = req_distinguished_name
x509_extensions = v3_ca
req_extensions = v3_req
[req_distinguished_name]
C = SG
ST = Clementi
L = NUS
O = Widgets For All
CN = localhost
[v3_ca]
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer:always
basicConstraints = critical, CA:true, pathlen:0
keyUsage = critical, digitalSignature, cRLSign, keyCertSign
[v3_req]
subjectKeyIdentifier = hash
basicConstraints = critical, CA:false
nsCertType = server
keyUsage = digitalSignature, nonRepudiation, keyEncipherment
extendedKeyUsage = serverAuth
subjectAltName = @alt_names
[alt_names]
DNS.1 = localhost
IP.1 = 127.0.0.1
IP.2 = ::1
```

[1] You could use cut and paste, and the editor `pico` (i.e. `pico localhost.cnf`).

You can change the values in the `[req_distinguished_name]` section to reflect your own organization and location (At the moment it is "Widgets for all", based in SG/Singapore)

Use `openssl` to create a root certificate[2]:

```
openssl genrsa -out testCA.key 2048
openssl req -x509 -new -nodes -key testCA.key -sha256 -days 365 -out testCA.crt \
   -config localhost.cnf -extensions v3_ca -subj "/CN=DOTA Test CA"
```

You can now create the certificate for the server:

```
openssl genrsa -out localhost.key 2048
openssl req -new -key localhost.key -out localhost.csr -config localhost.cnf -extensions v3_req
openssl x509 -req -in localhost.csr -CA testCA.crt -CAkey testCA.key -CAcreateserial \
   -out localhost.crt -days 365 -sha256 -extfile localhost.cnf -extensions v3_req
```

This creates a certificate `localhost.crt`.

Testing your CA

You can now test your CA! First, add your CA (from `testCA.crt`) to the list of trusted CAs in your browser. It will appear as "DOTA test CA". You can see how to do this at:

`https://wiki.wmtransfer.com/projects/webmoney/wiki/Installing_root_certificate_in_Mozilla_Firefox`
or
`https://tinyurl.com/ee8dbhut`

Openssl itself comes with a very primitive test webserver that we will use as a test SSL webserver. Create a small `index.html` file for testing purposes that should be displayed when people connect via SSL. Run the test webserver the first time only to see information exchanged between the webserver and the client. You can start the SSL webserver on port 4443 by typing:

```
openssl s_server -accept 4443 -www -state -cert localhost.crt -key localhost.key
```

Use `firefox` to connect to `https://localhost:4443`. Look at the information given on the server side, as well as on the client side. Now run the same command but with "`-WWW`" in place of "`-www`".

Connect to `https://localhost:4443/index.html`. Look to see the details of your encrypted connection.

Writeup

Once you have finished the lab, answer the following three questions, using the normal grading website. Here is the site: `https://hugh.comp.nus.edu.sg/DOTA/lab4/gradeslab4-1.php`

1. What is the purpose of the extra information in the (CA) certificate you created?
2. Explain the files you created. What is each file used for?
3. Why is the webserver using both the certificate and private key? (What does it use each for?)

[2]By the way - on my machine, there were two installations of openssl, and it would not run properly. If I specified the correct one for each command (/usr/bin/openssl) - then it worked fine.

2 Lab 4, part 2: SEED Buffer-Overflow Vulnerability Lab

Have a look at the SEED Buffer-Overflow Vulnerability lab, found at

`https://seedsecuritylabs.org/Labs_20.04/Software/Buffer_Overflow_Setuid/`

There is a lot in this laboratory, but I would like you to just try tasks 1, 2 and 3, a simple buffer overflow attack. Once you have finished the lab, you can upload your exploit.c code to the Luminus submission folder for Lab4-2. Please name your file t09XXXXXX.c (your login id followed by .c).

Access the DOTA grading website, and enter in a brief description of the steps someone could follow to recreate your attack, using your code.

`https://hugh.comp.nus.edu.sg/DOTA/lab4/gradeslab4-2.php`

3 Lab 4, part 3: SEED Return-to-libc Attack lab

Have a look at this paper

`http://www.infosecwriters.com/Papers/C0ntex_return-to-libc.pdf`

It provides some useful background to the SEED Return-to-libc Attack lab, which you are going to do, found at

`https://seedsecuritylabs.org/Labs_20.04/Software/Return_to_Libc/`

There is a lot in this laboratory, but I would like you to just try Section 3, tasks 1,2 and 3, a simple return to libc attack. Once you have finished the lab, you can upload your exploit.c code to the Luminus submission folder for Lab4-3. Please name your file t09XXXX.c (your login id followed by .c).

Access the DOTA grading website, and describe how you decide the values for X, Y and Z. Either show us your reasoning, or if you use trial-and-error approach, show your trials. After your attack is successful, change the file name of retlib to a different name, making sure that the length of the file names are different. For example, you can change it to newretlib. Repeat the attack (without changing the content of badfile). Is your attack successful or not? If it does not succeed, explain why.

`https://hugh.comp.nus.edu.sg/DOTA/lab4/gradeslab4-3.php`