ING4 SI01

Security of IS

Cryptography

Lab 2

Author(s):
Gabriel PADIS
Anastasia DUCHESNE

Teacher:
Mr. Hamon

Paris - Semester 7 - March 5, 2019

Contents

1 Self-signed certificate

1.1 Creation

We want to create our own certificate for our web server so we can connect to it in HTTPS instead of HTTP. The command we used to create a 4096 bit certificate valid for 90 days is the following:

```
openssl req -newkey rsa:4096 -nodes
-keyout <private key name>.key -x509 -days 365 -out <certificate name>.crt
```

From the manual of openssl we now that:

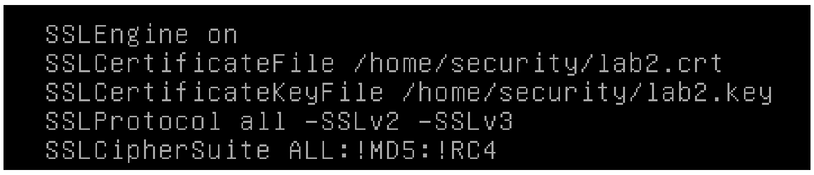
- req: certificate request and certificate generating utility
- -newkey: creates a new certificate and a new private key
  - rsa: number of bits
- -keyout: give the name of the filename
- -x509: outputs a self signed certificate instead of a certificate request
- -days: number of days to certify for
- -out: specifies filename

1.2 Issues

The security issue with the self-signed certificate is that no authority can confirm that it is a valid certificate. You are the one that validates your own site. No one can know if you're a valid secured website or if you only pretend so by having your own certificate. Since you're not a recognized authority you're not trustable.

2 Enabling HTTPS in Apache

We entered the file **/etc/apache2/sites-available/default-ssl.conf**.
In it we added the following lines:



```
SSLEngine on
SSLCertificateFile /home/security/lab2.crt
SSLCertificateKeyFile /home/security/lab2.key
SSLProtocol all -SSLv2 -SSLv3
SSLCipherSuite ALL:!MD5:!RC4
```

Figure 1: SSL configuration

Each line does, in order:

- Activates SSL
- Precises the path of the certificate
- Precises the path oh the key
- Enables all protocols for connections except SSLv2 and SSLv3
- Enables all protocols for Hashing and Encoding except MD5 and RC4

To activate our Virtual Host we did:

```
ln-s /etc/apache2/sites-available/default-ssl.conf
    /etc/apache2/sites-enabled/default-ssl.conf
```

This command creates a hard link or a symbolic link to an existing file:

- -s: symbolic links instead of hard links

To enable the SSL and have a functioning apache server we did:

```
a2enmod ssl
service apache2 restart
```

Finally we updated our firewall so that the HTTPS calls could pass, it allows the server to function correctly:

```
iptables -I INPUT -p tcp -d 192.168.10.100 --dport 443 -j ACCEPT
```

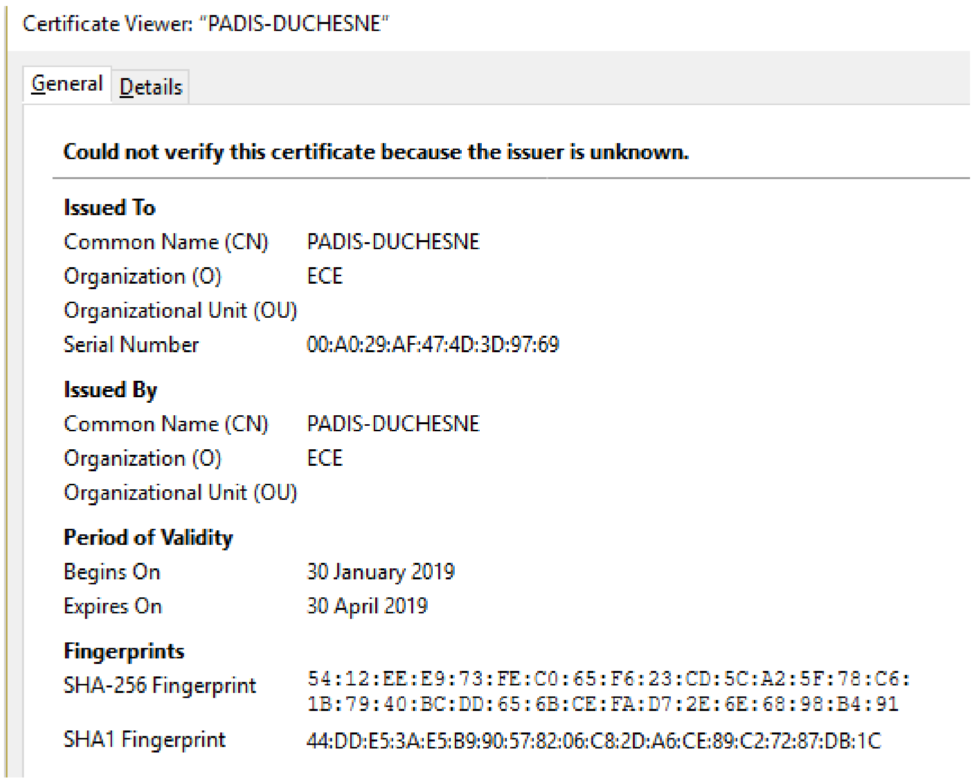
3 Connection to the web server in HTTPS

To connect in HTTPS to our website we typed in our research bar:

```
https://192.168.10.100
```

Our browser warned us that the certificate for HTTPS used by that website was not validated by a known authority. We proceeded anyway since we knew it was our website and that we self-signed it.
We then had access to the website like normal but in a secure mode.

We could verify the certificate by showing it and verifying that it was our own.



Certificate Viewer: "PADIS-DUCHESNE"

General | Details

Could not verify this certificate because the issuer is unknown.

Issued To

Common Name (CN)	PADIS-DUCHESNE
Organization (O)	ECE
Organizational Unit (OU)	
Serial Number	00:A0:29:AF:47:4D:3D:97:69

Issued By

Common Name (CN)	PADIS-DUCHESNE
Organization (O)	ECE
Organizational Unit (OU)	

Period of Validity

Begins On	30 January 2019
Expires On	30 April 2019

Fingerprints

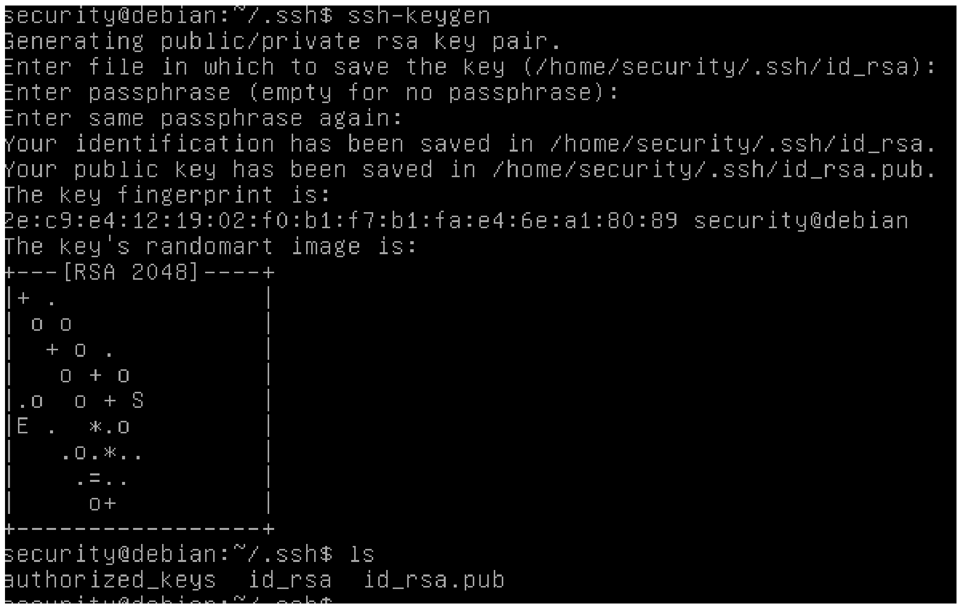
SHA-256 Fingerprint	54:12:EE:E9:73:FE:C0:65:F6:23:CD:5C:A2:5F:78:C6:1B:79:40:BC:DD:65:6B:CE:FA:D7:2E:6E:68:98:B4:91
SHA1 Fingerprint	44:DD:E5:3A:E5:B9:90:57:82:06:C8:2D:A6:CE:89:C2:72:87:DB:1C

Figure 2: Our website's certificate

4 SSH connection

4.1 Creation of a SSH key pair

To create a SSH key pair, we simple have to use ssh-keygen command line. This function generates two files: **id_rsa** which corresponds to our private key, and **id_rsa.pub** which is the public one.



```
ecurity@debian:~/.ssh$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/security/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/security/.ssh/id_rsa.
Your public key has been saved in /home/security/.ssh/id_rsa.pub.
The key fingerprint is:
2e:c9:e4:12:19:02:f0:b1:f7:b1:fa:e4:6e:a1:80:89 security@debian
The key's randomart image is:
+---[RSA 2048]----+
|+ .              |
| o o             |
|  + o .          |
|   o + o         |
|.o  o + S        |
|E .  *.o         |
|   .o.*..        |
|    .=..         |
|     o+          |
+-----------------+
ecurity@debian:~/.ssh$ ls
authorized_keys  id_rsa  id_rsa.pub
```

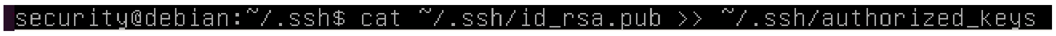
Figure 3: ssh-keygen

The key fingerprint is a short sequence used to identify the public key.

4.2 SSHd configuration

Our SSH connection is going to be encrypted with our public key. Then, we're going to decrypt it, in order to connect to our server, with our private key.
There is several steps to SSHd configuration.

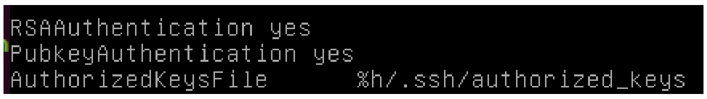
First, you have to add your public key to the **authorizedkeys** file. In this file are stores all the keys used to encrypt the SSH connection.



```
security@debian:~/.ssh$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Figure 4: Adding the public key

Once it is done, you have to configure the SSHd file **/etc/ssh/sshdconfig**.



```
RSAAuthentication yes
PubkeyAuthentication yes
AuthorizedKeysFile      %h/.ssh/authorized_keys
```

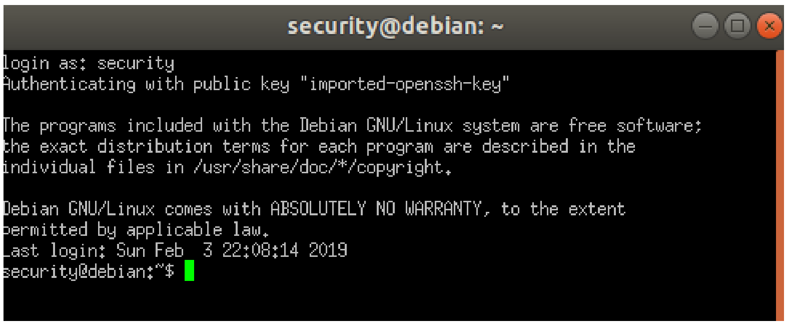
Figure 5: /etc/ssh/sshdconfig configuration

Two lines are important:

- **PubkeyAuthentication yes**: enables public key authentication
- **AuthorizedKeysFiles <path to file>**: tells which are the encryption keys file(s).

Now that our file is encrypted with our public key on our server, we're going to decrypt it with the private key from our computer.

4.3 SSH connection



```
login as: security
Authenticating with public key "imported-openssh-key"

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Feb  3 22:08:14 2019
security@debian:~$
```

Figure 6: /etc/ssh/sshdconfig configuration

In order to connect to our server from our computer, we use Putty. First, we have to convert our idrsa key to a .ppk format, which is the one supported by putty. Once it's done, we only have to inquire it in, with the server IP address.

Then, it opens a dialog box, that asks our login: security. The connection to our server is successful, it authenticated itself with the public key !

It is way more secure because to an asymmetric algorithm, even though it works much slowly because of the mathematically intensive tasks.
Each entity has a different key: our server has a public key to encrypt it files,

and our computer has the private one. These two keys are mathematically related which makes the decryption possible, but are too different which makes it impossible to encrypt and decrypt a file with the same one. Having this resolves the problem of distribution encountered with the symmetric key.