NT Assignment 6

Security  
By: Brent Schoenmakers & Cihan Kurt

Introduction

This assignment deals with the Network Security. The students who have already followed Digital Security Specialization Route can choose whether to complete tasks 1-3 dealing with hashing, cryptography and authentication or whether to complete task 4. The students that did not follow the Digital Security Specialization Route should complete tasks 1-3. Task 5 deals with firewalls using iptables and is for everybody optional.

Task 1: Hashing

We want to investigate how hashing can help us to detect changes in our data.

Download the plain text file ‘alice.txt’ from:

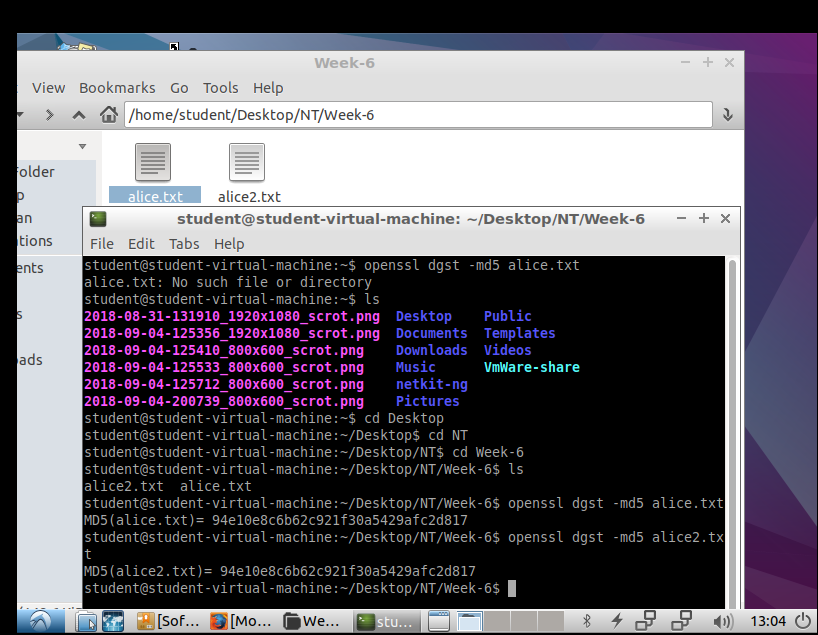
**http://www-net.cs.umass.edu/wireshark-labs/alice.txt**

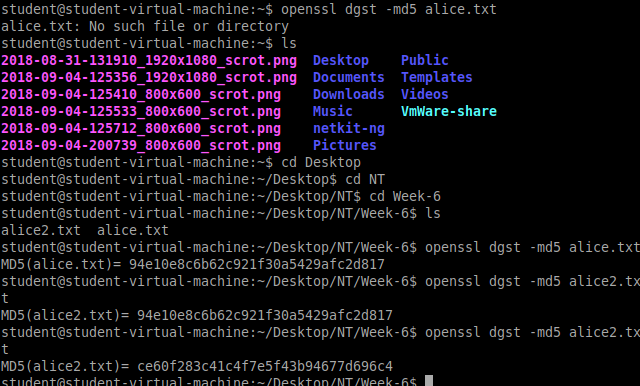
Make a copy of this file called ‘alice2.txt’.

Create a MD5 hash (digest) of both files:

**openssl dgst -md5 alice.txt**

**openssl dgst -md5 alice2.txt**

**Done:**Then start a text editor and **make a small change in the alice2.txt file**, e.g. change one character or one word. Now make a new MD5 hash of alice2.txt:

**openssl dgst -md5 alice2.txt   
Done:**

What is the hash of alice.txt?

The hash value of the unchanged file alice.txt is: 94e10e8c6b62c921f30a5429afc2d817  
The hash value of the changed file alice2.txt is: ce60f283c41c4f7e5f43b94677d696c4  
The changes that I made:   
The first word of the .txt file is Alice’s. I changed it to BAlice’s.

What is the influence of copying and/or renaming of a file on its hash?   
Nothing. The hash value will still be the same. The hash value is based on the contents of a file. Not the name or the location.

What is the influence of modifying a file?   
Changing the content of a file will change the hash value completely. Not just a bit of the hash, but everything.

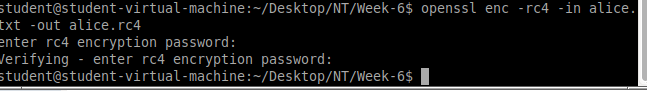
Task 2: Cryptography

We want to investigate cryptography to protect our data.

Encrypt alice.txt using the RC4 stream cipher algorithm:

**openssl enc -rc4 -in alice.txt -out alice.rc4**

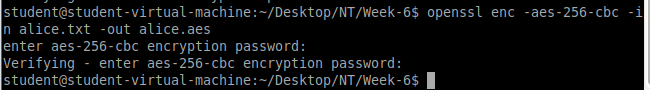
**Done:**



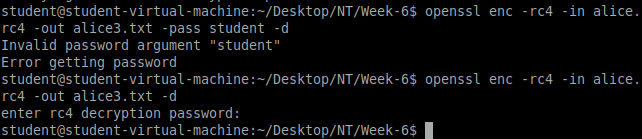
**The password is: student**

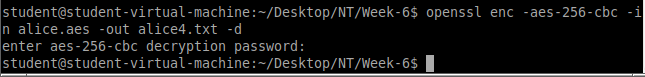
Encrypt alice.txt using the AES block cipher algorithm:

**openssl enc -aes-256-cbc -in alice.txt -out alice.aes   
Done:**

 **The password is: student**

Decrypt alice.rc4 and write the output to alice3.txt  
**Done:**



Decrypt alice.aes and write the output to alice4.txt  
**Done:**

Answer the following questions:

1. Are alice.rc4 and alice.aes text files?   
   No, they are encrypted files and can not be opened with a text editor.
2. How did you decrypt the files (specify the commands)?   
   After using the following website: <http://manpages.ubuntu.com/manpages/xenial/man1/enc.1ssl.html>

We found out that we had to change some commands. The input should be the encrypted file and the output should be alice3/alice4.txt. However, the website says that we had to use the -pass flag as well. This did not work as shown on the screenshot for rc4 decrypting.   
So after thinking for a while, we found out that encrypting didn’t need a -pass flag immediately. It was asked after the command. So we tried the same. It asked us for the password after the command for decrypting.   
  
This was the command:  
  
*openssl enc -rc4 -in alice.rc4 -out alice3.txt -d*  
We added the -d because it is the flag for decrypting, while -e is the flag for encrypting. The password will be asked after executing this command.

3. Do the encrypted files alice.rc4 and alice.aes significantly differ in size?

No. After looking at the properties of both encrypted files we found out that they are exactly the same size.

4. Do the files alice4.txt, alice3.txt and alice.txt have exactly the same content? How did you check that?

Yes. We opened all three the files and scrolled quickly through them. They did not look any different. Since we did not check every word, we can’t really tell if they are EXACTLY the same. That’s why we checked the amount of bytes of each file. They were exactly the same. So the answer is Yes.

Write a simple encryption program (in C/C++ or C#) that:

* Can receive a password from the user
* Can read a user specified file (like alice.txt)
* Implements a simple stream cipher encryption : XOR the (repeated) password with the data in the input file
* Writes output to a user specified file

Write a decryption program (separate or combined in one program with encryption) that:

* Can receive a password from the user
* Can read a user specified file
* Decodes the data in the input file using the provided password
* Writes output to a user specified file

Task 3: Authentication

We will investigate a little bit of the concept of ‘Certificates’ for authentication and learn how to use them to implement HTTPS, a secure version of HTTP.

We will install and start an Apache Server on our Linux host by issuing the following command:

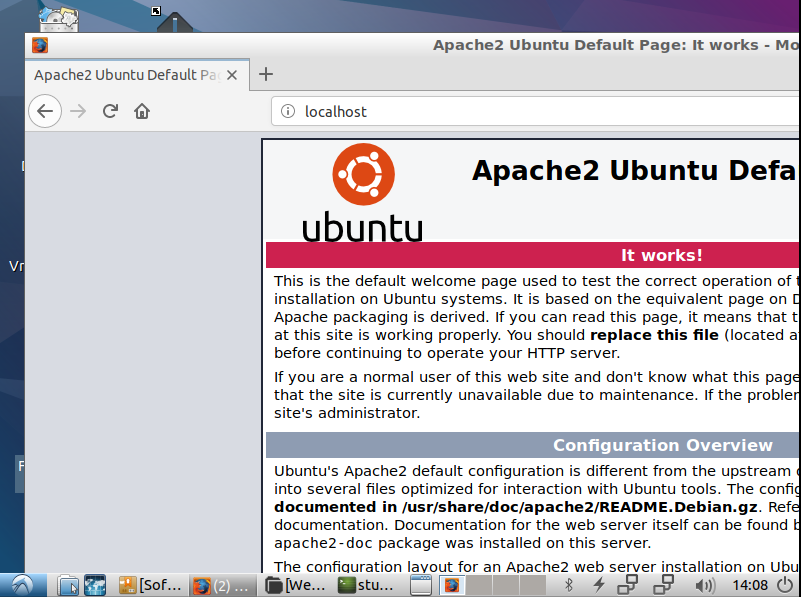
sudo apt-get update

sudo apt-get install apache2

sudo /etc/init.d/apache2 start.

You can check your successful installation by using “localhost” in the URL of your favourite browser.

**Done:**



Let’s now try do the same, but instead of using <http://localhost>, use <https://localhost>.

What difference do you see in your browser?  
The difference is that <http://localhost> is actually working while <https://localhost> is not working. It does not connect.

To be able to browse through https and to get encrypted communication we’re going to create ssl certificates.

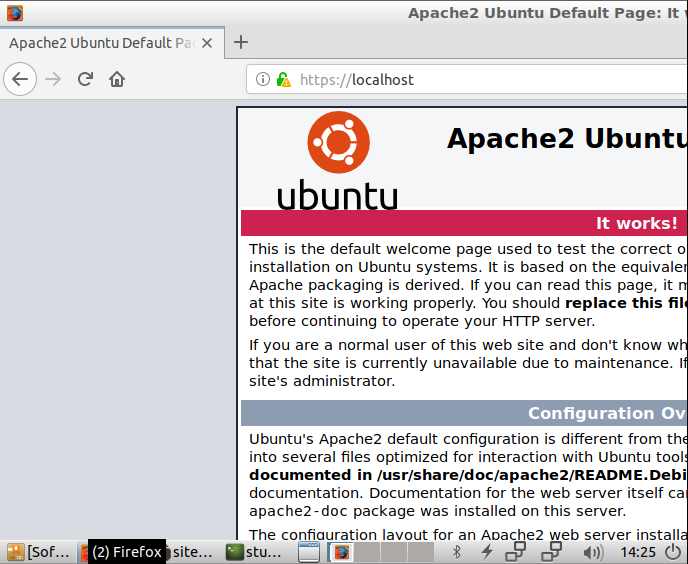
To do this, you can follow the tutorial at this page:

<https://www.digitalocean.com/community/tutorials/how-to-create-a-ssl-certificate-on-apache-for-ubuntu-14-04>.

*Tip: For your testing on localhost use* ***localhost*** *as your domain name.*

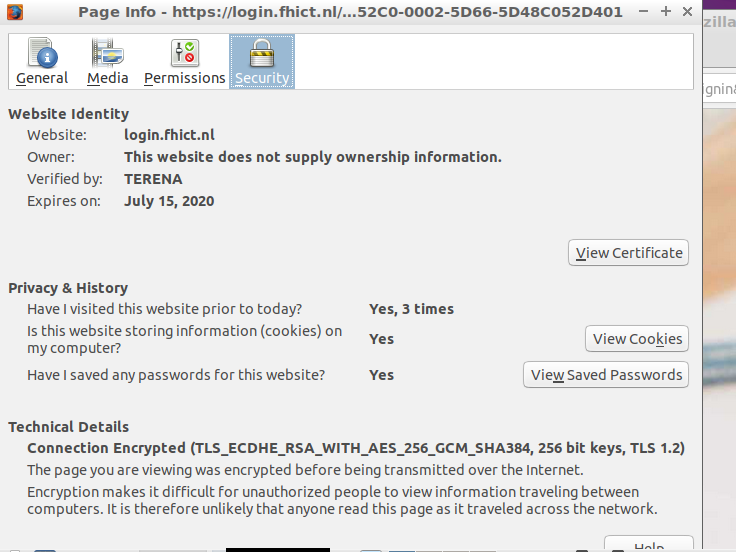
Now you should be able to use encrypted communication, although you’ll still have problems with your certificate because it’s self assigned and not trusted by any authority.

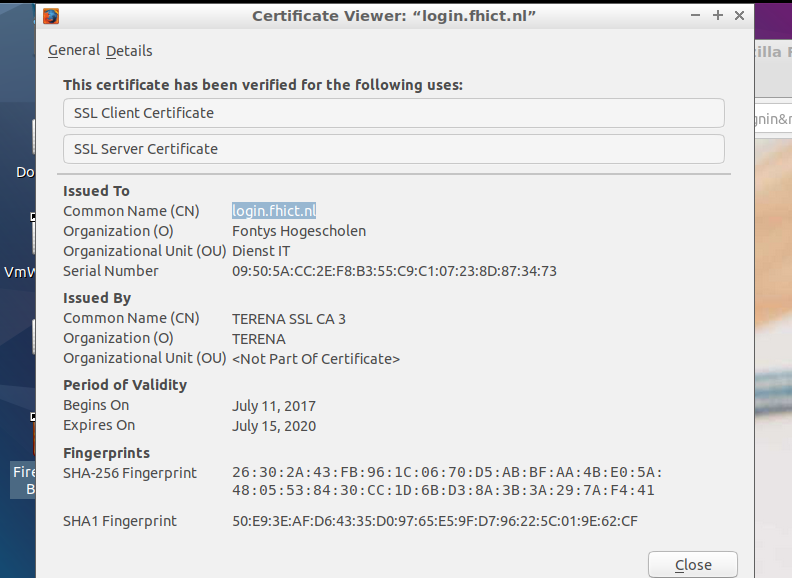
Provide a screenshot of your <https://localhost> after you’ve created and configured the certificates. You should be able to see the certificate by clicking on the lock icon.



I had to add an exception for this ssl because it was self-signed.

Go to the portal.fhict.nl site and provide a screenshot of the certificate this side is using. Who has signed this certificate? And who is Root CA of portal.fhict.nl?





We don’t know it exactly. But the signer should be Fontys Hogescholen Dienst IT (O and OU) while Terena SSL CA 3 should be the Root CA.

Evaluation Table for Assignment 6

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Insufficient (O)** | **Sufficient (V)** | **Good (G)** |
| Hashing | Task 1 (hashing) not done correctly. | Some answers are incomplete but the core of the task 1 (hashing) is done | Task 1 (hashing) is complete and can be well explained |
| Cryptography (use of openssl) | Task 2 (use of openssl) not done correctly | Some answers to T  task 2 (use of openssl) are incomplete but the core of the task 2 (openssl) is done | Task 2 (openssl) is complete and can be well explained |
| Cryptography (C, C++ or C# program) | Required program doesn’t work according to requirements | Core of the program but implementation of some of the requirements is incomplete or not completely correct. | Functionality of the program is fully according to the requirements |
| Task 3 (Authentication) | Task 3 (Authentication) is not done correctly without giving a good reason why not | Task is executed but questions are not answered exhaustively. | Task is executed and questions are answered exhaustively. |
| Task 4  (Presentation)  Optional to task 1-3 only for Cyber Security students | Task 4 incomplete and/or the topic not agreed upon by the teacher | Presentation is satisfactory. | Presentation is excellent, a deep research on the topic is done |
| Task 5 (optional) |  |  | Your chance of getting “G” for the assignment is much higher |
| PO | The assignment is not documented. | The assignment is documented. | The assignment documentation is neat and professionally done. |