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:

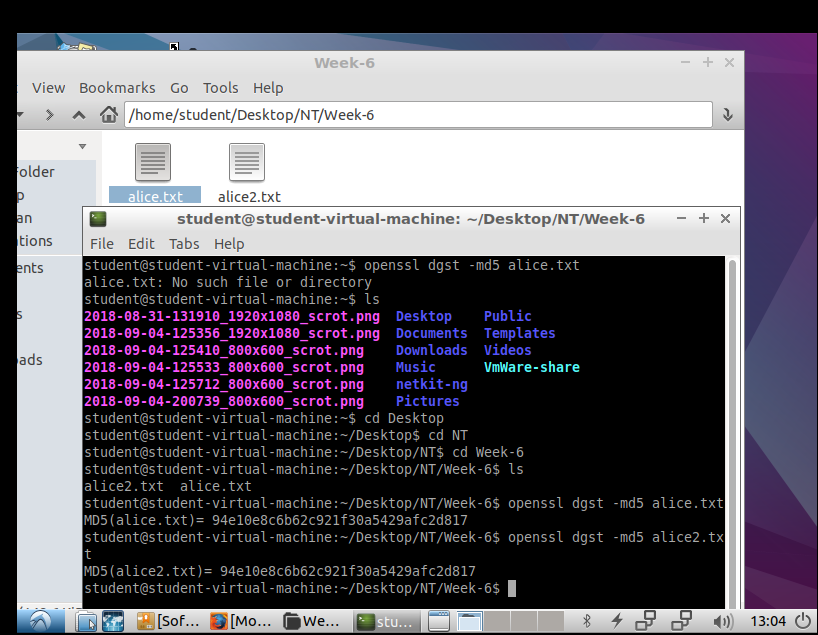
**http://www-net.cs.umass.edu/wireshark-labs/alice.txt**

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**

What is the hash of alice.txt?

What is the influence of copying and/or renaming of a file on its hash?

What is the influence of modifying a file?

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**

Encrypt alice.txt using the AES block cipher algorithm:

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

Decrypt alice.rc4 and write the output to alice3.txt

Decrypt alice.aes and write the output to alice4.txt

Answer the following questions:

1. Are alice.rc4 and alice.aes text files?
2. How did you decrypt the files (specify the commands)?

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

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

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.

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

What difference do you see in your browser?

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.

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?

Task 4 (alternative for people that followed Digital Security Specialization)

Create and present a +-10 minutes presentation/demonstration based on one of the chosen security topics. Don’t repeat exactly what you’ve done during your specialization route but be creative. Consult your choice of the topic with the teacher.

A few examples for the topics:

1. Security in IOT
2. GPG/PGP
3. IpSEC
4. SSL
5. Attacks , …

Task 5 (Optional): Configure Firewall on Linux/Netkit

For this task you can use the provided nat-dhcp lab. We’re going to use *iptables* command which can be used both for NAT configuration and for firewall implementation.

A short explanation of *iptables* before we start the exercise:

Iptables has three types of tables :FILTER, NAT and MANGLE. For this assignment we’re going to use FILTER table (default).

An *iptables* table consists of chains. A chain consists of a set of rules.

The FILTER table of *iptables* consists of three chains: INPUT, OUTPUT and FORWARD.

INPUT chain applies to all packets destined to firewall.

OUTPUT chain applies to all packets originating from firewall.

FORWARD chain applies to all packets passing through firewall.

En example of iptables command for adding a rule to FILTER table:

iptables –A INPUT –i eth0 –s 122.33.44.5/24 –j ACCEPT

* accept all packets coming from interface eth0 and having source address 122.33.44.5/24
* -A means this rule is going to be appended to the bottom of the existing rules
* the rules will be applied in the order : the first rule that matches the packet determines the action for the packet

A few more examples :

iptables –A INPUT –i eth0 –j DROP

* drop all packets coming from interface eth0

iptables –L

* list all current entries of FILTER table

iptables –F

* flush all rules

iptables –D INPUT 2

* delete the second rule in the INPUT chain

iptables –I INPUT 1 –p–s 232.16.4.0/24 --dport 22 –j ACCEPT

* -I INPUT 1 : insert the rule at the top
* accept all TCP connections to port 22 (ssh) from 232.16.4.0/24

And of course, to see more of the iptables command use you can consult either internet or man pages of iptables.

Now that you’ve learned a bit about *iptables*, execute the following tasks:

1. Study the configuration of this lab and whether you have full communication between all of the nodes. You should start this lab with –f option.

Configure iptables on the *Router* node in such a way, that Server can’t reach PC1 anymore. Test with a ping.

Provide the *iptables* command that you used and a screenshot of the contents of your *iptables* on the *Router* node.

1. Delete all your entries from the *iptables* on the *Router*.

Provide a new screenshot of the *Router* *iptables* output.

Your Server should be able to ping PC1 again. Configure *iptables* on the PC1 node in such a way that server is not able to ping PC1 again.

Provide the command that you used and a screenshot of the contents of your *iptables* on the *PC1* node.

1. Delete all your entries from the *iptables* on *PC1*.

Wait for a TCP connection on a port bigger than 1024 by using netcat :

nc –l –p <port\_number>

Test from the *Server* that your connection works.

nc <IP\_ADDRESS\_OF\_PC1> <port\_number>.

Stop nc.

Configure your firewall on *PC1* in such a way that nc can’t reach your port <port\_number> anymore.

Test by reissuing the nc commands.

Provide the *iptables* command that you used on *PC1* and screenshot of unsuccessful communication. Provide a screenshot of a successful ping form *Server* to *PC1*, as this should still work.

1. As a variation of the exercise above you can now with your nc simulate a web server on the *Server* node. To be able to do it, you have to listen to the right port for HTTP and specify a file that your “web server” is going to serve. For this you can use index.html which is in your /var/www directory.

You can use index.html as an input to your nc command like this:

<”your nc listening command”> < /var/www/index.html.

Go to *Server* node and try this command. Then go to your *PC1* node and start links text browser by issuing:

links <IP\_ADDRESS\_OF\_YOUR\_SERVER>.

You should be able to see the known page.

Now configure *iptables* of your *Server* node in such a way that links can’t connect anymore.

Provide listing of your *Server* *iptables* and screenshot of the links browser trying to connect to your “web browser”.

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. |