**Project Description for Computer and Network Security**

**A Secure Image Repository**

April 23, 2017

Team Members: Riccardo Gulia, Terry Griffin, SrividyaKuppa VenkataNaga, Bharadwaj Nallavelli, Bill Stronge

**Introduction**

Our project is a client/server application that provides a secure image repository. This application allows authorized users to securely upload and download images to/from the repository. The uploaded images are signed so that they are associated with the person who uploaded them. The signing can help resolve copy right issues, both in helping to identify who owns an image and to identify anyone who tries to take credit for someone else’s images. In addition to uploading, authorized users may download images. The downloaded images are accompanied by a signature file which can be used to verify the image and identify the person who uploaded it.

An example of someone who might use such an application is TMZ. They could authorize their Paparazzi photographers to upload their images of Angelina Jolie and Brad Pitt to the secure image repository. The Paparazzi would know that each image they uploaded would be owned by them.

The project includes elements of the following topics covered in this class:

1. The use of Diffie-Hellman public key/private key exchange to agree on an encryption key
2. Encrypting/Decrypting the communication over the socket using AES-128
3. User authentication at the server using X.509 Public Key Certificates
4. Using SHA-256 to sign the image downloads so that the downloader can verify the image and identify the owner.

In operation the client and server behave similar to a secure FTP client/server, with the added feature of creating the signature file for each upload and providing a way to verify downloaded files.

The project makes use of the OpenSSL crypto library, version 1.0.2g, for the low level cryptographic functions. The DH\_ family of routines is used for the Diffie-Hellman key generation. The EVP\_ family is used for encrypting the socket communication. The X509\_ family is used for managing the certificates, and the SHA128\_ family is used for generating and verifying the file hash. The openssl command line tools are used for generating the X509 certificates.

The work breakdown of the project was as follows:

* Riccardo Gulia – AES128 encryption/decryption for secure socket communication. Setup and maintenance of git source control and Jenkins automated test servers.
* Terry Griffin – Basic client/server structure and Diffie-Hellman key exchange.
* Bill Stronge – X.509 public key certificate generation and client/server authentication handshake.
* SrividyaKuppa VenkataNaga and Bharadwaj Nallavelli – SHA256 hash generation and verification for the signature file.

**Structure**

The client process is named pictstor and the server process is pictstord. The server follows the typical client/server architecture, creating a TCP socket and waiting on a port for a client connection and then forking a copy of itself to service the client.

After the two processes have established a socket connection they perform a Diffie-Hellman key exchange. The parameters for generating the Diffie-Hellman keys, two large primes, have been pre-generated and are passed from the server to the client. Both sides then generate key pairs and pass the public key to the other party. Both sides then use their private key and the other side’s public key to generate the shared key.

Once the shared key is generated, all further communication over the socket is encrypted using AES-128.

The next step in the connection process is client and server authentication. A 4-way handshake is used to verify the client’s identity, ensure that the client is authorized to use the service, and verify the service on the client side. X.509 public key certificates are used by the client and server to for authentication.

Certificates for the server and two test users are included in the project’s data directory. A README file containing instructions for creating additional X.509 certificates using openssl is also included.

At this point, secure communication has been established and the client and server have verified each other’s identifies. The client can now use the service much like a typical FTP server. The commands we have supported are:

|  |  |
| --- | --- |
| put <file name> | Copies a file to the server. A signature file will also be created on the server using the name <file name>.sig |
| get <file name> | Copies a file from the server to the local directory. The signature file,  <file name>.sig will also be copied. |
| verify <file name> | Verifies <file name> using the <file name>.sig file. |
| ls | list the files in the server’s current directory.  .sig files are not included in the file list |
| rm <file name> | remove a file from the server |
| mkdir <directory> | create a directory on the server |
| cd <directory> | change the server’s current directory |
| rmdir <directory> | remove a directory on the server |
| lls | list the files in the local directory |
| lcd | change the local current directory |
| quit | quit the program |
| help | display a list of supported commands |

When a file is uploaded (using the put command) an additional file containing a signature file is also created on the server. The signature file contains the name of the uploading user, a timestamp, and an SHA256 hash. The hash is generated using the user’s name, the timestamp, and the contents of the uploaded file. (Including the name and timestamp in the hash prevents someone from later modifying the signature file).

When a file is downloaded (using the get command) the signature file is also downloaded. The user can then, or later, use the verify command to verify that the file is indeed the one uploaded by the original user. If the verification process is successful the name of the original user and the timestamp are displayed.

**Project Directory Structure**

The directory structure of the project is:

|  |  |
| --- | --- |
| data | Includes the X.509 certificates and the pre-generated Diffie-Hellman parameters needed for key generation. |
| testfiles | Includes sample files used for testing |
| src | The project source code |
| src/stest | A test directory for running the server. Created by test\_setup.sh |
| src/user1 | A test directory for user1. Created by test\_setup.sh |
| src/user2 | A test directory for user2. Created by test\_setup.sh |

**Building the Project**

From the src directory, the project can be built using make. This will create both the client and server programs (pictstor and pictstord).

However, both programs expect to find a data directory holding the runtime information they need, such as their certificates. The easiest way to setup a test environment is to run test\_setup.sh. This will build the project (run make) and then create three directories, stest, user1, and user2. data subdirectories will be created and populated with the necessary files for the server and both users.

The keys and certificates used in the tests are stored in the top level "data" directory. A README file in that directory describes how the credentials are created

**Running the Program**

Running the program requires two terminal windows. In one window run the server, using pictstord <port>. After running test\_setup.sh in the src directory:

$ cd stest

$ ./pictstord 5610

In a second window run the client. To use the program interactively use pictstor <host> <port>. (The program can also be run giving a single command on the command line using pictstor <host> <port> <command>.) After running test\_setup.sh in the src directory, the following commands will copy the ducks.jpg file to the server a user1.

$ cd user1

$ ./pictstord localhost 5610

> mkdir birds

> cd birds

> put ducks.jpg

sending file blocks ... ####################################################################

(lines removed for clarity)

send\_file done

> ls

Apr 23 2017 10:30 3978352 ducks.jpg

> quit

At this point the ducks.jpg file and its signature file exist in the server’s file store. Now user2 can retrieve and verify the file:

$ cd ../user2

$ ./pictstor localhost 5610

> cd ../user1/birds

> ls

Apr 23 2017 10:30 3978352 ducks.jpg

> lls

data pictstor

> get ducks.jpg

receiving file blocks ... #####################################################################

(lines removed for clarity)

receive file done

receiving file blocks ... #

receive file done

> lls

data ducks.jpg ducks.jpg.sig pictstor

> verify ducks.jpg

File verified. Uploaded by user1 on Sun Apr 23 10:30:41 2017

> quit

In this example both the server and client were run on the same machine, and so localhost was used for the hostname. The project will work equally well with the client and server running on different hosts.

For simple automated testing, there is a test script run\_tests.sh in the user1 directory. This script will test the put, get, and verify commands using the three sample files in the user1 directory.

An excerpt of a full test sessions can be found in the three text files in the documentation directory:

*FTP-Server\_tests.txt* - transcript of the FTP server session.

*user1\_tests.txt* - Transcript of the execution of the run\_tests.sh script, which acts as user1. It performs the PUT of three files on the server and then gets them back and verifies their signature.

*user2\_tests.txt* -- Transcript of the execution of a test for user2. It browse the file storage, finds a file to download belonging to user1 and after doing a GET, it verifies its signature.