(Using Certificates)

- Confidentiality and Secure Sockets Layer operation
  Man pages for Wireshark
- Preparing a server for secure communication
  http://www.securityfocus.com/infocus/1818 (Good article about how to set up SSL support for Apache web server, part 1.)
  http://www.securityfocus.com/infocus/1820 (Good article about how to set up SSL support for Apache web server, part 2.)
  http://www.openssl.org/docs/HOWTO/certificates.txt (How to create certificates using OpenSSL.)
  http://www.openssl.org/docs/HOWTO/keys.txt (How to create keys using OpenSSL.)
  http://www.openssl.org/docs/apps/openssl.html (The OpenSSL application commands manual.)

5 Laboratory assignments

There are three parts of this lab and all parts should be done in the Linux (StuDAT) environment.

Assignment 1: Authentication, digital certificates and chains of trust. In the electronic commerce, it is important for clients to be confident that transactions are secure. This means that the client can verify the identity of the e–commerce server and that the transactions between the client and the server are protected from eavesdropping. One way of providing authentication is to use a Public Key Infrastructure (PKI) and to let the e–commerce servers authenticate themselves by providing trusted certificates to the clients. The clients can then benefit from the PKI when verifying the identity of the server. In this assignment we will use a web browser to further investigate the details of certificates and PKI. This assignment requires the use of an ordinary web browser. The instructions are written for the Mozilla Web Browser but any browser of your choice will do as long as it supports certificates.

We will now look into what happens on the user level when a browser connects to a web site providing security. We will focus on events that are visible to the user.

Q1: What are the possible reasons for this security warning?

The security warning holds information about the web server's certificate. Answer the following questions by further investigating the certificate, i.e. click the button "View..." under "Add Exception...", which can be found under "I Understand the Risk".

Definition: OpenSSL is an open-source library containing cryptographic tools.

1. OpenSSL usage :

Install openssl (or ensure it is installed).
Syntax: *openssl command [options] [arguments]*
In this lab, we will use the following commands:
passwd, enc, genrsa, x509, dgst, req, verify.

2. Encoding with base64 Definition:

Base64 is an encoding scheme which uses 65 printable characters (26 lower-case letters, 26 upper-case letters, 10 digits, characters '+' and '/', and special character '='). Base64 allows to exchange data with limited encoding problems.

Syntax: To encode with base64, the following command is used:
openssl enc -base64 -in input-file -out output-file
To decode with base64, the following command is used:
openssl enc -base64 -d -in input-file -out output-file

- Encode a text file containing an arbitrary password with base64, and send it to your lab partner. Your lab partner has to find your password from this file.
- Is base64 a safe way to protect a password?

3. Password files Definition:

File /etc/passwd contains information on users. By default, the file is readable by every user.
man 5 passwd

Definition: File /etc/shadow contains information on the password of users. By default, this file is readable only by the super-user.
man 5 shadow

Create an user user1 with password pass1 (for instance) using the adduser command.
man 8 adduser

In file /etc/shadow, identify the field containing the password of the user. Note that the password is encrypted.

Definition: The password can be encrypted using DES or MD5. If the field containing the password starts with 1, the password is encrypted using MD5, otherwise it is encrypted with DES. The salt (which is an additionnal randomness) used for the password occurs before the next $. Finally, the encrypted password follows the last $.
Syntax: openssl passwd [options]
Where the possible options are:
-crypt: to encrypt
-1: to change the standard encryption algorithm from DES to MD5
-salt salt: to add the salt.

- From the (known) password and the salt from /etc/shadow, obtain the field containing the encrypted password using openssl.
- Change the pasword from user user1 and use only four lower-case letters. Implement a program that takes as input the encrypted password and the salt from /etc/shadow, and that finds out (using brute force) the password. How long does it take to crack the password?
- Would it be possible to adapt your program so that the salt is no longer required? How long would it take to run?

4. Encryption

Syntax: To encrypt, we can use command enc. To decrypt, we can use command enc with option -d. To use DES, we can use option -des. To use triple-DES, we can use option -des3. We will also use option -nosalt in the following.

- Encrypt an arbitrary file and decrypt it using the good password (with DES and no salt).

- Encrypt an arbitrary file and try to decrypt it using a wrong password. Remark: openssl detects that the password was wrong. We will try to see how openssl could detect it.
- Compare the size of a plaintext and the corresponding cipher. Explain the difference.
- Consider two different files plaintext1 and plaintext2 (with different sizes). Encrypt these two files with a password. You obtain files cipher1 and cipher2. Decrypt these two files with *option -nopad*. You obtain plaintext1nopad and *plaintext2nopad.*
- With an hexadecimal editor (such as tool xxd), study the difference between plaintext1 and plaintext1nopad, and between plaintext2 and plaintext2nopad. Explain the differences.
- Consider another file plaintext3. Encrypt it, and ask your lab partner to decrypt it using option -nopad (either with a good or a wrong password) into a file password3nopad. Without comparing plaintext3 and plaintext3nopad, can you tell whether the password was good or not?

5. Encryption with RSA Key generation Syntax:

(a) The key generation is performed using:
openssl genrsa -out key.pem size

(b) Create a private RSA key of 50 bits, and save it into file mySmall.pem.

(c) Syntax: To obtain information on a key, we can use:
openssl rsa -in key.pem -text -noout

(d) Display information on the private key you generated.

(e) Should the private key be stored as plaintext? Is your key safely stored in file mySmall.pem? Break the following 50-bits RSA key otherSmall.pem by finding the two factors of n (called the modulus)
-----BEGIN RSA PRIVATE KEY-----
MDcCAQACBwLXEvQUSR0CAwEAAQIHAf9IXGeOgQIEAcx/UQIEAZQyDQIEAK3/cQIE AKWD/QIDPAMq -----END RSA PRIVATE KEY-----

- Crack your own RSA key by factorizing the modulus. Why could you crack it? Syntax: In order to save the key in a safe way, a symmetric encryption algorithm (such as triple-DES) can be used:
openssl genrsa -des3 -out private-key size
- Create a private key *private1.pem* of 1024 bits, without encryption. Create a private key *private2.pem* of 1024 bits, with encryption.
- Retrieve the information from keys *private1.pem* and *private2.pem.*
Syntax: To retrieve the public key associated with a private key, we use:
openssl rsa -in private-key -pubout -out public-key
- Create a public key associated to the private keys *private1.pem* and *private2.pem.*
- Should we store the public key in plaintext?
Syntax: To display information on a public key, we have to use option -pubin in order to inform openSSL that the input file only contains public data. Retrieve the information of the two public keys

Encryption and decryption Syntax:

a) To encrypt using a public key, we use:
openssl rsautl -encrypt -in plaintext -inkey public-key -pubin -out cipher
b) To decrypt, we use option

-decrypt.

c) Encrypt and decrypt a small file.
d) Encrypt a large file. What happens? Why?
e) Ask your lab partner for his/her public key. Encrypt a short message with his/her public key. Give the cipher to him/her, and ask him/her to decrypt it.
   Syntax: To encrypt using DES, we use:
   openssl enc -des -in plaintext -out cipher
f) To decrypt with DES, we use:
   openssl enc -des -d -in cipher -out plaintext

- Ask your lab partner for his/her public key. Encrypt a long message with a symmetric encryption algorithm (such as DES), by using an arbitrary password. Encrypt the password with the public key of your lab partner. Send to your lab partner both the encrypted password and the cipher. Ask him/her to decrypt the message.

Signatures

g) Syntax: To hash a file, we use:
h) openssl dgst -md5 -out hash file
i)
   Syntax: To sign a hash, we use:
j) openssl rsautl -sign -in hash -inkey key -out signature

k) Should we use the public key or the private key for a signature? Syntax: To verify a signature, we use: openssl rsautl -verify -in signature -pubin -inkey public-key -out hash2 Exercise 5.16: Sign a message.
l) Verify the signature. You can use the tool diff.
m) Create three messages. Sign all of them. Slightly modify one or two of them, and send them to your lab partner, together with the signatures. Ask him/her to determine which messages were modified.

6. Certificates

Certificate generation

a) Syntax: To generate a certificate, a request has to be created first:
   openssl req -new -key key -out request
   Remark: The key has to be large enough to sign the certificate. Generally, a 1024-bit key should be sufficient.

b) Create a certificate request called user-request.pem, concerning an entity called user. This request is waiting to be signed by a certification authority.
   Syntax: To visualize a certificate request or a certificate, we use:
   openssl x509 -in certificate -text -noout
c) Visualize the content of the certificate request.
   Create a pair of keys for the certification authority. Create a certificate request called carequest.pem and concerning the certificate authority (referred to as CA).

   Syntax: To autosign a certificate request, we use:
   openssl x509 -req -in request -signkey private-key -out certificate

d) Autosign the certificate request of CA in a certificate called ca-certificate.pem.

e) Visualize the certificate ca-certificate.pem.
Syntax: To sign a certificate request, we use:
openssl x509 -days duration -CAserial serial -CA certificate -CAkey key -in request -req -out certificate
The file serial contains a number stored in hexadecimal (using an even number of hexadecimal digits).

f) Sign the certificate request user-request.pem to produce a certificate user-certificate.pem.
Syntax: To verify a certificate, use:
openssl verify -CAfile ca-certificate certificate

g) Verify the certificate user-certificate.pem.

h) Modify the content of the certificate user-certificate.pem and try to verify it.

Certification chain

- Create two certification authorities CA1 and CA2. CA1 is the root certification authority, and CA2 is certified by CA1.
- Create a certificate request and sign it by CA2.
- Verify the complete certification chain.

Final Assignment on OpenSSL

Generate two files like in the following.

file.txt

```
000000000000000
111111111111111
222222222222222
000000000000000
111111111111111
222222222222222
000000000000000
111111111111111
222222222222222
```

modified_file.txt (only one DIGIT change)

```
000000000000000
111111111111111
222222222222222
000000010000000
111111111111111
222222222222222
000000000000000
111111111111111
222222222222222
```

Use AES encryption and mode of operation ECB and CBC to analyze the cipher text.

Use syntax like in the following

```
$ openssl  aes-256-ecb -e  -in file.txt -out cipher_ecb.bin  -nosalt
```

After use

```
$xxd cipher_ecb.bin
```

Observe the output you will able to understand the known plaintext

Same way find another mode of operation CBC.

```
$ openssl  aes-256-cbc -e  -in file.txt -out cipher_cbc.bin  -nosalt
```

After use

```
$xxd cipher_cbc.bin
```

Find the observations