INFO-F-405 – Computer Security – Project 2

Group 3 – Vadim Baele, Marwan Bellouti, Quentin Stievenart, Damien Wiltgen

December 16, 2012

1 Implementation choices

In this section, we describe our implementation choices, their advantages and disadvantages.

1.1 Cryptographic methods used

Unsecure cryptographic methods (MD5, DES, ...) were avoided, and the following methods were chosen:

- AES-256 in CBC mode with a random IV for encrypting documents;
- SHA256 as a cryptographic hash function;
- RSA with 2048-bit keys for signing and encrypting symmetric keys.

In the rest of this document, when we refer to *encrypt* something, it means encrypting it with AES-256 with a random IV. When we refer to *hash* something, it means hashing it with SHA256. When we refer to *sign* something or to *encrypt with the public key*, it respectively refers to sign or to encrypt with RSA using 2048-bit keys and SHA1 as the hash function.

1.2 Securing the data on the server

1.2.1 User passwords

The user passwords are stored on the database. There are multiple way to store them:

1. **plain password**: store the plain password, which should be avoided because anyone having (potentially malicously) read access to the database will know the password of every user;
2. **hashed password**: store the hash of the password using a secure cryptographic hash function, which has the disadvantage of having the same stored value for two identical passwords;
3. **hashed password with salt**: store the hash of the string formed by the password concatened to a salt, and store the salt used, which has the advantage of having different stored passwords for identical passwords.

The third method, which is the most secure, was chosen. The stored password is the hash of *name|password|salt*, where *name* is the name of the user, *password* its password, and *salt* a (pseudo-)random value. The '|' characters are there to have a different scheme from what is advised on the internet, to avoid finding the password from the hash by using a precomputed online table containing such hashes.

1.2.2 Files

We found multiple way to encrypt files on the server:

1. Using symmetric encryption with only one key, stored on the filesystem or in the application code source. This has the disadvantage that when the server is compromised, the encryption key can be found and every file can then be decrypted.

2. Using symmetric encryption with one key (the same for all files) and a salt, where the salt is stored in the database and is different for each file. This has the advantage that if the database server is separated from the file server, having one of the two server compromised does not compromise the files

3. Using symmetric encryption with a different key for each file, saving the key in the database. This has the same advantage as the second method.

4. Encrypt the file with the user's public-key. This has the disadvantage that for big files, the encryption might be slow (asymmetric encryption is slower than symmetric encryption) and that it requires the file to be saved multiple times (once for each user) if it is shared among multiple users. However, if the server is compromised, the files are not decryptable by the attacker.

5. Using a method similar to PGP: each file is encrypted with a random key, the random key is encrypted with the user's public key and saved on the filesystem. It is faster than the method 4. This also has the advantage that in any case, the files on the server could not be decrypted by somebody who has access to all of the server's file and databases. However, it requires more manipulations by the user.

Even though the last method complicates the user manipulations, it was chosen, because the program given with the application could be extended to do most of the work for the user (though it is not the case in the program given, it can be extended later to do so).

To summarize how this method works:

- When the user signs up to the application, the server creates two pairs of keys (one for generating and verifying signature, the other for the encryption and decryption) and gives them to the user, which should copy them into the program given with the application.

- When the user wants to upload a file, he uses the program to compute the file's signature, and upload the file with its signature to the server. The server then generates a random key, encrypt the file with this random key, encrypt the random key with the user's public key and saves the encrypted file, the encrypted key and the signature to the file system.

- When the user wants to download a file, he downloads the encrypted file, the encrypted key and the signature. He decrypts the secret key with its private key, decrypts the file with the secret key, and check that the resulting file matches the signature.

- When the user A wants to share a file, he downloads the encrypted secret key used to encrypt the file, and the public key of the user he wants to share the file with (B). He decrypts the encrypted secret key with its private key, encrypts it with B's public-key, and send the result to the server, which saves it.

The main disadvantage of this method is that it requires the user to do a lot of things, which might discourage users to use this service.

1.3 Revocation

1.3.1 Revocation of the certificate

When the certificate of the user is revocated, it is deleted and a new one is generated. The new certificate and private key are showed to the user, who should copy them into the Java program.

Since all the signatures of the user made with this certificate are now considered invalid (and cannot be verified since the old certificate has been deleted), the web service deletes them (it deletes all the signatures, because an user only have one certificate at a time).

If the user wants to sign the files that do not have a signature anymore, he can download them, decrypt them, sign them, and reupload them.

1.3.2 Revocation of the encryption key

The revocation of the encryption/decryption key pair is similar to the revocation of the certificate used for signing files, except that not only the signature are deleted, but also the files, because the old encryption key is not considered secure anymore. The user is thus strongly advised to download all his files before revoking the encryption key.

1.4 Libraries used

The following libraries were used during this project:

- `openssl` and `mcrypt` to do encryption and signature verification on the server's side with PHP. Only `openssl` could have been used, but the padding it uses when encrypting with `aes-256-cbc` is not the same as the one used in the Java program (the standard *PKCS #5*). `mcrypt` does not support AES but supports *Rijndael* and allows fine graining of the parameters. Thus, *PKCS #5* padding was implemented in PHP, and `mcrypt`'s Rijndael was used with 128-bit block size and 256-bit keys, which results in the AES cipher.
- PHP's *PDO* to communicate to the MySQL server avoiding SQL injections, through the use of prepared statements.
- *BouncyCastle* in Java to be able to read certificates in the PEM format (the format exported by OpenSSL).
- Java's *Crypto* library and *Security* framework to sign, verify signatures, encrypt and decrypt data in the Java application

2 Threat model

This threat model was done using the methods described by the OWASP project[1] and by Microsoft[2]. It is organized as in the Microsoft's Web Application Threat Model Template[3].

Table 1: Application description

Name	Virtual Safe
Version	1.0
Authors	Vadim Baele, Marwan Bellouti, Quentin Stievenart, Damien Wiltgen
Description	The application consists of a web service that allows the users to upload files on the service, knowing they will be securely saved. The users can then share files with other users, to allow other users to read those files. The service also have administrators, who have the responsibility to validate the user accounts. A program is made available to the users to manage their certificate and keys, and to sign and decrypt files.

2.1 Security objectives

The security objectives of the applications are:

- Prevent an attacker from obtaining files accessible by the application's users without having the authorization from the file owner.

[1] `https://www.owasp.org/index.php/Application_Threat_Modeling`
[2] `http://msdn.microsoft.com/en-us/library/ms978527.aspx`
[3] `http://msdn.microsoft.com/en-us/library/ff648866.aspx`

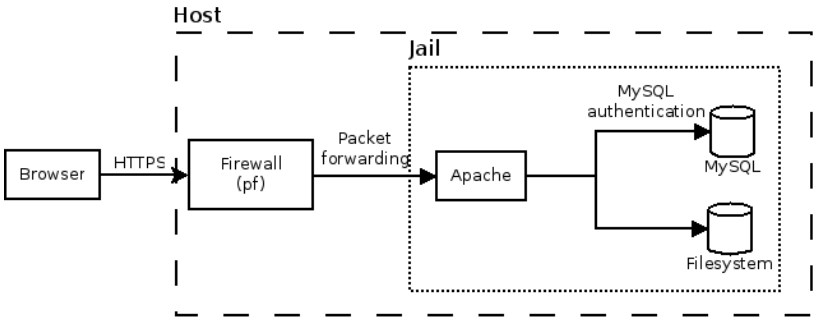

Figure 1: End-to-end scenario

- Protect the integrity of the files, ie. prevent an attacker to modify a file stored by some user without the user noticying it.
- Prevent an attacker to validate non valid users.
- Prevent an attacker to log in with another user account or with an administrator account.
- Prevent an attacker to obtain user or administrator passwords.

2.2 Application Overview

2.2.1 End-to-end scenario

The diagram representing the end-to-end scenario is represented in the figure 1.

2.2.2 Roles

There are two kinds of users of the web service: users and administrators. They have the following roles

- Administrators can validate user accounts, but cannot do anything else.
- Users can read and update their own data.

Other trusts levels can be identified, and are listed in the following table:

Table 2: Trust Levels

ID	Name	Description
1	Anonymous web user	A user who has connected to the application and is not logged
2	User with valid credentials	A user who is logged in the application using valid login credentials
3	Administrator with valid credentials	An administrator of the website who is logged in the application using valid credentials
4	Server administrator	The administrator who has access to the server with a root account
5	Database web user	The database user who has access to the tables ‘user’, ‘file’ and ‘share’ in read and write, and to the ‘admin’ table in read
6	Database admin user	The database user who has access to the table ‘admin’ in read and write
7	Web server user process	The process which executes the source code of the website

2.2.3 Key Scenarios

The key scenarios are listed in the following table: 4

Table 3: Usage scenarios

ID	Description
1	Anonymous user creates new (non-valid) account
2	Administrator validates a non valid account
3	User logs in with a valid account
4	User lists its own files
5	User lists the files shared by him
6	User lists the files shared with him
7	User downloads a file (either owned by him or shared with him)
8	User shares a file with another user
9	User uploads a file along with its signature
10	User revocates its certificate
11	User revocates its encryption key
12	User deletes a file he owns

2.2.4 Technologies

- **Operating System**: FreeBSD 8.0 with security patches
- **Web Server Software**: Apache 2.2.23 with OpenSSL 1.0.1 to encrypt HTTP traffic
- **Database Server Software**: MySQL 5.5.28
- **Development Languages**: PHP 5.4.7 and Java
- **Data Access Logic**: PDO (PHP)
- **Business Logic**: PHP classes

The external dependencies are explained in more details in the following table. The security of the application depends on the security of those external dependencies.

Table 4: External Dependencies

ID	Description
1	The application will run on a FreeBSD server running Apache. This server will be configured in a secure way.
2	The database server will be MySQL and it will run on the same FreeBSD server. The MySQL configuration will be hardened.
3	The connection between the web server and the database will only be done on the same machine.
4	The server is behind a firewall and the only communication available will be HTPS.
5	The application will be written with PHP. The default configuration of PHP will be hardened and the developpers will use good practice for writing the code.
6	The PHP libraries used are `mcrypt`, `openssl`, `pdo`.
7	The program given with the application will use Java and the Bouncy Castle library.

2.2.5 Application Security Mechanisms

- Sensitive data is stored encrypted (for files) or hashed (for passwords) using cryptographically strong methods.
- Every communication with the web server is made over an encrypted channel (HTTPS), which is considered secure.

- The web service is authenticated to the database using MySQL authentication scheme, and the MySQL server only accept connections on the local host.
- Server administration can only be performed from a SSH connection to the server's host, available only when having an accepted private key.
- The server's logs are monitored (by `logwatch` for example), and any suspect behaviour should be detected, the server's administrator will be notified.
- The server's software versions are monitored (by `portaudit` for example), and any security flaw should be discovered in one of the installed software, the server's administrator will be immediately notified and should perform what is necessary to ensure the application security.
- The web service relies on the fact that the user knows the good security principles and can securely manage their keys itself (ie. the user has to ensure that its keys are kept on its hard drive and no malicious user have access to its hard drive).

2.3 Application Decomposition

2.3.1 Trust Boundaries

Identified trust boundaries are:

- The perimeter firewall, were no data coming from the outside can be trusted.
- The database trusts calls from the identified web application.

2.3.2 Data Flows

The general flow of data for the web service is represented in figure 2.

2.3.3 Entry Points

Entry points are described in the following table

Table 5: Entry points

ID	Name	Description	Trust Levels
1	HTTPS port	The application will only be accessible via HTTPS. All the pages of the applications are available from this entry point	(1) Anonymous web user (2) User with valid credentials (3) Administrator with valid credentials (4) Server administrator
1.1	Admin login page	The page allowing administrators to connect using using their login credentials	(1) Anonymous web user
1.2	User validation page	The page allowing administrator to validate non-valid users	(3) Administrator with valid credentials
1.3	User login page	The page allowing users to connect using their login credentials	(1) Anonymous web user (2) User with valid credentials
1.4	Account creation page	The page allowing anonymous users to create a new account	(1) Anonymous web user
1.5	User file list page	The page listing the files that the user can access	(2) User with valid credentials

Continued on next page

Table 5: Entry points

ID	Name	Description	Trust Levels
1.6	User file download page	The page that let the user download a file	(2) User with valid credentials
1.7	User share page	The page allowing the user to share files with other users	(2) User with valid credentials
1.8	User file deletion page	The page that let the user delete one of its files	(2) User with valid credentials
1.9	User certificate revocation page	The page allowing the user to revocate its certificate	(2) User with valid credentials
1.10	User key revocation page	The page allowing the user to revocate its encryption key	(2) User with valid credentials
1.11	User file upload page	The page allowing the user to upload a new file	(2) User with valid credentials
1.12	User password modification page	The page allowing the user to change its password	(2) User with valid credentials
2	Database port	The database will only be accessible on the local host, through a socket file	(4) Server administrator (5) Database web user (6) Database admin user
2.1	Database connection	A connection made with valid credentials on the database server	(4) Server administrator (5) Database web user (6) Database admin user
3	Server's host	The server that host the *jail* where the application's server is	(4) Server administrator
3.1	Server SSH connection	A connection made with a valid user on the server's host	(4) Server administrator

2.3.4 Exits Points

Exits points are:

- The file list page, which displays the file name specified by the file owner as well as the name of the file owner
- The menu of the application (displayed on each page), which display the user name specified by the user

2.3.5 Assets

Table 6: Assets

ID	Name	Description	Trust Levels
1	User data	All data related to the users	
1.1	User login credentials	The user names and their passwords	(2) User with valid credentials (5) Database web user (7) Web server user process

Continued on next page

Table 6: Assets

ID	Name	Description	Trust Levels
1.2	User file list	The lists of the file that an user can access	(2) User with valid credentials (4) Server administrator (5) Database web user (7) Web server user process
1.3	User files	The files of the user stored in the file system	(2) User with valid credentials (4) Server administrator (7) Web server user process
1.4	User file encryption key	The symmetric key used to encrypt a file	(2) User with valid credentials (4) Server administrator (7) Web server user process
1.5	User private key	The key used by the user to sign its file	(2) User with valid credentials (7) Web server user process
1.6	User private encryption key	The key used by the user to decrypt its file	(2) User with valid credentials (7) Web server user process
2	Administrator data	All data related to the administrators	
2.1	Administrator login credentials	The administrator names and their passwords	(3) Administrator with valid credentials (5) Database web user (6) Database admin user (7) Web server user process
3	System	Assets relating to the underlying system	
3.1	Availability of the website	The web service should always be available and accessible by everyone	(4) Server administrator
3.2	Ability to execute code as a web server user	The ability to execute PHP code as a web server user	(4) Server administrator (7) Web server user process
3.4	Ability to execute SQL code on the databases	The ability to execute SQL code on all the databases of the server	(4) Server administrator

Continued on next page

Table 6: Assets

ID	Name	Description	Trust Levels
3.5	Access to the file system	The ability to access to the file of the file system with read and write permissinos	(4) Server administrator
4	Website	Assets retaling to the web service	
4.1	Access to the database server	Full access to the database server, for the database of the application	(4) Server administrator
4.2	Ability to execute SQL code as the 'web' user	The ability to execute code with the same privileges as the 'web' user	(4) Server administrator (5) Database web user (7) Web server user process
4.3	Ability to execute SQL code as the 'admin' user	The ability to execute code with the same privileges as the 'admin' user	(4) Server administrator (6) Database admin user (7) Web server user process
4.4	Ability to create new admins	The ability to create a new 'admin', which can then validate users	(4) Server administrator
4.5	Log and audit data	All the data logged by the server's processes	(4) Server administrator

2.4 Threats

The tables that follows lists the existing threats.

Table 7: Threat: Access to login credentials

ID	1
Description	Adversary get access to an user or admin login credentials
STRIDE classification	Elevation of privileges
Known mitigation	The security of the login credetials depend on the cryptographic methods used to store them and on the security of the transport layer (SSL). The only place where the login credentials are used is the *login* page and the *password modification* page. The password modification page requires the user to enter its previous password, so an attacker having access to this page is not able to change the user's password without previously knowing it. Moreover, all the user passwords are stored salted and hashed in the database, and users are not allowed to choose weak passwords (the passwords should be at least 10 characters long, with at least 2 non-alphabetical characters).
Entry points	(1.1) Admin login page (1.3) User login page (1.11) User password modification page (2.1) Database connection
Assets	(1.1) User login credentials (2.1) Administrator login credentials
Threat tree	Similar to the threat tree of the threat 2 (figure 3)

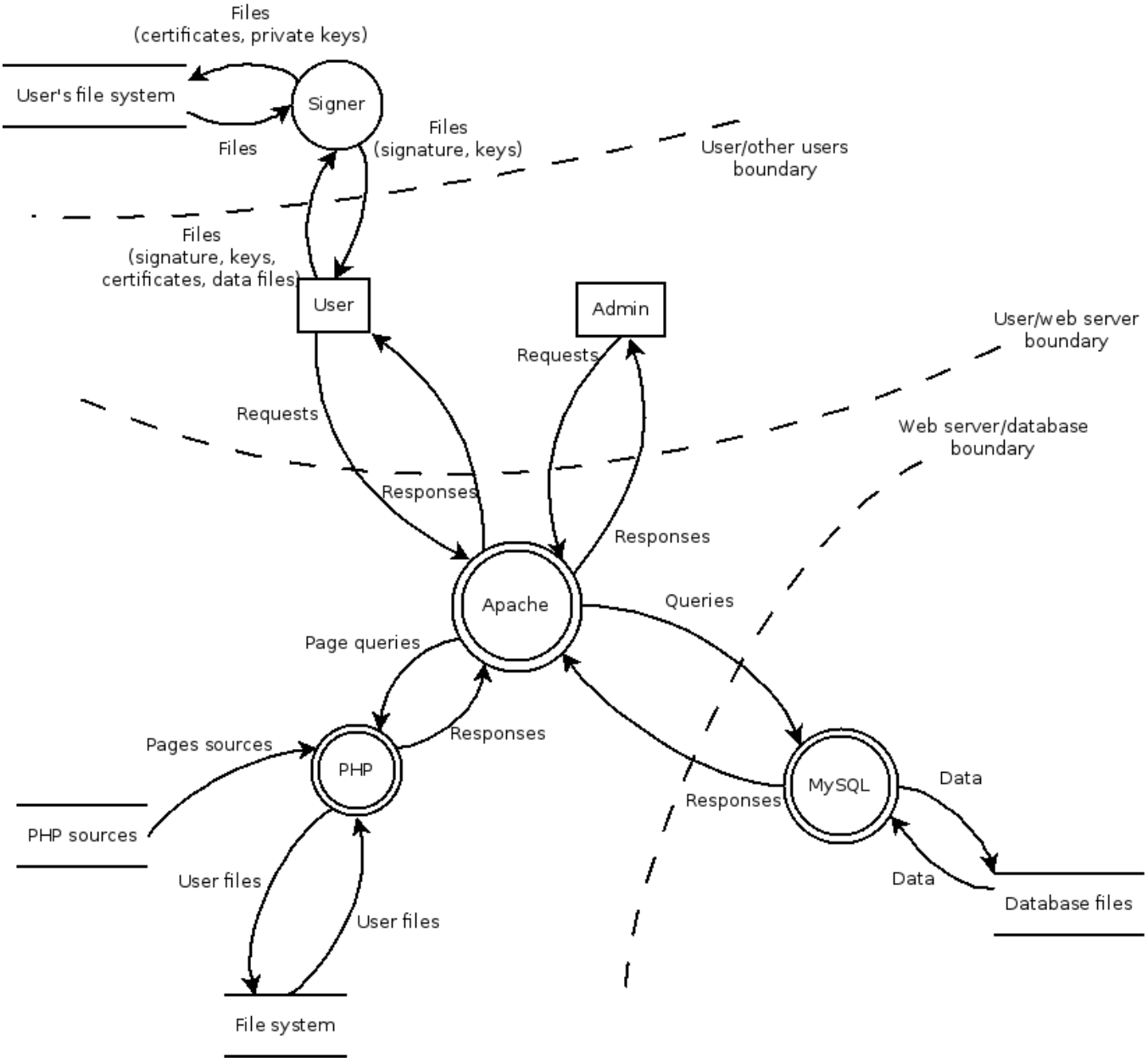

Figure 2: General dataflow of the web service

Figure 3: Threat tree of *user file list disclosure* threat

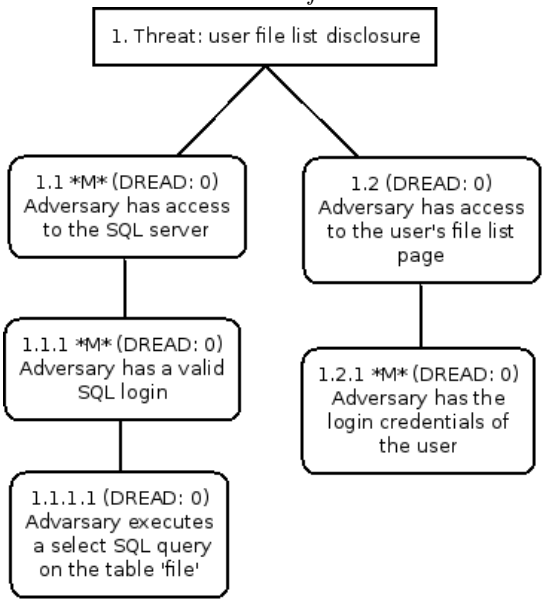

Table 8: Threat: User file list disclosure

ID	2
Description	Adversary get access to the liss of the files owned by an user, shared by an user or shared with an user.
STRIDE classification	Information disclosure
Known mitigation	See the corresponding threat tree.
Entry points	(1.5) User file list (2.1) Database connection
Assets	(1.2) User file list
Threat tree	See figure 3

Table 9: Threat: Access to the pages of an user

ID	3
Description	Adversary get access to the pages of an user without having valid credentials, for example by hijacking the session of the user. This can allow the adversary to see information about the user.
STRIDE classification	Elevation of privileges, Information disclosure
Known mitigation	Each page that requires a valid user checks that the user saved in the server-side session is logged in and valid. If this is not the case, a link to the login page is displayed instead of the actual page. Moreover, user sessions cannot be hijacked because all the communications between the user and the web server are made over a secure channel, thus the adversary cannot see the session's ID.
Entry points	(1.5) User file list page (1.6) User file download (1.7) User share page (1.8) User file deletion page (1.9) User certificate revocation page (1.10) User key revocation page (1.11) User file upload page (1.12) User password modification page
Assets	(1.2) User file list (1.3) User files (1.4) User file encryption key
Threat tree	None

Table 10: Threat: Modification of user data

ID	4
Description	Adversary can force user to modify its data without the user knowing it, for example with cross-site request forgery (CSRF).
STRIDE classification	Repudiation, Elevation of privileges
Known mitigation	Each page containing a form that allows the user to modify some data is protected using a unique CSRF token, which is then verified on the validation of the form. The attacker is thus not able to forge request because he does not know this unique token. Also, some modification operation require some operation by the user (more than just loading the page), such as decrypt and reencrypt a key for sharing a file, for example.
Entry points	(1.2) User validation page (1.7) User share page (1.8) User file deletion page (1.9) User certificate revocation page (1.10) User key revocation page (1.11) User file upload page (1.12) User password modification page
Assets	(1.1) User login credentials (1.3) User files (1.4) User file encryption key
Threat tree	None

Table 11: Threat: SQL injection in user input

ID	5
Description	Adversary tries to inject SQL commands through user input forms.
STRIDE classification	Tampering, Elevation of privileges
Known mitigation	The SQL queries are made with PHP's PDO extension, which, through the use of *prepared statements*, allow to build SQL queries by automatically escaping all user-supplied data, and thus prevents SQL injections. Moreover, all user data is first validated (which also prevent cross-site scripting (XSS)), and only text formed of simple character sets is allowed (alphanumeric characters, dots, dashes, spaces)
Entry points	(1.1) Admin login page (1.3) User login page (1.4) Account creation page (1.7) User share page (1.10) User file upload page
Assets	(4.1) Access to the database server
Threat tree	None

Table 12: Threat: Direct access to the database

ID	6
Description	Adversary get a valid connection to the database server, with a valid MySQL account. He can then read or alter some data in the database, depending on the account used.
STRIDE classification	Tampering, Information disclosure, Elevation of privileges
Known mitigation	The MySQL server is only accessible on the local host, so the adversary should previously be connected to the server. Moreover, database user credentials should be strong and not written anywhere. Also, multiple database accounts exists depending on the access needed (the `web` and `admin` user, who are described in the trust levels table).
Entry points	(2.1) Database connection
Assets	(4.1) Access to the database server
Threat tree	None

Table 13: Threat: Full access to the server

ID	7
Description	Adversary get a valid connection to the server with the root user account. He can then access the whole database with root privileges, all user files, keys and signatures which are stored on the filesystem, and stop the server processes or modify their configuration.
STRIDE classification	Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privileges
Known mitigation	There is no SSH server listening on the server. The server is installed on a FreeBSD jail, and the only way to access it is through the FreeBSD host. This FreeBSD host only have a SSH server and the minimal software needed to manage the jails. The SSH server do not accept connection with paswords, only with keys. The attacker should then have a valid SSH key to connect as user and know the root password of the server, which is very strong.
Entry points	(3.1) Server SSH connection
Assets	(3.1) Availability of the website (3.2) Ability to execute code as a web user (3.3) Ability to execute SQL code on the databases (3.4) Access to the file system
Threat tree	None

Table 14: Threat: Creation of an admin account

ID	8
Description	Adversary manage to create an admin account. He can then validates any user.
STRIDE classification	Elevation of privileges
Known mitigation	The only way to create an admin account is to have access to the database with the 'admin' account. The mitigation for this were explained in the threat 6.
Entry points	(2.1) Database connection
Assets	(4.4) Availability to create new admins
Threat tree	None

Table 15: Threat: Access to an user unencrypted file

ID	9
Description	Adversary manage gain access to an user encrypted file and to decrypt it.
STRIDE classification	Information disclosure
Known mitigation	If the adversary manage to download the file and its encrypted key (for example using threat 3), he still has to decrypt the key using the private key of the user. The private key of the user is only stored on the user's computer, and the user should thus make sure its computer is not accessible by malicious users.
Entry points	(1.6) User file download page
Assets	(1.3) User files
Threat tree	None