GROUP 1

ANTHONY DEBRUYN
BRIAN DELHAISSE
ALEXIS LEFEBVRE
AURÉLIEN PLISNIER

Computer Security

Project 2

Services and Authentification

Prof. OLIVIER MARKOWITCH AND NAÏM QACHRI INFO-F-405

2ND DECEMBER 2013

1 Threat modeling

1.1 Information document

Step of the development: Version 1.0

Director: Naïm Qachri

Participants: Anthony Debruyn, Brian Delhaisse, Alexis Lefebvre and Aurélien Plisnier.

Reviewer: Naïm Qachri

Location: ULB

Description: Two web services are proposed: a blackboard and a keychain service. The authentification of the web services and the clients is realized by an authentification server. A small website will be made to allow an admin to manage the stored data, and the clients who can have access to those web services.

1.2 Use Scenarios

ID: 1

Description: The services will be accessible from anywhere and only for authorized clients. There are two services: a secure virtual blackboard on which authorized clients can publish posts and a virtual keychain server in which authorized clients can securely store their passwords.

ID: 2

Description: An admin will be able to manage the authorized users and their RSA certificate list via a website.

1.3 External Dependencies

ID: 1

Description: The security of the application depends on the AES and RSA keys generator used.

ID: 2

Description: The security of the application depends on the quality of the certificates generated by OpenSSL.

1.4 Implementation Assumptions

ID: 1

Description: The servers and clients are all aware of the communication protocol used in this project and behave accordingly.

1.5 External security notes

ID: 1

Description: The user is responsible for the strength of his password, the server doesn't have an integrated strength enforcement system that tells the user whether his password is strong enough or not.

1.6 Internal security notes

ID: 1

Description: We trust the implementations of the encryptions and decryptions used in this project, but we read the javadoc associated to understand them.

1.7 Levels of trust

ID: 1

Name: The identity of the web server process.

Description: This identity is used to authenticate a web server in its database when it stores and retrieves information.

ID: 2

Name: A registered user.

Description: The user uses his ID and certificate to authenticate and access the web services.

ID: 3

Name: The admin.

Description: The admin uses his PHP application to manage the authorisation server's database.

1.8 Assets

ID: 1

Name: User's data.

Description: The user ID and certificate.

Level of Trust: A user with verified token access (2), the admin (3).

2 Analysis of the model

2.1 Data flow diagrams

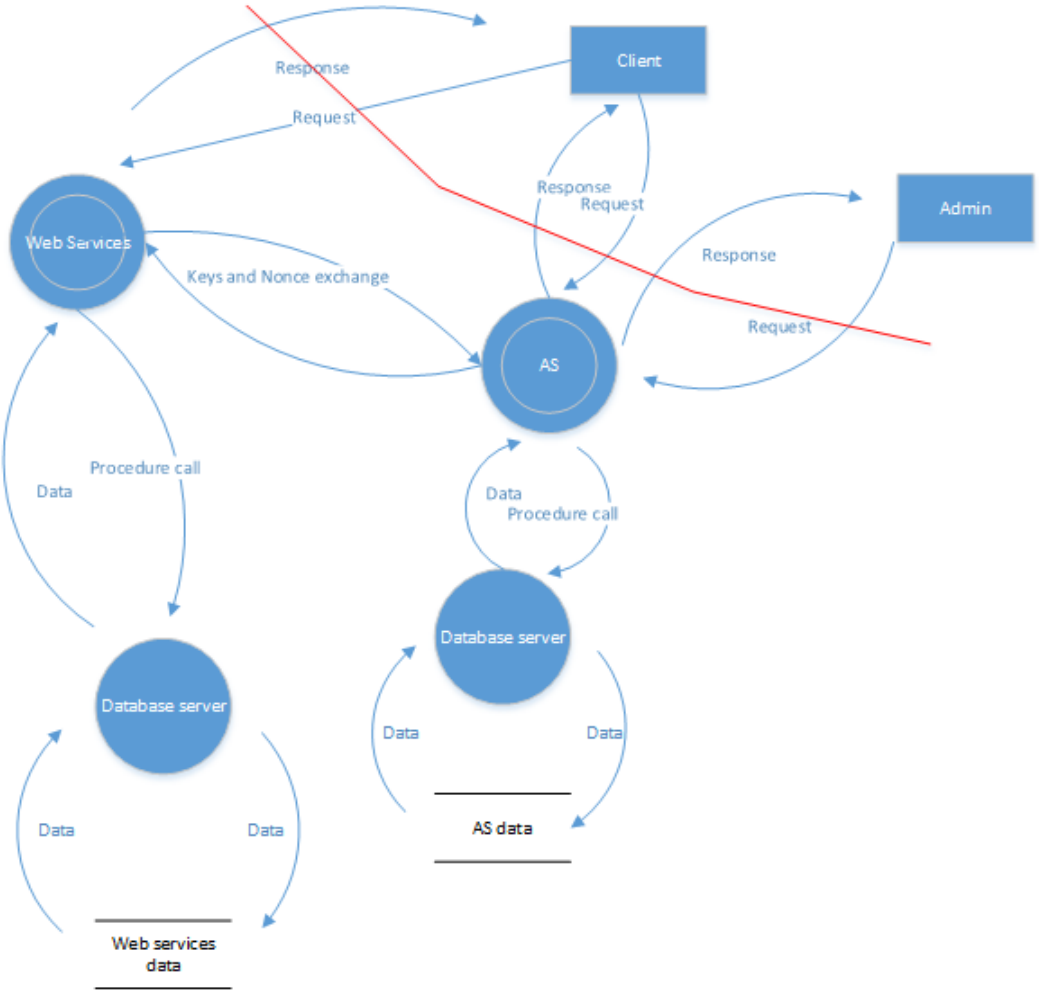

Figure 1: data flow diagram

2.2 Documentation of the potential threats

ID: 1

Name: Denial of service attack.

Description: Many requests are being sent by the attacker(s) to the server in order to saturate/overload this one.

STRIDE classification: D (Denial of Service)

Entry points: The listening port of the web server.

Assets: None

Threat tree: see fig. 2

ID: 2

Name: Gain the client's privileges

Description: The attacker acquires a private key and the ID from a valid user (client).

STRIDE classification: E (Elevation of privilege.)

Entry points:

Assets: User's data.

Threat tree: see fig. 3

ID: 3

Name: Gain the admin privileges

Description: The attacker acquires the admin's password.

STRIDE classification: E (Elevation of privilege).

Entry points: Admin

Assets: Admin's password

Threat tree: see fig. 4

2.3 Threat trees

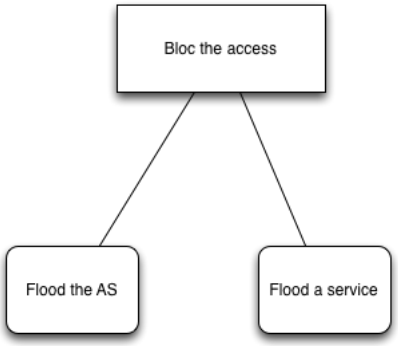

Figure 2: Threat tree

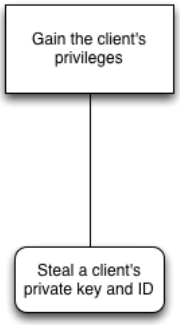

Figure 3: Threat tree

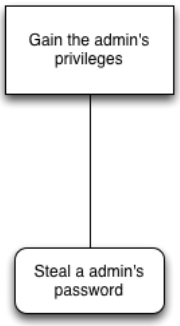

Figure 4: Threat tree

3 Implementation

3.1 Web Interface for the admin

A simple web interface allowing the admin to access and update the authorisation service database was made. It is accessed using https. The https connection was set up using the instruction given during the last practicals.
The data stored in the AS database was not encrypted due to severe lack of time. As a proof that the https connection was set up in time, a printscreen of our web app was taken on Linux and added into the directory PHP.

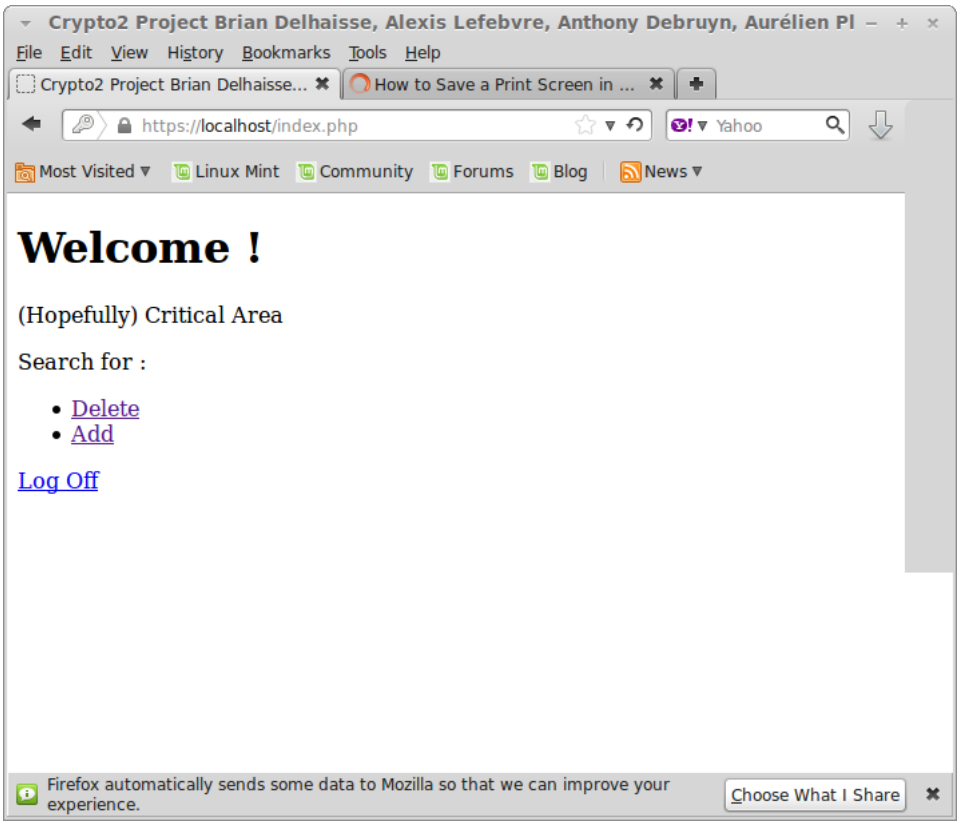

Figure 5: The web interface for the admin

Here is a short overview of the php files:

add.php : Presents a form allowing the admin to type in a user ID and the related certificate to add.

computeAddUser.php : Runs a SQL statement to insert the new entry if the typed ID does not already exists in the database or to update the existing user if the ID already exists in the database.

computeDelete.php : Runs a SQL statement to delete the entry corresponding to the specified ID.

footer.php : A useless footer.

head.php : A useless header.

identityTest.php : Tests the password typed in by the admin in the logOn page.

index.php : Main page of the application, presents the welcome message and the options.

logOff.php : Log off.

logOn.php : Log on by typing in the password (chicken).

menu.php : Contains the options: add, delete, logon and logoff.

3.2 Authorization Server

The role of the authorization server is described in the assignments.
Here is a short overview of the different classes:

Main : Creates a link to the MySQL database and launches the authorization server.

connection.AuthorisationServer : The AS server: accepts new connections in a while(true) loop. This class is also able to transmit the AES session key shared by a client and a web service to the web service.

connection.RSASecuredService : Thread created every time a client contacts the AS. The thread handles the mutual authentication by sending the AS certificate to the client and by implementing the Needham-Schroeder protocol.
Once the authentication made, the thread generates and distributes AES session keys. These keys have a cryptoperiod of 2 hours and should be usable only during one session. If a session exceeds 2 hours time, new session keys should be generated. This was not fully implemented due to severe lack of time.

crypto.RsaKey : This class loads the AS private key and the admin certificate using OpenSSL generated files. The AS certificate is also loaded from the database.
A copy of the admin's certificate is stored in each app's src directory to enable them to check the validity of the signature of other parties certificates.

dataBase.DbLink : This class handles the connection to the MySQL database and allows to retrieve certificates associated to given ID. ID 0 is the AS, ID 1 is the blackboard and ID2 is the keychain. ID higher than 2 are user's ID.

3.3 1st Web Service: Blackboard

This web service allows to a lient to send a string. The latter is then printed in the service console.

Main : Creates a link to the MySQL database and launches the blackboard server.

connection.BlackboardWebService : Initiates a connection to the authorisation server. Once the service is authentified and has received an AES key to communicate with the authorisation server, it enters a while(true) loop to accept connections from clients.
A list of valid clientID / AES keys is maintained to allow the web service to communicate with its clients.

connection.BlackboardToAuthorisationServerUsingRSA : This class handles the connection and authentication with the authorisation server. The authorisation server's certificate is first received and its signature verified using the stored admin's certificate. Challenges are then traded to allow both parties to authenticate using the Needham Schroeder protocol. Services and clients to not have to send their certificates to the authorisation server for verification since their certificates are already stored in the authorisation server's database.

connection.BlackboardAESSecuredService : This thread is started every time a client connects to the blackboard. If the client is the AS, the service receives a client ID and the associated AES key and stores it in the BlackboardWebService. If the client is a user, a blackboard service is handled to him using AES link.

crypto.RsaKey : This class loads the blackboard private key and certificate and the admin certificate using OpenSSL generated files.
A copy of the admin's certificate is stored in each app's src directory to enable them to check the validity of the signature of other parties certificates. A method verify is implemented to do so.

dataContainers.IDAES : A simple data structure containing ID and AES keys couples. It is used by the web service server to store a list of the connected clients and their AES key.

3.4 2nd Web Service: Keychain

This web service allows a client to store his/her passwords in a secure database and to access them later.

Main : Creates a link to the MySQL database and launches the keychain server.

connection.KeychainWebService : Initiates a connection to the authorisation server. Once the service is authentified and has received an AES key to communicate with the authorisation server, it enters a while(true) loop to accept connections from clients.
A list of valid clientID / AES keys is maintained to allow the web service to communicate with its clients.

connection.KeychainToAuthorisationServerUsingRSA : This class handles the connection and authentication with the authorisation server. The authorisation server's certificate is first received and its signature verified using the stored admin's certificate. Challenges are then traded to allow both parties to authenticate using the Needham Schroeder protocol. Services and clients to not have to send their certificates to the authorisation server for verification since their certificates are already stored in the authorisation server's database.

connection.KeychainAESSecuredService : This thread is started every time a client connects to the keychain. If the client is the AS, the service receives a client ID and the associated AES key and stores it in the KeychainWebService. If the client is a user, a keychain service is handled to him using AES link.

crypto.RsaKey : This class loads the keychain private key and certificate and the admin certificate using OpenSSL generated files.
A copy of the admin's certificate is stored in each app's src directory to enable them to check the validity of the signature of other parties certificates. A method verify is implemented to do so.

database.DbLink : Handles the connection the the keychain's database. Implements methods to insert and retrieve data.

dataContainers.IDAES : A simple data structure containing ID and AES keys couples. It is used by the web service server to store a list of the connected clients and their AES key.

3.5 Client

The client is a local application used by one of the webservices costumer. It consists in a console application where the client can input its choices and commands to the webservices.

The application first tries to connect to the authorisation server and to get its public key (RSA). This public key is always verified to check if it has been signed by the CA (admin). If yes, the Needham-Shroeder protocol can begin (the authorisation server already got the key of the client in its DB).

Then the NS followed as seen in the guidelines for a client connection. The id of the service wanted by the client is sent to the AS to initiate a connection with the required service. After the protocol is finished, the communication between the 2 parts (client-WSx) can begin under protection of a AES encrypted channel.

Main : creates an instance of the client application.

connection.Client : Connects to the authorisation server to mutually authenticates, ask the user what service he wants and initiates the connection to the chosen service.

connection.ClientToAuthorisationServerUsingRSA : This class handles the connection and authentication with the authorisation server. The authorisation server's certificate is first received and its signature verified using the stored admin's certificate. Challenges are then traded to allow both parties to authenticate using the Needham Schroeder protocol. Services and clients to not have to send their certificates to the authorisation server for verification since their certificates are already stored in the authorisation server's database.

crypto.RsaKey : This class loads the client private key and certificate and the admin certificate using OpenSSL generated files.
A copy of the admin's certificate is stored in each app's src directory to enable them to check the validity of the signature of other parties certificates. A method verify is implemented to do so.

4 Remarks :

For the cryptoperiod, we have left this problem for the end of the project, we haven't had enough time to do a full investigation about this.

We also haven't had the time the hash the password sent by the client to the keychain.

The authorisation server's database is also not ciphered. We were considering the idea of using a different AES key to cipher / decipher each entry of the table. This would have been investigated and implemented with more time.