My CoRe

Audit de code ownCloud

Référence : Audit_Code_MyCoRe

Date de dernière mise à jour : 03/12/2013

Version du document : 1.0

Etat : validé

Auteurs : équipe projet My CoRe / prestataire DSI

Objet du document :
Un audit de code du logiciel ownCloud (partie serveur) dans le cadre du projet My CoRe. Ce document constitue le rapport technique de cet audit.

Table des mises à jour du document

Version du document	Date	Objet de la mise à jour
1.0	03/12/2013	Création du document

Sommaire

1 RESUME

Le périmètre de cet audit de code est le suivant :

- Application OwnCloud dans sa version 5.0.12 (code PHP uniquement)
- Contenu additionnel « APPS » : USER_SERVERVARS
- Contenu additionnel « APPS » : SHIBBOLETH
- Contenu additionnel « APPS » : Checksum
- Contenu additionnel « APPS » : Antivirus
- Contenu additionnel « APPS » : File_encryption

Ce qui représente une volumétrie, pour cet audit de code, de plus de un million de lignes de codes.

L'application OwnCloud présente un niveau de sécurité globalement satisfaisant. Plusieurs mécanismes de sécurité sont « natifs dans l'application. Les contenus additionnels APPS » ne présentent pas de vulnérabilités apparentes.
La majorité des vulnérabilités découvertes sont liés aux fonctions d'installation et de configuration de l'application. De ce fait, la suppression de ces éléments sur la plateforme de production couvre ces risques.
Les points remarquables suivants ont été identifiés :

- L'application est développée sous la « norme » PEAR Coding Standards.
- Les entêtes de fichiers et de fonctions sont renseignés.
- Les contrôles de sécurité « natifs » sont activés par défaut.
- L'APPS « SHIBBOLETH » ne présente pas de vulnérabilité.
- Le mécanisme d'authentification offerte par l'APPS « SHIBBOLETH » est suffisant d'un point de sécurité.
- L'APPS « Checksum » ne présente pas de vulnérabilité.
- L'APPS « File_Encryption » ne présente pas de vulnérabilité.
- L'APPS « Antivirus » ne présente pas de vulnérabilité.
- Les algorithmes de chiffrement utilisés sont issus d'une librairie considérée comme sûr.

Toutefois, les vulnérabilités suivantes ont été découvertes :

- Faiblesse dans la gestion des contextes.
- Ressources externes codés en dur.
- Présence de codes sources non nécessaires ou non utilisés (tests, débogage).
- Flux sensibles non chiffrés.
- Paramètres de sécurité des cookies trop permissifs.
- Des pages de tests sont vulnérables à de l'injection de commandes systèmes.
- Faiblesse dans le contrôle des données en entrée permettant une attaque de type XSS.
- Faiblesse dans la manipulation des chemins.

- Faiblesses dans le contrôle des données utilisées dans une requête SQL.
- Utilisation de générateur d'aléa considéré comme non sûr.
- Présence de fonction de traces trop verbeuses.

2 INTRODUCTION

2.1 Contexte et objectifs

Le CNRS a fait réaliser un audit de sécurité de l'application ownCloud par une société spécialisée dans le domaine, dans le cadre du projet de déploiement de cette solution.

2.2 Organisation du document

Les principaux résultats de cet audit sont résumés au § 1, une synthèse des risques au § 3, les détails des tests étant fournis au § 4 et suivants. Une synthèse des recommandations, des vulnérabilités et des points remarquables est proposée au § 6.
Chaque **vulnérabilité** découverte est associée à une **recommandation** décrite avec sa priorité, le coût estimé et la difficulté technique de sa mise en œuvre, sur le modèle suivant :

Vulnérabilité			
Vx: libellé de la vulnérabilité.			
Recommandation	**Priorité**	**Coût**	**Difficulté**
Rx: libellé de la recommandation.	Forte	*	*

Le coût **financier** et la **difficulté technique** inhérents à la mise en œuvre des recommandations sont estimés sur une échelle croissante allant de 0 à 3 étoiles (- / * / ** / ***), une étoile représente une cotation de chacun de ces facteurs.
La mesure de la priorité résulte d'une appréciation globale tenant compte de la probabilité de l'exploitation de la vulnérabilité sous-jacente par un attaquant (difficulté technique d'exploitation, disponibilité d'outils publics, intérêt de la cible, etc.) et de son impact (compromission de la machine, déni de service, etc.).
Dans le cas où les tests réalisés n'ont pas mis en avant de faiblesses ou bien ont démontré un comportement conforme au niveau de sécurité attendu, un point remarquable est rédigé dans le rapport sous la forme suivante :

Point remarquable
Px: libellé du point remarquable.

Le tableau suivant liste les niveaux de priorité associés aux recommandations et donne leur correspondance avec la criticité indiquée dans les scénarios de risque :

Priorité Recommandation	Criticité Risque	Description
Maximale	***	Vulnérabilité relativement facile à exploiter ayant un impact fort (intrusion, détournement d'un service etc.).
Forte	**	Vulnérabilité plus difficile à exploiter ou ayant un impact limité (déni de service, fuite d'informations etc.).
Moyenne	*	Vulnérabilité ne pouvant seule mener à une compromission mais pouvant être utilisée dans un but malveillant, souvent en combinaison avec d'autres failles.
Faible	Pas de scénario de risque	Vulnérabilité ne pouvant pas être exploitée à ce jour, ou recommandation non associée directement à une vulnérabilité.
Optimisation	N/A	Le comportement du service est expliqué. Aucun problème de sécurité n'est relevé. Cependant, la recommandation peut améliorer la sécurité de la cible.

Tableau 1 – Correspondance priorité de la recommandation / criticité du risque

3 SYNTHESE DES RISQUES

Les principaux risques techniques identifiés au cours de l'audit sont présentés selon le modèle suivant :

Risque N°	Cible Description de la vulnérabilité ou faille de sécurité.
Type de vulnérabilité	Conception, Configuration, Exploitation, Logicielle (et/ou)
Scénario d'attaque	Catégorie : locale ou distante, interne ou externe. Scénario diffusé (préciser si outil disponible) ou non. Exemple de scénario d'attaque.
Complexité de l'attaque	Faible / Moyenne / Élevée
Type d'impact	Atteinte à la disponibilité, intégrité ou confidentialité d'une ressource. Sur ressources réseaux/systèmes (intrusion, compromission, saturation) et sur données (divulgation, altération, destruction).
Criticité	* à ***
Vulnérabilité(s) associée(s)	Vulnérabilité(s) exploitée(s) pour réaliser le scénario de risque (liens vers les vulnérabilités détaillées dans le rapport).
Recommandation(s) associée(s)	Recommandation(s) visant à réduire/supprimer la vulnérabilité (liens vers les recommandations détaillées dans le rapport).

Tableau 2 – Modèle de fiche de risque

Les risques présentés sont associés aux vulnérabilités découvertes et aux recommandations correspondantes.
Les vulnérabilités sont classées par **type** : de **conception** (architecture matérielle ou logicielle), de **configuration** (matériel, système ou application), d'**exploitation** (procédures, utilisation des ressources, surveillance) ou **logicielle** (vulnérabilité connue dans la cible, pour laquelle un correctif est disponible mais n'est pas déployé).

Lorsqu'un scenario d'attaque est envisagé (exploitation d'une vulnérabilité), la **catégorie** de l'attaque indique les conditions initiales nécessaires à sa réalisation :

- **Locale :** pour l'exploitation de la vulnérabilité, l'utilisateur malveillant doit être connecté avec une session interactive sur la cible (par exemple, en ayant déjà compromis la cible avec une autre attaque réussie).

- **Distante :** attaque qui ne nécessite pas de session interactive et dont la vulnérabilité peut être exploitée à distance depuis le réseau. Nous indiquons si l'attaque est possible depuis un réseau **externe** au système d'information de la cible (par ex. Internet), ou si elle nécessite d'être menée depuis le réseau **interne** du système d'information de la cible (par ex. réseau local).

La **complexité** d'exploitation d'une vulnérabilité est définie sur une échelle de 3 niveaux (faible, moyen, élevé). Le premier niveau (faible) correspond à une attaque connue et simple à réaliser (outil ou technique publiquement disponible) et le dernier niveau (élevée) correspond à une attaque complexe qui nécessite des compétences et des moyens conséquents et pour laquelle aucune technique de mise en œuvre n'a été rendue publique.
La **mesure de la criticité** d'un risque tient compte de 2 facteurs : **la potentialité** ou facilité d'exploitation de la vulnérabilité et **l'impact** sur les ressources du système d'information si l'attaque réussit. Sur l'échelle de criticité définie, le premier niveau (*) correspond à une vulnérabilité difficile à exploiter ou ayant un impact faible sur le système d'information, le dernier niveau (***) correspond à une vulnérabilité facilement exploitable et ayant un impact fort sur une ressource du système d'information (intrusion sur le réseau privé, divulgation d'informations sensibles, compromission d'un dispositif de sécurité, etc.).
Les scénarios de risques sont ordonnés par **type de vulnérabilités** (conception, configuration, exploitation, logicielle), puis par **criticité**.

3.1 Risques de conception

Aucune vulnérabilité importante de ce type n'a été relevée.

3.2 Risques de configuration

Aucune vulnérabilité importante de ce type n'a été relevée.

3.3 Risques d'exploitation

Aucune vulnérabilité importante de ce type n'a été relevée.

3.4 Risques logiciels

Aucune vulnérabilité importante de ce type n'a été relevée.

4 ANALYSE DE LA QUALITE DU CODE

4.1 Générale

4.1.1 Utilisation d'une norme de développement

Observation
L'application est développée sous la « norme » PEAR Coding Standards. L'utilisation d'une norme de développement permet d'avoir une qualité de code homogène. La norme PEAR Coding Standards traite des aspects suivants :

- Contrôle de structures

- Appel de fonctions

- Définition de classes

- Définition de fonctions
- Pointeurs
- Commentaires
- Utilisation du SVN
- Convention de nommage
- Format de fichiers

Point remarquable
P1 : L'application est développée sous la « norme » PEAR Coding Standards.

4.1.2 Utilisation de PHPDoc

Observation

La documentation indique que toutes les API doivent être identifiées avec PHPDoc. Les informations contenues dans les entêtes de classe et de fonctions permettent d'identifier rapidement le contenu de ces dernières.

```
/**
* Command line tool to work with database schemas
*
* Functionality:
* - dump a database schema to stdout
* - import schema into database
* - create a diff between two schemas
* - apply diff to database
*
* @category Database
* @package MDB2_Schema
* @author Christian Weiske <cweiske@php.net>
* @license BSD http://www.opensource.org/licenses/bsd-license.php
* @link http://pear.php.net/packages/MDB2_Schema
*/
class MDB2_Schema_Tool
{
/**
* Run the schema tool
*
* @param array $args Array of command line arguments
       */
```

Capture 1 – Fichier 3rdparty\MDB2\Schema\Tool.php

Point remarquable
P2 : Les entêtes de fichiers et de fonctions sont renseignés.

4.1.3 Ressources non fermés

Observation

Certaines ressources sont ouvertes en écriture / lecture à chaque appel de la fonction mais elles ne sont jamais fermés.

```
function _readConfigDataFrom($file)
{
$fp = false;
if (file_exists($file)) {
$fp = @fopen($file, "r");
```

Capture 2 – Fichier 3rdparty/PEAR/Config.php

Risque
Une accumulation d'ouvertures et d'écritures dans ces ressources sans qu'elles ne soient jamais fermés pourrait provoquer à terme un dysfonctionnement de l'application.

Recommandation
Il est souhaitable d'identifier dans tout le code le traitement des exceptions afin que :

- les contextes soient restaurés à l'état d'origine ;
- les buffers soient libérés ;
- les transactions soient annulées ;
- les fichiers et sockets soient fermés ;

Vulnérabilité			
V10 : Utilisation de générateur d'aléa considéré comme non sûr.			
Recommandation	**Priorité**	**Coût**	**Difficulté**
R10 : Utiliser les fonctions de chiffrements considérées comme sûrs.	Forte	**	**

4.1.4 Ressources externes codés en dur

Observation
Des ressources externes sont codées en dur dans le code source de l'application.

```
<!-- Le HTML5 shim, for IE6-8 support of HTML5 elements -->
<!--[if lt IE 9]>
<script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
```

Capture 3 – Fichier 3rdparty/fontawesome/docs/assets/less/twbs-222/tests/buttons.html

```
<!--[if lt IE 9]>
<script src="http://html5shim.googlecode.com/svn/trunk/html5.js"></script>
<![endif]-->
3rdparty/fontawesome/docs/index.html
$plaintext = str_pad($this->toBytes(), strlen($n->toBytes(true)) - 1, "\0", STR_PAD_LEFT);
if (openssl_public_encrypt($plaintext, $result, $RSAPublicKey, OPENSSL_NO_PADDING)) {
return new Math_BigInteger($result, 256);
}
                                                    }
```

Capture 4 – Fichier apps/files_external/3rdparty/phpseclib/phpseclib/Math/BigInteger.php

Risque
En cas de modification des ressources externes (exemple : changement d'url) cela nécessitera une revue des constantes de l'application. Dans le cas où une donnée de configuration est incorrecte (erreur, modification malveillante), le problème sera difficile à identifier et long à corriger (création d'un nouveau package de l'application).

Recommandation
Il est préférable de stocker les informations relatives à des liens externes dans des fichiers de configuration.

Vulnérabilité			
V2 : Ressources externes codés en dur.			
Recommandation	**Priorité**	**Coût**	**Difficulté**
R2 : Utiliser un fichier de configuration dédié pour stocker les données de configuration.	Faible	*	*

4.1.5 Code en cours de développement

Observation

Des fichiers ont été identifiés comme non nécessaires pour un usage en production. Certains éléments servent à des fins de test et de débogage (identifiés à l'aide du tag « FIXME »). L'exemple ci-dessous montre des extraits de code non nécessaires pour la mise en production de l'application

```
Search "fixme" (41 hits in 28 files)
3rdparty\MDB2\Schema\Tool.php (1 hit)
Line 322: //FIXME: exception class
3rdparty\MDB2\Schema\Validate.php (1 hit)
Line 982: // FIXME: find a way to issue a warning in MDB2_Schema object
3rdparty\MDB2\Schema.php (1 hit)
Line 2036: /* FIXME: tables marked to be added are initialized by createTable(), others don't */
3rdparty\PEAR\Builder.php (1 hit)
Line 343: // FIXME make configurable
3rdparty\PEAR\Dependency.php (3 hits)
Line 292: // XXX Fixme: Implement a more flexible way, like
Line 355: // XXX FIXME honor safe mode
Line 383: // XXX Fixme: There is no way to know if the user has or
3rdparty\PEAR\Downloader\Package.php (2 hits)
Line 1379: // FIXME do symlink check
Line 1625: ///FIXME need to pass back some error code that we can use to match with to cancel all further operations
3rdparty\PEAR\RunTest.php (1 hit)
Line 197: // FIXME review if this is really the struct to go with
apps\calendar\lib\app.php (1 hit)
Line 44: // FIXME: Correct arguments to just check for permissions
apps\contacts\js\app.js (4 hits)
Line 511: // FIXME: I suck at regexp. /Tanghus
Line 741: // FIXME: Settings needs to be refactored
Line 803: // FIXME: Refactor this to be usable for favoriting also.
Line 1217: // FIXME: This should only apply for contacts list.
apps\contacts\js\contacts.js (3 hits)
Line 262: * new datastructure will be added to the object. FIXME: Not implemented yet.
Line 363: action: ((obj && obj.defaultValue) || self.data[element].length) ? 'save' : 'add', // FIXME
Line 1670: * FIXME: If continious loading is reintroduced this will have
apps\contacts\js\groups.js (1 hit)
Line 424: * FIXME: This works fine for adding, but will need refactoring
apps\contacts\js\modernizr.js (1 hit)
Line 634: // legacy webkit syntax (FIXME: remove when syntax not in use anymore)
apps\files_encryption\lib\proxy.php (1 hit)
Line 273: // FIXME: handling for /userId/cache used by webdav for chunking. The cache chunks are NOT encrypted
apps\files_encryption\tests\proxy.php (1 hit)
Line 55: // $this->session = new Encryption\Session( $view ); // FIXME: Provide a $view object for use here
apps\files_external\3rdparty\google-api-php-client\src\io\Google_REST.php (1 hit)
Line 118: //FIXME work around for the the uri template lib which url encodes
apps\files_pdfviewer\3rdparty\pdfjs\pdf.js (2 hits)
Line 36955: function fixMetadata(meta) {
Line 36975: meta = fixMetadata(meta);
apps\files_texteditor\js\vendor\ace\src-noconflict\worker-javascript.js (1 hit)
                                    Line 7549: E031: "Bad assignment.", // FIXME: Rephrase
```

```
apps\search_lucene\lib\indexer.php (1 hit)
Line 62: // FIXME uses ZipArchive ... make compatible with OC\Files\Filesystem
apps\search_lucene\lib\lucene.php (1 hit)
Line 195: //FIXME emulates the old search but breaks all the nice lucene search query options
apps\user_ldap\lib\helper.php (1 hit)
Line 59: //FIXME oracle hack: need to explicitly cast CLOB to CHAR for comparison
lib\app.php (1 hit)
Line 178: //FIXME oracle hack: need to explicitly cast CLOB to CHAR for comparison
lib\connector\sabre\locks.php (3 hits)
Line 50: //FIXME oracle hack: need to explicitly cast CLOB to CHAR for comparison
Line 68: //FIXME oracle hack: need to explicitly cast CLOB to CHAR for comparison
Line 80: //FIXME oracle hack: need to explicitly cast CLOB to CHAR for comparison
lib\db.php (3 hits)
Line 341: //FIXME: check limit notation for other dbs
Line 531: /* FIXME: use CURRENT_TIMESTAMP for all databases. mysql supports it as a default for
DATETIME since 5.6.5 [1]
Line 633: /* FIXME: use CURRENT_TIMESTAMP for all databases. mysql supports it as a default for
DATETIME since 5.6.5 [1]
lib\files\cache\cache.php (1 hit)
Line 128: //FIXME hide this HACK in the next database layer, or just use doctrine and get rid of
MDB2 and PDO
lib\files\mount.php (1 hit)
Line 93: $storage = $this->createStorage(); //FIXME: start using exceptions
lib\helper.php (1 hit)
Line 449: //FIXME: should also check for value validation (i.e. the email is an email).
lib\setup.php (1 hit)
Line 515: //FIXME check tablespace exists: select * from user_tablespaces
lib\user.php (1 hit)
    Line 613: if (OC_Config::getValue( 'dbtype', 'sqlite' ) === 'oci') { //FIXME oracle hack
```

Risque

La présence de codes de tests et de débogage non identifiés peut être une source d'erreur pour le développement de l'application et de son évolution. Les conséquences peuvent être des dysfonctionnements difficiles à déceler dans l'application.

De plus, la présence de codes non utilisés peut entraîner un risque de sécurité supplémentaire en augmentant la surface d'attaques (ex : fonctions non utilisées mais activées dans l'application).

Recommandation

Organiser une arborescence « propre » pour compiler le package destiné à la plateforme de production : suppression des fichiers non nécessaires, des outils tiers pour le développement et la compilation du code, des fichiers de tests et de débogage, etc.

Vulnérabilité			
V3 : Présence de codes sources non nécessaires ou non utilisés (tests, débogage).			
Recommandation	**Priorité**	**Coût**	**Difficulté**
R3 : Organiser une arborescence « propre » pour la compilation du package à mettre en production comprenant uniquement le code utile.	Faible	*	*

5 ANALYSE DES MECANISMES DE SECURITE

5.1 Controles de sécurité « natifs »

Observation

Des contrôles de sécurités sont activés par défaut sur chaque page. Il est possible de les désactiver en ajoutant un tag dans l'entête du fichier. Pour information, aucun tag n'a été identifié dans les sources, il en résulte que l'ensemble des mécanismes de sécurités sont activés. Ces mécanismes de sécurité sont les suivants :

- @CSRFExemption: Désactive la vérification du token CSRF
- @IsAdminExemption: Désactive la vérification si l'utilisateur est un administrateur
- @IsLoggedInExemption: Désactive la vérification si l'utilisateur est authentifié
- @IsSubAdminExemption: Désactive la vérification si l'utilisateur est un « subadmin »

```
<?php
/**
* @CSRFExemption
* @IsAdminExemption
* @IsLoggedInExemption
* @IsSubAdminExemption
*/
public function index(){
$templateName = 'main';
$params = array(
'somesetting' => 'How long will it take'
);
                        return $this->render($templateName, $params); }
```

Capture 5 – Liste de fichiers en cours de développement

Point remarquable
P3 : Les contrôles de sécurité « natifs » sont activés par défaut.

5.2 APPS

5.2.1 SHIBBOLETH

Observation

L'APPS « Shibboleth » est une interface qui gère l'authentification des utilisateurs. Cette authentification peut être réalisée via un annuaire LDAP ou bien par un mécanisme d'authentification par login / mot de passe. L'extrait de code suivant illustre comment est fait ce choix au niveau de l'application.

```
public function checkPassword($uid, $password) {
if (Auth::getShibIdentityProvider() && $password === 'irrelevant') {
//distinguish between internal and external Shibboleth users
//internal users log in with their LDAP (entry)uuid,
$adapter = new LdapBackendAdapter();
$mail = Auth::getMail();
if (($mail !== false) && ($adapter->getUuid($mail) === $uid)) {
return $uid;
}
//external users log in with their hashed persistentID
$persistentId = Auth::getPersistentId();
$loginName = LoginLib::persistentId2LoginName($persistentId);
if ($loginName === $uid) {
$this->updateQuota($uid);
return $uid;
                                                }
```

Capture 7 – owncloud\user_shibboleth\user_shibboleth.php

La fonction suivante illustre le mécanisme qui vérifie comment est effectuée l'authentification en récupérant l'adresse mail de l'utilisateur.

```
//distinguish between internal (those in the LDAP) and external Shibboleth users
$adapter = new \OCA\user_shibboleth\LdapBackendAdapter();
$loginName = $adapter->getUuid($mail);
if ($loginName) {//user is internal, backends are enabled, and user mapping is active
$adapter->initializeUser($loginName);
} else {//user is external
//crop $mail to fit into display_name column of oc_shibboleth_user
if (strlen($mail) > 64) {
$mail = substr($mail, 0, 64);
}
//make sure that user entry exists in oc_shibboleth_user
$loginName = \OCA\user_shibboleth\LoginLib::persistentId2LoginName($persistentId);
$displayName = $mail;
if (\OCA\user_shibboleth\DB::loginNameExists($loginName)) {
//update display name if it has changed since last login
if ($displayName !== \OCA\user_shibboleth\DB::getDisplayName($loginName)) {
\OCA\user_shibboleth\DB::updateDisplayName($loginName, $displayName);
                                                    }
```

Capture 8 – owncloud\user_shibboleth\login.php

```
public static function endsWith($string, $suffix, $caseInsensitive = true) {
$stringLength = strlen($string);
$suffixLength = strlen($suffix);
if ($suffixLength < $stringLength) {
$comp = substr_compare($string, $suffix, $stringLength - $suffixLength, $suffixLength,
$caseInsensitive);
if ($comp === 0)
return true;
}
return false;
                                                    }
```

Capture 9 – owncloud\user_shibboleth\lib\login_lib.php

L'analyse de cette APPS ne nous a pas permis d'identifier de vulnérabilité. Le fonctionnement semble conforme par rapport à la description du produit. De plus, cette APPS semble offrir un contenu plus étoffé que l'APPS « USER_SERVERVARS ». Le mécanisme d'authentification offerte par l'APPS « SHIBBOLETH » est d'un niveau de sécurité supérieur par rapport au mécanisme d'authentification natif offert par l'application.

Point remarquable
P4 : L'APPS « SHIBBOLETH » ne présente pas de vulnérabilité.

Point remarquable
P5 : Le mécanisme d'authentification offerte par l'APPS « SHIBBOLETH » est suffisant d'un point de vue sécurité

5.2.2 Checksum

Observation

Il s'agit d'un plugin qui permet d'obtenir le md5 d'un fichier en cliquant sur une icône. Aucun impact de sécurité n'a été identifié sur cette APPS.

5.2.3 File_Encryption

Observation

Cette APPS est chargée de chiffrer chaque document déposé sur le serveur. Chaque fichier est découpé en deux composants : les données chiffrées (catfile) et un keyfile. Il y a également en option une clé de partage (share key) qui permet de partager le document avec un autre utilisateur.

En cas de perte de mot de passe d'un utilisateur, les administrateurs peuvent utiliser le compte « recoveryadmin » pour générer un nouveau keyfile afin que l'utilisateur ne soit pas bloqué et puisse récupérer ses documents.

La paire de clés (publique et privée) est générée à partir de la librairie OpenSSL avec une longueur de 4096 bits comme illustré dans l'exemple ci-dessous. Une empreinte des clés est effectué avec du SHA512.

```
/**
* @brief Create a new encryption keypair
* @return array publicKey, privatekey
*/
public static function createKeypair() {
$return = false;
$res = openssl_pkey_new(array('private_key_bits' => 4096));
if ($res === false) {
\OCP\Util::writeLog('Encryption library', 'couldn\'t generate users key-pair for ' .
\OCP\User::getUser(), \OCP\Util::ERROR);
while ($msg = openssl_error_string()) {
\OCP\Util::writeLog('Encryption library', 'openssl_pkey_new() fails: ' . $msg,
\OCP\Util::ERROR);
}
} elseif (openssl_pkey_export($res, $privateKey)) {
// Get public key
$keyDetails = openssl_pkey_get_details($res);
$publicKey = $keyDetails['key'];
$return = array(
'publicKey' => $publicKey,
'privateKey' => $privateKey
                                                );
```

Capture 10 – Fichier apps\files_encryption\lib\crypt.php

De plus, cette APPS utilise un mécanisme de sécurité qui permet de filtrer les requêtes de type « Directory Traversal ». Cette fonction de validation des données pour les chemins relatifs est détaillée ci-dessous.

```
/**
* @brief glob uses different pattern than regular expressions, escape glob pattern only
* @param unescaped path
* @return escaped path
*/
public static function escapeGlobPattern($path) {
                        return preg_replace('/(\*|\?|\[)/', '[$1]', $path);
```

Capture 11 – Fichier apps\files_encryption\lib\helper.php

Point remarquable
P7 : L'APPS « File_Encryption » ne présente pas de vulnérabilité.

5.2.4 Antivirus

Observation

L'APPS « Antivirus » permet d'ajouter un composant de contrôle antivirus sur la plateforme. Le logiciel utilisé est l'antivirus ClamAV. Il fonctionne à la fois en mode bouclier et en scan planifié.
De ce fait, chaque document qui est uploadé sur le serveur est contrôlé par l'antivirus. Il est également possible d'ajouter un contrôle antivirus sur l'ensemble des fichiers sous la forme d'une tâche planifiée. En cas de détection de code malveillant, une information est transmise à l'utilisateur.

Point remarquable
P8 : L'APPS « Antivirus » ne présente pas de vulnérabilité.

5.3 Authentification

Observation

L'utilisation du protocole HTTPS n'est pas activée par défaut sur l'application.

Risque

L'échange de données sans aucune protection facilite fortement l'interception de celles-ci par un attaquant. En effet, il est assez simple de d'intercepter un flux réseau, notamment sur un réseau interne, ou bien de réaliser une attaque de type "Man in The Middle" sans être détecté.

Recommandation

Certaines données peuvent être publiques et donc n'avoir qu'un besoin de sécurité faible, mais il est nécessaire d'assurer la protection des données sensibles (informations confidentielles ou nominatives, accès d'administration à des systèmes, etc.), à l'aide de protocoles sécurisés, tels que SSL, afin de permettre une confidentialité et intégrité des flux, ainsi qu'une authentification forte des systèmes communiquant.

Vulnérabilité			
V4 : Flux sensibles non chiffrés.			
Recommandation	**Priorité**	**Coût**	**Difficulté**
R4 : Mettre en place un chiffrement des flux sur l'application.comprenant uniquement le code utile.	Maximale	*	*

5.3.1 Sécurité des cookies

Observation

Les paramètres de sécurité des cookies ne sont pas assez restreints. En effet, aucun paramètre de sécurité n'est activé pour les cookies.

```
/**
* @brief Set cookie value to use in next page load
* @param string $username username to be set
*/
public static function setMagicInCookie($username, $token) {
$secure_cookie = OC_Config::getValue("forcessl", false);
$expires = time() + OC_Config::getValue('remember_login_cookie_lifetime', 60*60*24*15);
setcookie("oc_username", $username, $expires, '', '', $secure_cookie);
setcookie("oc_token", $token, $expires, '', '', $secure_cookie, true);
setcookie("oc_remember_login", true, $expires, '', '', $secure_cookie);
}
/**
* @brief Remove cookie for "remember username"
*/
public static function unsetMagicInCookie() {
unset($_COOKIE["oc_username"]);
unset($_COOKIE["oc_token"]);
unset($_COOKIE["oc_remember_login"]);
setcookie("oc_username", null, -1);
setcookie("oc_token", null, -1);
setcookie("oc_remember_login", null, -1);
                                                }
```

Capture 12 – Fichier lib/user.php

Risque

Les cookies sont la plupart du temps utilisés comme jeton de session. Un attaquant peut donc mettre en oeuvre des moyens pour récupérer un cookie afin de pouvoir usurper l'identité de sa victime en le réutilisant. Dans le cas où les différents paramètres de sécurité des cookies ne sont pas mis en place, la récupération d'un cookie peut être facilitée, notamment si les paramètres "secure" et "httpOnly" ne sont pas utilisés. Cependant, l'absence de paramètres de sécurité sur les cookies est généralement utilisée par les attaquants conjointement avec une autre vulnérabilité pour récupérer les cookies.

Recommandation

Afin de protéger au mieux les cookies utilisés par l'application, il est recommandé d'utiliser les paramètres de sécurité suivants, de la façon la plus restrictive :

- Secure : le cookie sera seulement transmis en HTTPS
- Path : permet de définir une sous-arborescence dans laquelle le cookie est valide
- Domain : permet de définir un domaine plus strict dans lequel le cookie est valide
- Expires : permet de définir quand le cookie expire

Vulnérabilité			
V5 : Paramètres de sécurité des cookies trop permissifs.			
Recommandation	**Priorité**	**Coût**	**Difficulté**
R4 : Mettre en place un chiffrement des flux sur l'application.comprenant uniquement le cR5 : Utiliser les paramètres de sécurité des cookies (secure, httpOnly, etc.).ode utile.	Moyenne	*	**

5.4 Validation des données

5.4.1 Injection de commandes systèmes

Observation

Des fonctions exécutent des commandes systèmes sans aucun contrôle des paramètres en entrée. Les vulnérabilités de type injection de commande n'ont été identifiées que sur des pages qui semblent être des pages de test L'exemple suivant illustre que le paramètre « commandline » ne passe par aucun contrôle de sécurité.

```
function system_with_timeout($commandline, $env = null, $stdin = null)
{
$data = '';
if (version_compare(phpversion(), '5.0.0', '<')) {
                              $proc = proc_open($commandline, array(
```

Capture 13 – Fichier 3rdparty/PEAR/RunTest.php

L'exemple suivant illustre que le paramètre « cmd » ne passe par aucun contrôle de sécurité.

```
if (OS_WINDOWS && isset($section_text['RETURNS'])) {
ob_start();
system($cmd, $return_value);
$out = ob_get_contents();
                                        ob_end_clean();
```

Capture 14 – Fichier 3rdparty/PEAR/RunTest.php

Risque

Un attaquant ayant accès à ces pages vulnérables pourrait modifier les paramètres exécutés par le serveur. Ainsi, il sera capable d'exécuter des commandes système directement depuis son navigateur.

Recommandation

Les vulnérabilités de type injection de commande n'ont été identifiées que sur des pages qui semblent être des pages de test. Il est recommandé de supprimer ces pages de test sur l'environnement de production.

Vulnérabilité			
V6 : Des pages de tests sont vulnérables à de l'injection de commandes systèmes.			
Recommandation	**Priorité**	**Coût**	**Difficulté**
R6 : Supprimer les pages de tests vulnérables	Moyenne	*	*

5.4.2 XSS

Observation

Les données fournies par l'utilisateur ne sont pas suffisamment contrôlées sur certaines pages et permettent de saisir du code JavaScript sur cette dernière. Un exemple est illustré ci-dessous :

```
xpublic function send($type, $data=null) {
if(is_null($data)) {
$data=$type;
$type=null;
}
if($this->fallback) {
$response='<script type="text/javascript">window.parent.OC.EventSource.fallBackCallBack('
.$this->fallBackId.',"'.$type.'",'.json_encode($data).')</script>'.PHP_EOL;
                                        echo $response;
```

Capture 15 – Fichier lib/eventsource.php

Le guide du développeur d'OwnCloud recommande de ne pas utiliser certaines fonctions qui sont vecteur de vulnérabilités XSS. Le guide recommande d'utiliser une fonction dédié au filtrage de données : « print_unescaped($data) ». Un exemple est illustré ci-dessous :

```
<form class="addBm" method="post" action="<?php print_unescaped(OCP\Util::linkTo('bookmarks',
'ajax/editBookmark.php'));?>">
<?php if(!isset($embedded) || !$embedded):?>
   <script type="text/javascript" src="<?php print_unescaped(OC_Helper::linkTo('bookmarks/js',
                                'full_tags.php'));?>"></script>
```

Capture 16 – Fichier apps\bookmarks\templates\addBm.php

Cette fonction de sécurité n'a pas été identifiée sur les APPS.

Risque

Un attaquant peut insérer du code JavaScript (ou autre) afin qu'il soit exécuté dans la session de la victime (ex: envoi d'un lien à cliquer, stockage dans un champ affiché par la victime, ...).

Cependant la victime doit être authentifiée sur l'application afin que l'attaque réussisse, ce qui limite la probabilité d'exploitation de ce risque.

Par conséquent, un attaquant parvenant à exploiter une vulnérabilité de type XSS pourra contrôler la session de sa victime (autre utilisateur de l'application).

Recommandation

Il est recommandé de créer des routines, utilisées avant tout traitement de données utilisateurs, qui permettent de rendre "saine" toute information.

Ces contrôles doivent être réalisés sur toutes les données utilisées en entrée par l'application: données fournies par l'utilisateur mais aussi données provenant d'un autre logiciel ou système, tel qu'une base de données.

Bien que la création et la mise en place de ces routines puissent avoir un coût au départ, cet investissement est amorti sur le long terme (centralisation des contrôles, filtrage systématique des informations entrées, amélioration du niveau de sécurité global de l'application).

Vulnérabilité			
V7 : Faiblesse dans le contrôle des données en entrée permettant une attaque de type XSS.			
Recommandation	**Priorité**	**Coût**	**Difficulté**
R7 : Filtrer les données en entrée de l'application pour éviter les attaques XSS.	Moyenne	**	**

5.4.3 Manipulation de chemin

Observation

Des ressources du serveur sont utilisées sans filtrage de données. Dans l'exemple suivant, la ressource « $file » est utilisée sans vérifier que le chemin de la ressource n'a pas été modifié

```
$fp = @fopen($file, "w");
if (!$fp) {
return $this->raiseError("PEAR_Config::writeConfigFile fopen('$file','w') failed
($php_errormsg)");
}
$contents = "#PEAR_Config 0.9\n" . serialize($data);
if (!@fwrite($fp, $contents)) {
return $this->raiseError("PEAR_Config::writeConfigFile: fwrite failed ($php_errormsg)");
}
                                        return true;
```

Capture 17 – Fichier 3rdparty/PEAR/Config.php

Risque

En cas de modification du chemin, un attaquant pourrait écrire dans un fichier du système d'exploitation ou bien modifier le fichier de configuration de l'application. De ce fait l'attaquant pourrait obtenir un accès sur la plateforme avec des privilèges plus élevés.

Recommandation

Il est recommandé de créer des routines, utilisées avant tout traitement de données utilisateurs, qui permettent de rendre "saine" toute information.

Ces contrôles doivent être réalisés sur toutes les données utilisées en entrée par l'application: que ce soit les données fournies par l'utilisateur ou les données provenant d'un autre composant, comme une base de données.

Bien que la création et la mise en place de ces routines puissent avoir un coût au départ, cet investissement est amorti sur le long terme (centralisation des contrôles, filtrage systématique des informations entrées, amélioration du niveau de sécurité global de l'application).

Vulnérabilité			
V8 : Faiblesse dans la manipulation des chemins			
Recommandation	**Priorité**	**Coût**	**Difficulté**
R8 : Mettre en place un filtrage sur l'ensemble de la manipulation des chemins. Mettre en place un chiffrement des flux sur l'application.comprenant uniquement le code utile.	Moyenne	*	*

5.4.4 Injection SQL

Observation

Des fonctions côté serveur ont été identifiées comme étant potentiellement vulnérables à une injection SQL. L'exemple suivant montre l'absence de filtrage des paramètres passés depuis la fonction appelante ainsi que la non utilisation de procédures stockées.

```
private static function createMySQLDatabase($name, $user, $connection) {
//we cant use OC_BD functions here because we need to connect as the administrative user.
$l = self::getTrans();
$query = "CREATE DATABASE IF NOT EXISTS `$name`";
$result = mysql_query($query, $connection);
if(!$result) {
$entry = $l->t('DB Error: "%s"', array(mysql_error($connection))) . '<br />';
$entry .= $l->t('Offending command was: "%s"', array($query)) . '<br />';
\OC_Log::write('setup.mssql', $entry, \OC_Log::WARN);
}
$query="GRANT ALL PRIVILEGES ON `$name` . * TO '$user'";
//this query will fail if there aren't the right permissions, ignore the error
mysql_query($query, $connection);
                                                };
```

Capture 18 – Fichier lib/setup.php

Note: Ce risque est limité car la vulnérabilité se situe dans un fichier d'installation de la plateforme. Cependant, le reste de l'application utilise pour ses requêtes SQL des procédures stockées. Cet exemple est illustré ci-dessous :

```
public static function getHomeDir($loginName) {
$query = \OCP\DB::prepare('SELECT home_dir FROM *PREFIX*shibboleth_user WHERE login_name = ?');
                        $result = $query->execute(array($loginName));
```

Capture 19 – Fichier owncloud\user_shibboleth\database\db.php

Risque

Un attaquant pourrait injecter du code SQL afin de modifier la requête SQL transmise à la base de données. Ainsi, en fonction des droits de l'utilisateur sur la base de données, l'attaquant pourrait accéder à tout ou une partie de la base de données, en lecture et/ou écriture.

Par conséquent, un attaquant parvenant à exploiter une vulnérabilité de type injection SQL contrôlera les informations stockées dans la base de données.

Recommandation

Il est recommandé de créer des routines, utilisées avant tout traitement de données utilisateurs, qui permettent de rendre "saine" toute information. En particulier, il convient de réaliser ces contrôles avant de transmettre les données à une base de données.

Vulnérabilité			
V9 : Faiblesses dans le contrôle des données utilisées dans une requête SQL.			
Recommandation	**Priorité**	**Coût**	**Difficulté**
R9 : Filtrer les données utilisées dans une requête SQL.	Moyenne	**	**

5.5 Chiffrement

5.5.1 Fonction de générateur d'aléa non sécurisé

Observation

Les générateurs d'aléa utilisés par l'application ne sont pas considéré comme robustes. Le guide du développeur indique que les fonctions « rand(), srand(), mt_rand() » doivent être remplacés par OC_Util::generate_random_bytes().

```
$random = '';
if ($size & 1) {
$random.= chr(mt_rand(0, 255));
                                                    }
```

Capture 20 – Fichier apps/files_external/3rdparty/phpseclib/phpseclib/Math/BigInteger.php

Il y a une fonction de génération de mot de passe aléatoire dans l'APPS « USER_SERVERVARS ». Le mot de passe est long, mais le générateur d'aléa utilisée n'est pas considéré comme sure.

```
function random_password() {
$valid_chars
= 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
$length = 20;
$ind_max_valid_chars = strlen($valid_chars) - 1;
$random_string = '';
for ($i = 0; $i < $length; $i++) {
$random_pick = mt_rand(0, $ind_max_valid_chars);
$random_char = $valid_chars[$random_pick];
$random_string .= $random_char;
                                                }
```

Capture 21 – Fichier owncloud\user_servervars\lib\hooks.php

Risque
L'utilisation de fonctions de chiffrements/de générateurs aléatoires considérés comme non sûrs rend l'utilisation de ces derniers caduque.
Recommandation
Il est recommandé d'utiliser les fonctions de cryptographie robustes, comme indiqué dans le guide des développeurs.

Vulnérabilité			
V10 : Utilisation de générateur d'aléa considéré comme non sûr.			
Recommandation	**Priorité**	**Coût**	**Difficulté**
R10 : Utiliser les fonctions de chiffrements considérées comme sûrs.	Forte	**	**

5.5.2 Algorithmes de chiffrement

Observation
Les algorithmes de chiffrement utilisés sont issus de la librairie « phpseclib ». Aucun algorithme de chiffrement « maison » n'a été identifié dans l'application. Les mécanismes de chiffrement ne présentent pas de vulnérabilité.

Point remarquable
P9 : Les algorithmes de chiffrement utilisés sont issus d'une librairie considérée comme sûr.

5.6 Traces

5.6.1 Fuite d'informations

Observation
Certaines fonctions ont été identifiées comme étant trop verbeuses, c'est-à-dire que l'ensemble de la pile d'information est affichée. Cette pile d'information pourrait contenir des éléments sensibles. Les exemples sont illustrés ci-dessous

```
function displayFatalError($eobj)
{
$this->displayError($eobj);
if (class_exists('PEAR_Config')) {
$config = &PEAR_Config::singleton();
if ($config->get('verbose') > 5) {
if (function_exists('debug_print_backtrace')) {
debug_print_backtrace();
                                        exit(1);
```

Capture 22 – Fichier 3rdparty/PEAR/Frontend/CLI.php

```
$search = 'zend_extensions';
ob_start();
                                    phpinfo(INFO_GENERAL);
```

Capture 23 – Fichier 3rdparty/PEAR/Command/Install.php

```
public function __get($name)
{
if (array_key_exists($name, $this->packlets))
return $this->packlets[$name];
else {
                                    debug_print_backtrace();
```

Capture 24 – Fichier apps/files_external/3rdparty/irodsphp/prods/src/packet/RODSPacket.class.php

Risque

Un développeur pourrait oublier de commenter des instructions de debug lors de la mise en production, entrainant un dysfonctionnement de la page générée ou affichant des informations confidentielles. De plus, ces commentaires ne facilitent pas la lecture du code.

Recommandation

Les bonnes pratiques de programmation préconisent de ne pas utiliser les commentaires pour les parties de debug du code avant la mise en production de l'application.

Il est souhaitable d'identifier clairement les fonctions de test et de debug dans le code (utilisation de mots-clés en commentaire, identification du rôle de la fonction, etc.). Afin de désactiver toutes les fonctions de debug pour la mise en production, il est nécessaire d'utiliser les instructions conditionnelles afin de supprimer les sections de code concernées à la compilation.

Vulnérabilité			
V11 : Présence de fonction de traces trop verbeuses			
Recommandation	**Priorité**	**Coût**	**Difficulté**
R11 : Supprimer les instructions de test et de débogage mises en commentaires ou utiliser à la place des instructions conditionnelles.	Faible	*	*

6 SYNTHESE

6.1 Recommandations (par priorité)

Recommandations	Priorité	Coût	Difficulté
R4 : Mettre en place un chiffrement des flux sur l'application.	Maximale	*	*
R10 : Utiliser les fonctions de chiffrements considérés comme sûrs.	Forte	**	**
R5 : Utiliser les paramètres de sécurité des cookies (secure, httpOnly, etc.).	Moyenne	*	**
R6 : Supprimer les pages de tests vulnérables	Moyenne	*	*
R7 : Filtrer les données en entrée de l'application pour éviter les attaques XSS.	Moyenne	**	**
R9 : Filtrer les données utilisées dans une requête	Moyenne	**	**
R8 : Mettre en place un filtrage sur l'ensemble de la manipulation des chemins.	Moyenne	*	*
R11 : Supprimer les instructions de test et de débogage mises en commentaires ou utiliser à la place des instructions conditionnelles.	Faible	*	*
R1 : Identifier dans tout le code le traitement des exceptions afin que : • les contextes soient restaurés à l'état d'origine ; • les buffers soient libérés ; • les transactions soient annulées ; • les fichiers et sockets soient fermés.	Faible	**	**
R2 : Utiliser un fichier de configuration dédié pour stocker les données de configuration.	Faible	*	*
R3 : Organiser une arborescence « propre » pour la compilation du package à mettre en production comprenant uniquement le code utile.	Faible	*	*

Tableau 3 – Synthèse des recommandations

6.2 Vulnérabilités

Vulnérabilités
V1 : Faiblesse dans la gestion des contextes.
V2 : Ressources externes codés en dur.
V3 : Présence de codes sources non nécessaires ou non utilisés (tests, débogage).

V4 : Flux sensibles non chiffrés.
V5 : Paramètres de sécurité des cookies trop permissifs.
V6 : Des pages de tests sont vulnérables à de l'injection de commandes systèmes.
V7 : Faiblesse dans le contrôle des données en entrée permettant une attaque de type XSS.
V8 : Faiblesse dans la manipulation des chemins
V9 : Faiblesses dans le contrôle des données utilisées dans une requête SQL.
V10 : Utilisation de générateur d'aléa considéré comme non sûr.
V11 : Présence de fonction de traces trop verbeuses

Tableau 4 – Synthèse des vulnérabilités

6.3 Points remarquables

Points remarquables
P1 : L'application est développée sous la « norme » PEAR Coding Standards.
P2 : Les entêtes de fichiers et de fonctions sont renseignés.
P3 : Les contrôles de sécurité « natifs » sont activés par défaut.
P4 : L'APPS « SHIBBOLETH » ne présente pas de vulnérabilité.
P5 : Le mécanisme d'authentification offerte par l'APPS « SHIBBOLETH » est suffisant d'un point de vue sécurité.
P6 : L'APPS « Checksum » ne présente pas de vulnérabilité même si elle utilise un algorithme (md5) obsolète.
P7 : L'APPS « File_Encryption » ne présente pas de vulnérabilité.
P8 : L'APPS « Antivirus » ne présente pas de vulnérabilité.
P9 : Les algorithmes de chiffrement utilisés sont issus d'une librairie considérée comme sûr.

Tableau 5 – Synthèse des points remarquables