# MediaWiki Security Assessment
## An OWASP Approach

Francesco Paolo Di Lorenzo    Gianmarco Forcella
Lorenzo Takanen

# Contents

**1 Introduction** **2**
1.1 Abstract . . . . . 2
1.2 Group composition . . . . . 2
1.3 The Mediawiki web application . . . . . 3

**2 Log meetings** **4**
2.1 First meeting . . . . . 4
2.2 Second meeting . . . . . 4
2.3 Third meeting . . . . . 4
2.4 Fourth meeting . . . . . 5
2.5 Fifth meeting . . . . . 5
2.6 Sixth meeting . . . . . 5

**3 Executive Summary** **5**
3.1 RIPS . . . . . 6
3.2 OWASP ZAP . . . . . 6
3.3 NESSUS . . . . . 7
3.4 PHPCS . . . . . 8
3.5 Final considerations on tools analysis . . . . . 9

**4 V6: Stored Cryptography Verification Requirements Analysis** **10**
4.1 V6.1 Data Classification . . . . . 10
4.2 V6.2 Algorithms . . . . . 13
4.3 V6.3 Random Values . . . . . 21
4.4 V6.4 Secret Management . . . . . 24

**5 Conclusion** **26**

# 1 Introduction

## 1.1 Abstract

This report is the final product of the group project assigned by Professor Francesco Parisi Presicce, for the Security in Software Applications class, for the Master Degree in Cybersecurity, in the accademic year 2019/2020.
The objective of the project is to verify that the *MediaWiki* web application meets the security requirements defined by the `OWASP` standard.
To analyze in detail all the requirements to be met, various groups were formed and each of them was assigned a requirement.
The task of each group is to verify that the application meets the assigned requirement.

## 1.2 Group composition

Our group is composed mainly by 3 students, listed below in alphabetical order:

1. Francesco Paolo Di Lorenzo
   (i) `studentID`: 1712990
   (ii) `email`: dilorenzo.1712990@studenti.uniroma1.it
2. Gianmarco Forcella
   (i) `studentID`: 1725967
   (ii) `email`: forcella.1725967@studenti.uniroma1.it
3. Lorenzo Takanen
   (i) `studentID`: 1772077
   (ii) `email`: takanen.1772077@studenti.uniroma1.it

Our job is to check the V6 requirements of the OWASP standard: `Stored Cryptography Verification Requirements`

## 1.3 The Mediawiki web application

MediaWiki is a web-based `wiki` software application. Developed by the Wikimedia Foundation, it is used for the execution of foundation projects, such as *Wikipedia*, *Wiktionary* and *Wikinews*.
MediaWiki is also used by other wikis internationally for the enhancement of websites. The application is written in the `PHP` programming language and uses a back-end database.
MediaWiki software is free and open source and is distributed under the GNU GPL 2 license or later.
The related documentation is distributed under the `Creative Commons BY-SA 3.0 license agreement` and is partially available in the public domain. The development of MediaWiki has favored the use of open source multimedia formats.

# 2 Log meetings

## 2.1 First meeting

During our first meeting (occurred on the 15th of November), we first started reading the sixth chapter of OWASP's Application Security Verification Standard document, in order to understand which was the requirement. After that, we started working on the analysis of the requirement 6.1

- Reading of OWASP's Application Security Verification Standard document;
- Analysis of Requirement 6.1.

## 2.2 Second meeting

In our second meeting (occurred on the 22nd of November), we started running the static analysis with some tools, which eventually did not produce no results for our analysis. We then discovered PHPCS, which was at least able to provide us the references of the .php files that uses algorithms. We then divided in 3 blocks the files that the program found and scheduled a new appointment for the 29th of the same month, in order to discuss our discoveries on these files together.

- Tried to run several static analysis programs, with no luck;
- Used PHPCS to at least find all the .php files that makes use of a crypto algorithm;
- Divided these files in 3 blocks, so that we could split the work among us.

## 2.3 Third meeting

In our third meeting (occurred on the 29th of November), we reviewed our discoveries on the files that PHPCS found, seeing if they could lead us to any direction with the requirements. Doing so, we were able to start working on the requirement 6.2 and fullfil the first analyses.

- Review of PHPCS files;
- Started working on the requirement 6.2;

## 2.4 Fourth meeting

In our fourth meeting (occurred on the 14th of December), we finished working on the requirement 6.2, after a long analysis of the whole thing. We then splitted jobs so that we could proceed, in parallel, with the remaining requirements;

- Review of the requirement 6.2;
- Finished working on the requirement 6.2;
- Division of the remaining work;

## 2.5 Fifth meeting

In our fifth meeting (occurred on the 21th of December), we discussed the outputs that the group member that had to work on the requirement 6.1 had to, and started also discussing about the 6.3.

- Review of the requirement 6.1;
- Started the review of requirement 6.3.

## 2.6 Sixth meeting

In our sixth meeting (occurred on the 2nd of January), we finally met to finish the whole work, including what was left of the requirement 6.3 and discussing the requirement 6.4.

- Review of the requirement 6.3;
- Started and finished the requirement 6.4

# 3 Executive Summary

To carry out requested analysis, we have been used various tools for static and dynamic analisys. The main goal of this step is to highlight the parts of code that may have vulnerability associated to V6 requirements in order to narrow the field of the manual review of the code.

## 3.1 RIPS

RIPS is the most popular static code analysis tool to automatically detect vulnerabilities in PHP applications. By tokenizing and parsing all source code files, RIPS is able to transform PHP source code into a program model and to detect sensitive sinks (potentially vulnerable functions) that can be tainted by user input (influenced by a malicious user) during the program flow. Besides the structured output of found vulnerabilities, RIPS offers an integrated code audit framework. The scan performed with RIPS (Figure 1) found many vulnerability but nothing related with the cryptography verification requirements.

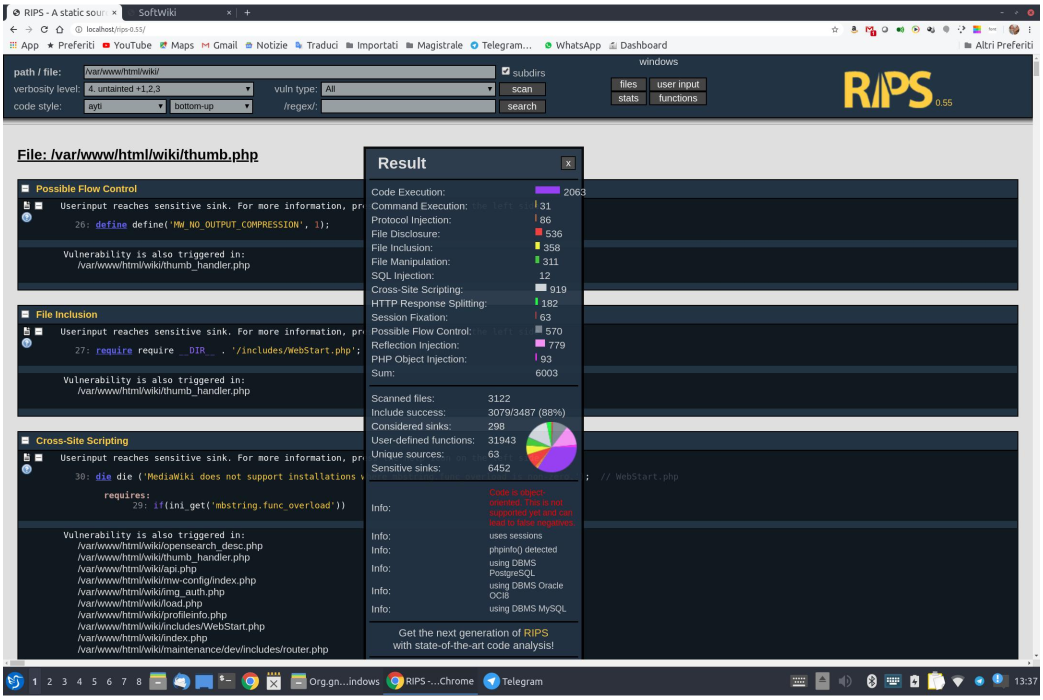


Figure 1: RIPS scan results

## 3.2 OWASP ZAP

The OWASP Zed Attack Proxy (ZAP) is one of the world's most popular free security tools. It can help you automatically find security vulnerabilities in your web applications while you are developing and testing your applications. Its also a great tool for experienced pentesters to use for manual security testing. At its core, ZAP is what is known as a "man-in-the-middle proxy". It stands between the tester's browser and the web application so that it can intercept and inspect messages sent between browser and web application, modify the contents if needed, and then forward those packets on to the destination. It can be used as a stand-alone application, and as a daemon process. The easiest way to start using ZAP is via the Quick Start tab. Quick Start is a ZAP add-on that is included automatically when you installed ZAP, so in our analysis we performed a automated scan. Also in this case the vulnerability found (Figure 2) do not concern with the requirements object of our analysis.

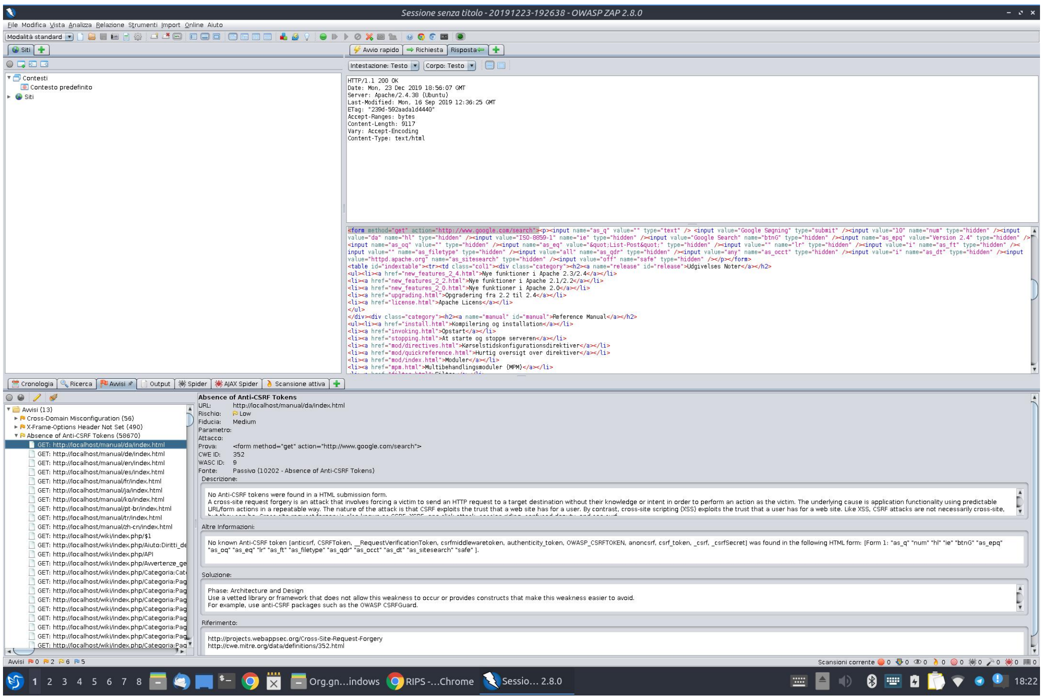


Figure 2: ZAP scan results

## 3.3 NESSUS

Nessus is a remote security scanning tool, which scans a computer and raises an alert if it discovers any vulnerabilities that malicious hackers could use to gain access to any computer you have connected to a network. It does this by running over 1200 checks on a given computer, testing to see if any of these attacks could be used to break into the computer or otherwise harm it.

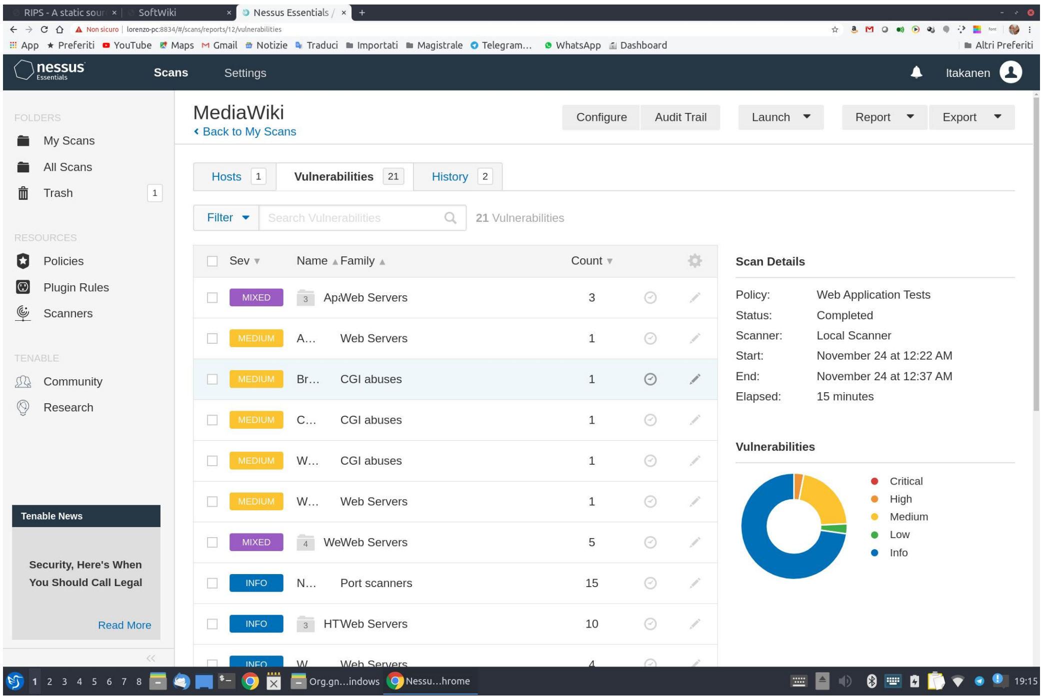

Figure 3: Nessus scan results

## 3.4 PHPCS

PHP_CodeSniffer is a PHP5 script that tokenises and "sniffs" PHP, JavaScript and CSS files to detect violations of a defined coding standard. It is an essential development tool that ensures your code remains clean and consistent. It can also help prevent some common semantic errors made by developers. Despite neither this tool match with the scope of our analysis we decide to use this tool as starting point of our manual analysis to extract the location of all the cryptographic funcion in the code.
This is an extract of the results obtained at the end of the static analysis carried out with phpcs;

```
FILE: /var/www/html/wiki/thumb.php
----------------------------------------------------------------------

FOUND 0 ERRORS AND 6 WARNINGS AFFECTING 5 LINES
----------------------------------------------------------------------

  27 | WARNING | Possible RFI detected with __DIR__ on require
 393 | WARNING | Crypto function md5 used.
 403 | WARNING | Function register_shutdown_function() that supports
     callback detected
 403 | WARNING | Function handling function register_shutdown_function()
     detected with dynamic parameter
```

```
 424 | WARNING | Crypto function sha1 used.
 633 | WARNING | Possible XSS detected with $content on echo
--------------------------------------------------------------------------

...
...

FILE: /var/www/html/wiki/includes/session/Session.php
-------------------------------------------------------------------
FOUND 0 ERRORS AND 18 WARNINGS AFFECTING 15 LINES
-------------------------------------------------------------------
 396 | WARNING | Crypto function hash_pbkdf2 used.
 424 | WARNING | Crypto function openssl_encrypt used.
 426 | WARNING | Crypto function openssl_error_string used.
 429 | WARNING | Crypto function mcrypt_encrypt used.
 447 | WARNING | Crypto function base64_encode used.
 447 | WARNING | Crypto function base64_encode used.
 448 | WARNING | Crypto function hash_hmac used.
 449 | WARNING | Crypto function base64_encode used.
 483 | WARNING | Crypto function hash_hmac used.
 484 | WARNING | Crypto function base64_decode used.
 493 | WARNING | Crypto function openssl_decrypt used.
 494 | WARNING | Crypto function base64_decode used.
 494 | WARNING | Crypto function base64_decode used.
 497 | WARNING | Crypto function openssl_error_string used.
 502 | WARNING | Crypto function mcrypt_decrypt used.
 503 | WARNING | Crypto function base64_decode used.
 503 | WARNING | Crypto function base64_decode used.
 515 | WARNING | Crypto function base64_decode used.
-----------------------------------------------

...
...

Time: 1 mins, 4.06 secs; Memory: 242.01MB
```

## 3.5 Final considerations on tools analysis

None of the different tool we tried gave us significant results in discovering vulnerability relative to the OWASP V6 requirement. This kind of requirement must be verified through manual analysis of the code.

# 4 V6: Stored Cryptography Verification Requirements Analysis

## 4.1 V6.1 Data Classification

Data is the most important assets of an application. it is very important that they are processed, stored and transmitted securely.
This requirement requires an assessment of how private data is protected within the application.

| | |
|---|---|
| **6.1.1** | **Verify that regulated private data is stored encrypted while at rest, such as personally identifiable information (PII), sensitive personal information, or data assessed likely to be subject to EU's GDPR**. |
| **Description** | Encryption refers to the procedure that converts clear text into a hashed code using a key, where the outgoing information only becomes readable again by using the correct key. This minimises the risk of an incident during data processing, as encrypted contents are basically unreadable for third parties who do not have the correct key. Encryption is the best way to protect data during transfer and **one way to secure stored personal data**. If private data is not encrypted there is a possibility that sensitive data will be exposed to malicious parties. |
| **Tools** | No Tools Used |
| **Testing** | A manual code review was performed. The review focused above all on those parts of the application where new users are created and stored in **shared caches** and **databases**. The *saveToCache()* function in the */includes/users/User.php* file, stores a user and all his private data in a shared cache. Starting from where the user is actually created and populated with private data, it turns out that this data is not encrypted neither before being stored in the shared cache, nor when the user is populated with that data. The same situation when the user is stored in databases, no encryption of his private data is performed, as can be seen in the *addToDatabase()* function in the */includes/users/User.php* These two examples therefore already allow to conclude that in the targeted points that have been analyzed, there is a lack of encryption of the user's private data (with the exception of the password). |
| **Recommended solution** | Based on what we discovered with the manual revision of the code, we can say that to satisfy the requirement, it would be advisable to make sure that whenever the user's private data is stored in a database or on a shared cache, these are encrypted correctly, so that they cannot be used by malicious parties. |
| **Verdict** | NOT PASSED |

| | |
|---|---|
| **6.1.2** | **Verify that regulated health data is stored encrypted while at rest, such as medical records, medical device details, or de-anonymized research records**. |
| **Description** | In order to decide whether the requirement is violated or not, one **must** know the scope of the application. And that's the core problem for this requirement: **Wikimedia** doesn't have a precise scope. The application is way too general and, in fact, is a program that has to be personalized, depending on the need for the final customer. But, aside from that, it is unlikely that WikiMedia will contain such data as per the requirement, since it has been designed to be a Wiki application. Because of that, it is more easy to find medical researchs or documentation instead of records and similar. |
| **Tools** | No Tools Used |
| **Testing** | - |
| **Recommended solution** | For the reason described above, we say that this requirement is not on scope and, therefore, not subject to our analysis. |
| **Verdict** | NOT PASSED |

| | |
|---|---|
| **6.1.3** | **Verify that regulated financial data is stored encrypted while at rest, such as financial accounts, defaults or credit history, tax records, pay history, beneficiaries, or de-anonymized market or research records**. |
| **Description** | In order to decide whether the requirement is violated or not, one **must** know the scope of the application. And that's the core problem for this requirement: **MediaWiki** doesn't have a precise scope. The application is way too general and, in fact, is a program that has to be personalized, depending on the need for the final customer. But, aside from that, for the way MediaWiki was built, it can't possibly store such sensible information as the financial data of an individual. |
| **Tools** | No Tools Used |
| **Testing** | - |
| **Recommended solution** | For the reason described above, we say that this requirement is not on scope and, therefore, not subject to our analysis. |
| **Verdict** | NOT PASSED |

## 4.2 V6.2 Algorithms

Cryptographic algorithms that a short time ago were considered strong and secure, today potentially they can no longer be. To cope with this it may be necessary to modify these algorithms with more secure algorithms. This section is considered mandatory for developers.

| | |
|---|---|
| **6.2.1** | **Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable Padding Oracle attacks**. |
| **Description** | Handling errors securely is a key aspect of secure coding. There are two types of errors that deserve special attention. The first is exceptions that occur in the processing of a security control itself. It's important that these exceptions do not enable behavior that the countermeasure would normally not allow. The other type of security-relevant exception is in code that is not part of a security control.<br>The padding oracle attack can be applied to the CBC mode of operation, where the "oracle" (usually a server) leaks data about whether the padding of an encrypted message is correct or not. The standard implementation of CBC decryption in block ciphers is to decrypt all ciphertext blocks, validate the padding, remove the PKCS7 padding, and return the message's plaintext. If the server returns an "invalid padding" error instead of a generic "decryption failed" error, the attacker can use the server as a padding oracle to decrypt messages. |
| **Tools** | No Tools Used. |
| **Testing** | A manual review of the code was performed. As we saw in point 6.1.1 MediaWiki doesn't use any cryptographic module to crypt and decrypt data because most of data are stored not crypted and password are managed with hash functions. Through a complete search inside the code we point out that the only decription function used was inside the session.php class inside the *getsecret()* function. Failing of this function seems correctly handled and messages of exception throw are the same from external module called openssl_decrypt [4], using officially approved cryptographic function Advanced Encryption Standard (AES) [3]. So we can say that for the cryptographic module inserted Padding Oracle attacks are not enabled. |
| **Recommended solution** | |
| **Verdict** | PASSED |

| | |
|---|---|
| **6.2.2** | **Verify that industry proven or government approved cryptographic algorithms, modes, and libraries are used, instead of custom coded cryptography**. |
| **Description** | |
| **Tools** | No Tools Used |
| **Testing** | With a manual review of the code we found out that no custom coded cryptography was used and all the cryptographic algorithms used to encrypt data are inserted in the FIPS 140-2 Annex A [3] list. |
| **Recommended solution** | - |
| **Verdict** | PASSED |

| | |
|---|---|
| **6.2.3** | **Verify that encryption initialization vector, cipher configuration, and block modes are configured securely using the latest advice**. |
| **Description** | In order for a cryptographic algorithm to work, sometimes it is needed to set some features such as the initialization vector, the cipher configuration etc. in order to have a good source for providing the cryptograh. |
| **Tools** | No Tools Used. |
| **Testing** | The requirement requires a two step procedure in order to understand whether it can be verified or not:<br>• First of all, a manual static analysis on PHP's crypto algorithms in order to understand whether, in the code, the requirement is satisfied;<br>• After that, a dynamic analysis should be performed in order to understand whether the algorithms used in Wikimedia could suffer, in practice of these problems or not |
| **Recommended solution** | The requirement is out of scope. While we could, in fact, check with a manual static analysis that the requirement is met, the dynamic analysis is hard to perform, not to mention that is not the objective of our security analysis. For this reason, we say that that the requirement is not met. |
| **Verdict** | OUT OF SCOPE. |

| | |
|---|---|
| **6.2.4** | **Verify that random number, encryption or hashing algorithms, key lengths, rounds, ciphers or modes, can be reconfigured, upgraded, or swapped at any time, to protect against cryptographic breaks**. |
| **Description** | Configuration of cryptographic and hash modules and algorithm must be flexible and upgradable easily. |
| **Tools** | No tools used |
| **Testing** | A manual review of the code was performed. In our analysis we found out that all the configuration are inserted out of the code in the */include/DefaultSettings.php* file, such as for the configuration relative to the hash function to use in order to store passwords (figure 4). In the configuration file is set the class that will handle passwords hash (standard */password/Pbkdf2Password.php* is selected, others for backwards compatibility), algorithm used, numbers of rounds and ciphers length, so this value are easily changed for upgrading in order to protect against cryptographic breaks. |
| **Recommended solution** | - |
| **Verdict** | PASSED |

```php
$wgPasswordConfig = [
    'A' => [
        'class' => 'MWOldPassword',
    ],
    'B' => [
        'class' => 'MWSaltedPassword',
    ],
    'pbkdf2-legacyA' => [
        'class' => 'LayeredParameterizedPassword',
        'types' => [
            'A',
            'pbkdf2',
        ],
    ],
    'pbkdf2-legacyB' => [
        'class' => 'LayeredParameterizedPassword',
        'types' => [
            'B',
            'pbkdf2',
        ],
    ],
    'bcrypt' => [
        'class' => 'BcryptPassword',
        'cost' => 9,
    ],
    'pbkdf2' => [
        'class' => 'Pbkdf2Password',
        'algo' => 'sha256',
        'cost' => '10000',
        'length' => '128',
    ],
];
```

Figure 4: /include/DefaultSettings.php Password Hash configuration

| | |
|---|---|
| **6.2.5** | **Verify that known insecure block modes (i.e. ECB, etc.), padding modes (i.e. PKCS#1 v1.5, etc.), ciphers with small block sizes (i.e. Triple-DES, Blowfish, etc.), and weak hashing algorithms (i.e. MD5, SHA1, etc.) are not used unless required for backwards compatibility**. |
| **Description** | In Cryptography Well-known insecure block mode like ECB (Electronic Codebook), encrypts equal input blocks as equal output blocks (does not provide cryptographic diffusion). Block mode like this may compromise the entire encryption. Encoding methods like PKCS if used with certain protocols are vulnerable to attacks [1], using of chipers with small block sized and continued use of weak hashing algorithms puts your clients' sensitive data at risk. |
| **Tools** | `phpcs`: This tool was used to perform a static analysis of the entire application in search of the cryptographic algorithms used. |
| **Testing** | The static analysis showed the presence of about 146 files in which cryptographic algorithms are used. Phpcs has highlighted the application's use of 32 different cryptographic algorithms including md5, hash, sha1, hmac, mcrypt_create_iv, crc32 etc. Algorithms such as md5 and sha1 are considered weak algorithms, their use within the application is very wide, the mcrypt_create_iv function was deprecated in PHP 7.1.0, and REMOVED in PHP 7.2.0 and not updated anymore [2]. |
| **Recommended solution** | The suggested solution is to update the application to use strong cryptographic algorithms, make sure to use secure block modes, chipers with a non small block size and not vulnerable padding modes. |
| **Verdict** | NOT PASSED |

| | |
|---|---|
| **6.2.6** | **Verify that nonces, initialization vectors, and other single use numbers must not be used more than once with a given encryption key. The method of generation must be appropriate for the algorithm being used**. |
| **Description** | In order to understand this requirement, it was necessary to read all the .php files that used any of PHP's crypto functions (or custom ones made from Wikimedia's developers) and understand the flow of the work. |
| **Tools** | No Tools Used. |
| **Testing** | Being that most of the time these crypto functions are used for generating a key for an associative array, the requirement is satisfied. Should, in fact, a for loop use the same data for generating an hash to insert inside an associative this would generate a data loss, since the key has to be one. And since Wikimedia's developers knew this, they made sure that this problem could never happen. |
| **Recommended solution** | // |
| **Verdict** | PASSED. |

| | |
|---|---|
| **6.2.7** | **Verify that encrypted data is authenticated via signatures, authenticated cipher modes, or HMAC to ensure that ciphertext is not altered by an unauthorized party**. |
| **Description** | The purpose of data authentication is to make sure the data is not changed in transit. To achieve this goal, the transmitter accompanies the frame with a specific code known as the Message Integrity Code (MIC). The MIC is generated by a method known to both receiver and transmitter. An unauthorized device will not be able to create this MIC. The receiver of the frame will repeat the same procedure and if the MIC calculated by the receiver matches the MIC provided by the transmitter, the data will be considered authentic. The MIC is also referred to as Message Authentication Code (MAC). |
| **Tools** | No tools used. |
| **Testing** | A manual review of the code was performed. In the code before using decrypt funcion (*openssl_decrypt()*[4]) data encrypted are verified through *hash_hmac()* [5] function, using *sha256* algorithm in order to control if data has been tampered. |
| **Recommended solution** | - |
| **Verdict** | PASSED |

| | |
|---|---|
| **6.2.8** | **Verify that all cryptographic operations are constant-time, with no 'short-circuit' operations in comparisons, calculations, or returns, to avoid leaking information**. |
| **Description** | Verify the constant time for a cryptographic operation is fundamental: should, in fact, a cryptographic function takes a long time for creating the encryption, this would result in a block of the application until the operation is finished! |
| **Tools** | No Tools Used |
| **Testing** | In order to test this, a dynamic analysis should be carried out, in order to stress the application with lots of requests and see after, i.e. 100000 requests, if the application took significaty time in performing the cryptographic operations or not. The same goes for the data leaking part: without performing a dynamic analysis, it would be almost imossible to be able to logically follow the information floq: sure, an idea could come in mind by reading the code but that doesn't mean that, eventually, the guess could be right. |
| **Recommended solution** | Perform a dynamic analysis in order to understand whether this requirement is met or not. |
| **Verdict** | OUT OF SCOPE |

## 4.3 V6.3 Random Values

True pseudo-random number generation (PRNG) is incredibly difficult to get right. Generally, good sources of entropy within a system will be quickly depleted if over-used, but sources with less randomness can lead to predictable keys and secrets

| | |
|---|---|
| **6.3.1** | **Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved cryptographically secure random number generator when these random values are intended to be not guessable by an attacker**. |
| **Description** | When a non-cryptographic PRNG is used in a cryptographic context, it can expose the cryptography to certain types of attacks. Often a pseudo-random number generator (PRNG) is not designed for cryptography. Weak generators generally take less processing power and/or do not use the precious, finite, entropy sources on a system. While such PRNGs might have very useful features, these same features could be used to break the cryptography. |
| **Tools** | No tools |
| **Testing** | A manual review of the code was carried out. The class interested in this point is MWCryptRand.php and the function *realGenerate($bytes, $forceStrong)* where *$bytes* is the length of the number to be generated and *$forcestrong* a boolean variable that allow to force the function to generate a cryptographically secure random number. So this function may generate both cryptographically secure random number then not secure pseudorandom data. A boolean variable *strong* is set to indicate the calling function whether the random source used is cryptographically strong or not. The modules used in the function are *mcrypt_create_iv*, *openssl_random_pseudo_bytes*, or *mt_rand*. The last one does not generate cryptographically secure values, and should not be used for cryptographic purposes. The review of the code point out that the MWCryptRand function is called without asking for cryptographically strong random data even if these are intended not to be guessable by an attacker, such as in *sesssion.php* class where a unique random identifier is generated to identify a user's session or a CSRF token is generated to avoid cross-site request forgery vulnerability. |
| **Recommended solution** | Uses $forceStrong = true or the PHP 7.0 function *random_bytes ( int $length ) : string* to generate cryptographic random bytes that are suitable for cryptographic use (cryptographically secure pseudo-random bytes) when the random number has not to be guessable by an attacker such as in the generation of Cross Site Request Forgery or session Id. |
| **Verdict** | NOT PASSED |

| | |
|---|---|
| **6.3.2** | **Verify that random GUIDs are created using the GUID v4 algorithm, and a cryptographically-secure pseudo-random number generator (CSPRNG). GUIDs created using other pseudo-random number generators may be predictable**. |
| **Description** | A Universally Unique Identifier (UUID) is an identifier standard used in many non-MultiValue databases and software to generate a Unique ID outside of using incremental numbers. The intent of a UUID is to generate an ID that can be used across different databases, and will always be unique. This is different from a sequential number that would only be unique in the file or table itself, and not across files, accounts, or databases. Version 4 is also commonly referred to as a GUID. |
| **Tools** | No tools |
| **Testing** | A manual review of the code was carried out. The class interested in this point is UIDGenerator.php where the function *newUUIDv4()* was arranged for this purpose but practically never used in the whole code, leaving the assignment of UUID to the DBMS. Farther, inside *newUUIDv4()* function, the call to *generate()* function of MWCryptRand is made without force requiring cryptographically secure random number through $forcestrong variable. |
| **Recommended solution** | Modify the *newUUIDv4()* function in order to make GUID generated not predictable and use this function to generate GUID. |
| **Verdict** | NOT PASSED |

| | |
|---|---|
| **6.3.3** | **Verify that random numbers are created with proper entropy even when the application is under heavy load, or that the application degrades gracefully in such circumstances**. |
| **Description** | Like point 6.3.1 but tested when the application is under heavy load. |
| **Tools** | No tools used |
| **Testing** | As wrote in 6.3.1 point OWASP asked to verify that all random numbers, etc. are generated using a cryptographic module with no vulnerability. As in the previous point this requirement is not satisfied because the requirement is the same with the additional condition to verify it when the application is under heavy load. |
| **Recommended solution** | - |
| **Verdict** | NOT PASSED |

## 4.4 V6.4 Secret Management

Although this section is not easily penetration tested, developers should consider this entire section as mandatory even though L1 is missing from most of the items.

| | |
|---|---|
| **6.4.1** | **Verify that a secrets management solution such as a key vault is used to securely create, store, control access to and destroy secrets**. |
| **Description** | A Secret Key is a string that is usually used to grant access to APIs / services. The role of a Key Vault is to maintain (and, if needed, create or delete) a secret. In enterprise applications, this is usually carried out with Microsoft's Azure Key Vault, but most of the times this is a part that gets implemented from scratch. But this is not the only thing that needs to be seen: it also requires to see if there are any hard-coded credentials on the application. |
| **Tools** | No tools used |
| **Testing** | In order to test this requirement, it first was necessary to perform a manual code analysis on the whole application. First of all, we've seen if there were some external libraries that required any secret key to work and, if so, if they were hardcoded and didn't find anything that could attract our interest. Afterwards, we scanned the application to see whether there were hardcoded passwords or not and we've found that the application, after its installation, saves the credentials for the database connection in a file called LocalSettings.php for further use. Then we started to realize that secrets are created as random generated strings and are not properly managed. |
| **Recommended solution** | In order to proper handle the secret keys, a Key Vault should either be implemented by hand or use a third party one. Also, it should be changed the fact that the application saves the database credentials in a .php file and, in stead, put them in a separate file outside the project. |
| **Verdict** | NOT PASSED |

| | |
|---|---|
| **6.4.2** | **Verify that key material is not exposed to the application but instead uses an isolated security module like a vault for cryptographic operations**. |
| **Description** | In this requirement, an application should have a dedicated module for handling all the possible secrets, according to the CWE 320 |
| **Tools** | No tools used. |
| **Testing** | The same testing procedure that was performed for the requirement 6.4.1 was performed for the 6.4.2. Though the testing of this requirement comes directly from the previous requirement: if there's no clear management for the secrets, there surely isn't as well a dedicated module |
| **Recommended solution** | Implement a separate module that handles all the possible secrets. |
| **Verdict** | NOT PASSED. |

# 5 Conclusion

Throughout this project, we learned how to perform a good static and manual code analysis on an application.
Though we couldn't make a full analysis (because of the assignment), we were able to analyze probably one of the most important part of every application: the cryptographic one. Without it, every application could lead in possible data leak / loss.
Though we couldn't do a dynamic analysis to analyze everything, we are very satisfied of what achieved. The Mediawiki application does not verify many, if not most, of the cryptographic requirements. So we cannot say that this application has a good level of security. Despite this, however, we must remember that in this analysis we considered the application as a level 3 application, but basically Mediawiki is a level 1 application. Furthermore it has been tested locally and using the default configuration, therefore the final verdict does not represent the real security level of the application, our solutions are all based on the adopted configuration.
Finally we can conclude that Mediawiki from a cryptographic point of view has a low level of security.

# References

[1] RSA Laboratories' Bulletin, Recent Results on PKCS #1: RSA Encryption Standard,

ftp://ftp.rsa.com/pub/pdfs/bulletn7.pdf

[2] PHP Manual,mcrypt_create_iv RSA Encryption Standard,
https://www.php.net/manual/en/function.mcrypt-create-iv.php

[3] Annex A: Approved Security Functions for FIPS PUB 140-2
https://csrc.nist.gov/csrc/media/publications/fips/140/2/final/documents/fips1402annexa.pdf

[4] PHP Manual, openssl_decrypt
https://www.php.net/manual/en/function.openssl-decrypt.php

[5] PHP Manual, hash_hmac
https://www.php.net/manual/en/function.hash-hmac.php