Introduction to Computer Security

Lab 3 - Exploring Identity & Authentication

Report
of
Andreas Wilhelm

February 11, 2019

You can find the code and everything else also on GitHub under the link *https://github.com/awilsee/CSec*. Maybe more convenient for you.

1 Exploring Certificate Authority Public Key Infrastructure (CA-PKI) with OpenSSL

1.1 A

Beside your descriptions I also have to change the following two options to get the web server in chapter 2 running. Additionally I have generated the ca-key with 2048 bits.

```
default_bits        = 1024
CipherString = DEFAULT@SECLEVEL=1
```

1.2 B

What do you observe? What fields are there?
It shows the data of the certificate with all the inputs from before. Additionally you can see that *sha256WithRSAEncryption* is used for encryption. Further more the RSA public key is included.

1.3 C

Open the key file your just created with a text editor and note your observations.
You can see the 3DES encrypted private key, modulus, the private/public exponent, both primes and exponents. All encrypted with your previously choosen keyphrase.

What do you observe in this certificate? Try to connect your observations to the earlier outputs. For example, where did the the serial number of the certificate come from?
The certificate includes the modulus and exponent of the certificate which is needed for the RSA decryption. Additionally it contains also the *X509v3 Authority Key Identifier* which is the identification from my created CA and its corresponding key. So it's now signed from the CA.

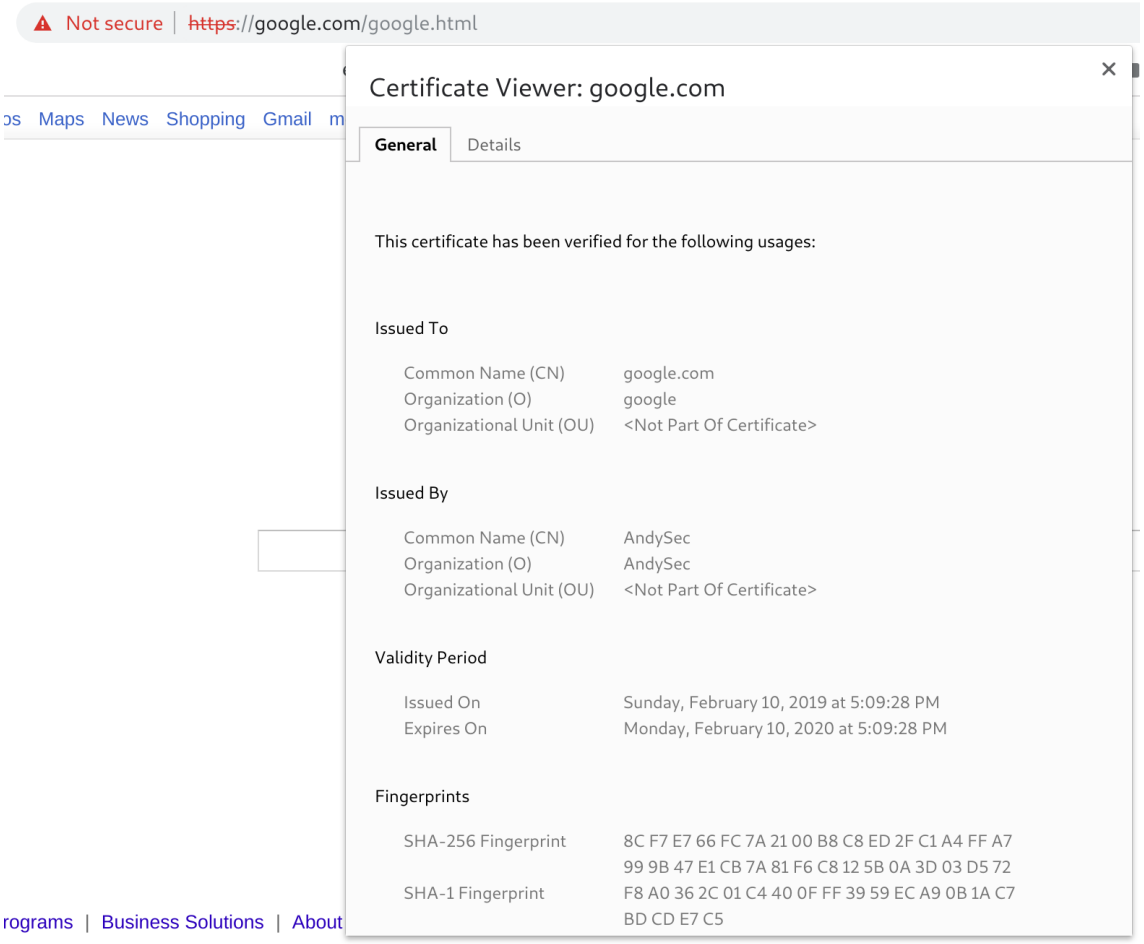
What capabilities have you just given an entity by signing their public key? What prevents anyone from impersonating Google or other wellknown service in this way?
I give the entity now the possibility to use their certificate which is signed and therefore trustful which means it can now basically used by him. The user wouldn't recognize that this CA wasn't real, because he doesn't check if all the certificates are from a trustful CAs.
The reason for that is that all not anybody can create a CA and sign certificates. Only certain CA can sign them and the browser only accept this ones, otherwise they block the content or at least warn the user.

2 Using CA PKI to Secure and Authenticate Web Sites

Can you verify this?
Yes, I can confirm that the browser doesn't accept the certificate. Though in chrome you can go to advanced and manually trust this certificate. In firefox you have to import the CA-key at first as described.

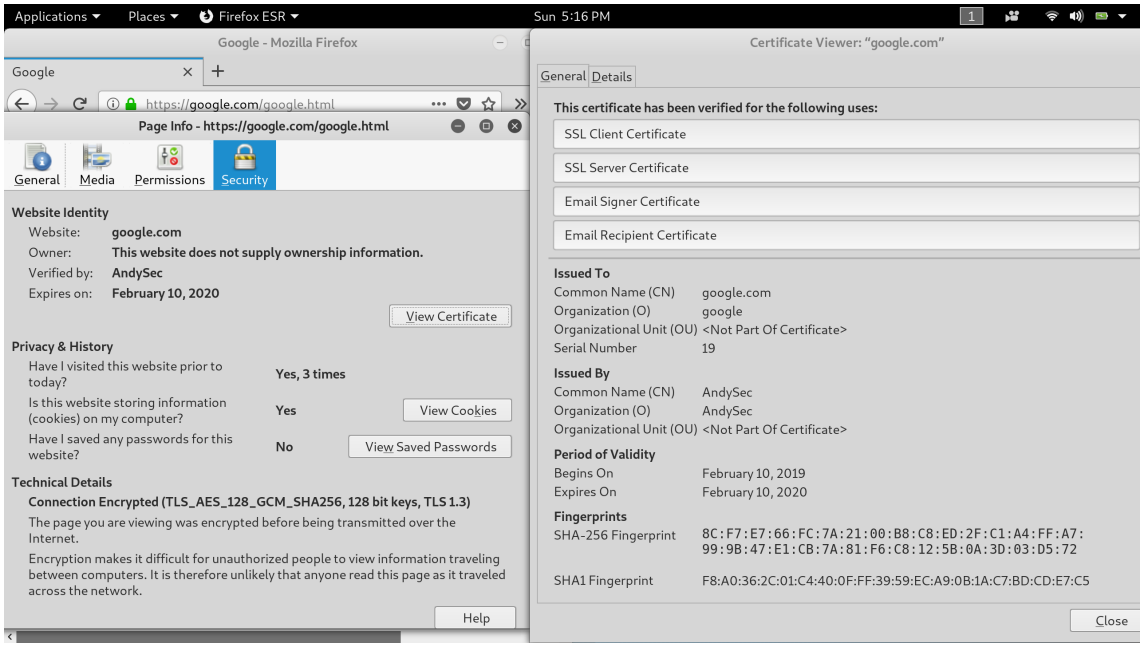
Is there any way for the user or the web browser to know you're not at the real google.com
In Chrome you get still warned that this site is insecure, as you can see in figure 1. If you click on the details, you can see for whom the certificate is, that it was issued by the CA-organisation, the validity and the fingerprints.

Figure 1: Screenshot of Chrome



After you imported the CA to the trustful CAs. You don't get any warning and a green locked key lock sign, as you can see in figure 2. So for the first view anybody can't see any difference compared to the original site except that the faked one is out of date.. Only if you click to see the details you see the same detailed informations about it as in chrome respectively decribed before.

Figure 2: Screenshot of Firefox



3 Password Files

What do you observe?
Well, for doing this, you obviously need superuser rights. When you open it as superuser you also only see the hash.

What is the hash of your password? What is the date of your last password change?
I won't publish my password, neiter the hash, sry! The number in the shadow file is 17908. This means 17908 days after January 1st 1970. So the actual date is January 12th 2019. That's the date where I set up the system.

What algorithm was used to protect your password? What is the value of the salt used to protect your password?
The SHA-512 algorithm was used to protect the password. The salt is *jn2qldDW*. The characters in *salt* are drawn from the set [a-zA-Z0-9./]. The encrypted password string contains 86 characters.

4 Password Cracking

```
1 import base64
2 import hashlib
3
4 pw_file = "./passwordfile.txt"
5 dict_file = "/usr/share/dict/words"
6 dict_file2 = "./passwords-dict.txt"
7
8
9 def read_userpw_file():
```

```
10      entries = []
11      with open(pw_file, 'r') as f:
12          for line_terminated in f:
13              username, pwhash = line_terminated.split(':')
14              entry = [username, pwhash.rstrip().split('}')[1]]
15              entries.append(entry)
16      return entries
17
18
19  def calc_hash_try_crack(file_path, userpw_entries):
20      with open(file_path, 'r', encoding='ISO-8859-1') as f:
21          for line_terminated in f:
22              line = line_terminated.rstrip('\n')
23              b64_hash = base64.b64encode(hashlib.sha1(line.encode('
        utf-8')).digest())
24              for userpw in userpw_entries:
25                  if b64_hash.decode() == userpw[1]:
26                      print("User {} has used PW: {}".format(userpw
        [0], line))
27                      break
28
29
30  if __name__ == '__main__':
31      userpw_entries = read_userpw_file()
32      calc_hash_try_crack(dict_file, userpw_entries)
33      calc_hash_try_crack(dict_file2, userpw_entries)
```

Listing 1: Code of task 4

4.1 A

By just looking at the hashes, what can you learn about an individual user's passwords? Is there anything you can learn about the passwords of all the users from this password file?
The passwords has all the same encryption. Furthermore all user are using a different password. They are encrypted using SHA-1 and bas64-encoded.

4.2 B

If your program completes (i.e. hashes all the words in the dictionary) without finding a match, what can you assume about the passwords? Which passwords were you able to crack?
If the program would complete without finding a match, the passwords would be well choosen, at least they contain no usual words which are in such simple dictionaries. However it could also be words in different languages, which are not in this special dictionary. The user *zachary* used the password *awesome*.

4.3 C

Which additional passwords did you crack?
User *ellie* has used PW: *2cute4u*
User *jamie* has used PW: *password1*

User *newton* has used PW: *zaq12wsx*

Were you able to crack all the passwords? Why or why not? If not, what characteristics do you think the un-cracked password has?
I wasn't able to crack the passwords of *copernicus* and *clara*. The passwords could be some real random ones, at least not any with simple combinations which are listed in this dictionary.

For those passwords you were unable to crack, what would be required to crack them and how do you think it would take?
You could for example brute-force them including the birthday attack. You would need 2^{80} SHA-1 evaluations, which will take long. With more efficient methods you would still need about 2^{60} evaluations which is in result 6,500 years of single-CPU computations and 110 years of single-GPU computations. But nowadays with renting stacks of GPUs, the needed time gets really small.

Using each of the cracked passwords, go to the user's respective web page. What is the secret word for each user?
For user *zachary* the secret word is 'This is my world now.'
For user *ellie* the secret word is 'Hoochie Mama!'
For user *jamie* the secret word is 'I've made a huge mistake.'
For user *newton* the secret word is 'S'up dawg.'