Group Project OWASP

Applying OWASP principles to FluxBB application

Riccardo Chiaretti 1661390
Serena Ferracci 1649134
Matteo Mariani 1815188

Contents

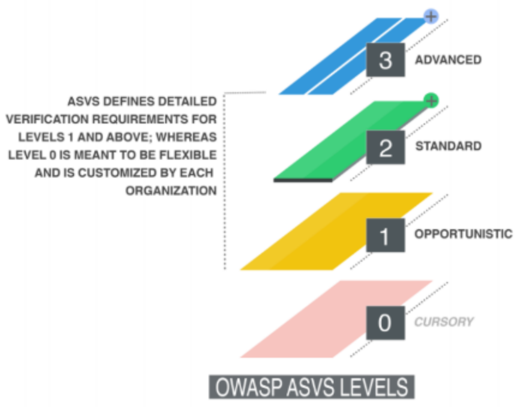
1 Introduction

The **OWASP Application Security Verification Standard (ASVS)** project provides a basis for testing web application technical security controls and also provides developers with a list of requirements for secure development.

The primary aim of the OWASP Application Security Verification Standard (ASVS) project is to normalize the range in the coverage and level of rigor available in the market when it comes to performing Web application security verification using a commercially-workable open standard.

The ASVS defines three security verification levels, with each level increasing in depth:

- *ASVS Level 1* is meant for all software;
- *ASVS Level 2* is for applications that contain sensitive data, which requires protection;
- *ASVS Level 3* is for the most critical applications - applications that perform high value transactions, contain sensitive medical data, or any application that requires the highest level of trust.


3 ADVANCED
2 STANDARD
1 OPPORTUNISTIC
0 CURSORY
ASVS DEFINES DETAILED VERIFICATION REQUIREMENTS FOR LEVELS 1 AND ABOVE; WHEREAS LEVEL 0 IS MEANT TO BE FLEXIBLE AND IS CUSTOMIZED BY EACH ORGANIZATION
OWASP ASVS LEVELS

The higher is the level, the more specific will be the requirements that an application has to satisfy in order to be “well-defined” and secure according to OWASP standards.

2 Assignment

The assignment requires to analyze FluxBB application by using OWASP principles.
FluxBB is a PHP open-source forum application released under the GNU General Public Licence that uses an SQL database. It was conceived and designed to be fast and light-weight. For our analysis, we focused on version v1.5.10.

All of us did not have any experience with PHP programming but we tried to overcome our limits by checking the documentation and taking care of some suggestions related to well-defined programming rules for PHP.

The categories assigned to our group were:

- *v1: Architecture, design and threat modeling*;
- *v3: Session Management.*

According to the assignment, the application should be analyzed checking whether it satisfies *all* the requirements proposed in each assigned category. As a consequence, we will treat FluxBB as ASVS Level 3 application.

For this reason, sometimes it can happen that the analysis required by the standard will be either too extreme or inapplicable to our task, making the analysis too demanding with respect to what the application has been developed for.

Throughout the analysis, it will be given a mark for each requirement:

- *satisfied*, if the requirement is completely satisfied by the application;
- *partially satisfied*, if the requirement is composed by multiple checks that have to be done and the application satisfies only a part of them. This mark will be also used when we do not find a proper way to verify a requirement (e.g. it is too vague and we interpret it) but we manage to find a suitable answer;
- *not satisfied*, if the requirement is completely not satisfied;
- *not applicable*, if the requirement cannot be applied to the application.

3 v1 Architecture, design and threat modeling

Architecture, design and thread modeling OWASP category represents for sure the most challenging one, especially for some analysts that have to check whether its requirements are either fulfilled or not without neither have taken part of the development process or have been in contact with the developers to better understand the design process.

V1 requirements concern about the design and the formal structure of the application, checking whether its architecture is well-defined or not: each component - library, modules, classes - should have a precise role in the application, functional and "effective".

The **goal** of V1 requirements is to minimize the attack surface that could be maliciously exploited. Attack Surface Analysis is about mapping out what parts of a system need to be reviewed and tested for security vulnerabilities. The point of Attack Surface Analysis is to understand the risk areas in an application, to make developers and security specialists aware of what parts of the application are open to attack, to find ways of minimizing this, and to notice when and how the Attack Surface changes and what this means from a risk perspective. Understanding application design is a key activity to perform application threat modeling. Therefore, it will be easier to identify all possible risks.

As introduced before, checking whether FluxBB respects or not these requirements was not an easy task. Some of the considerations related to the entire V1 category were born from **conjectures** that we made in order to give an interpretation to some vague checks that OWASP suggests and then apply them on the application of interest.

3.1 Requirements

1.1

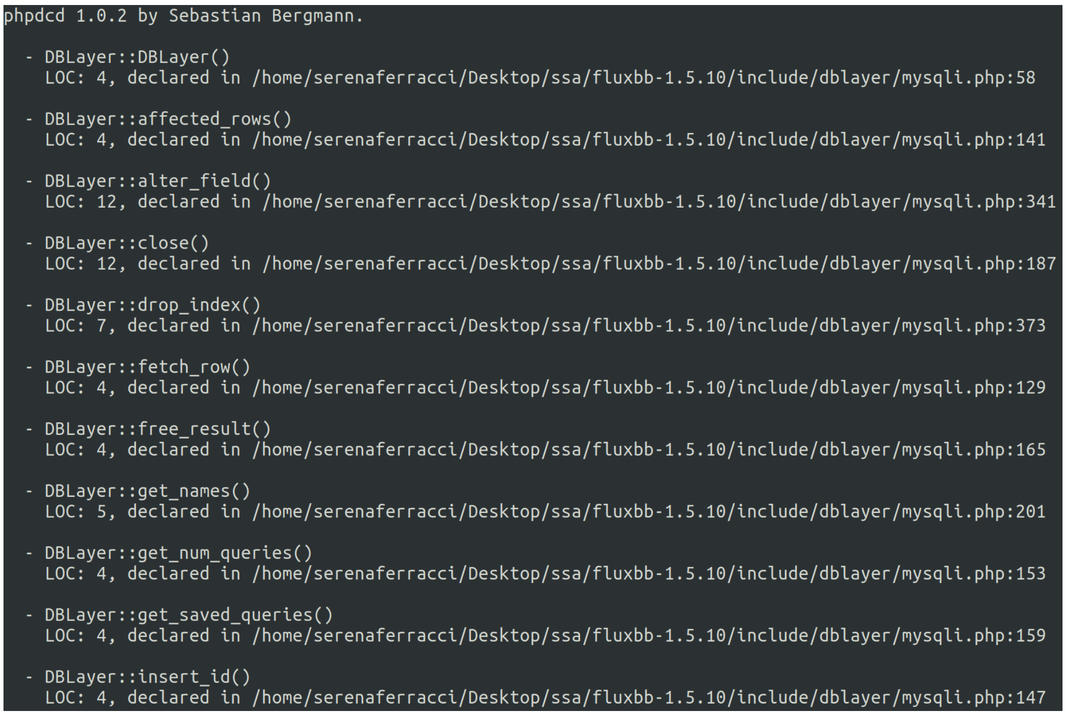
`Verify that all application components are identified and are known to be needed.`

The requirement has a dual purpose.
In many cases, the code contains third party components which can introduce publicly known vulnerabilities out of the developer's control, as well as, the latter may be tempted to keep dead code / dependencies that are not needed. This way of reasoning increases attack surface without adding any benefit in functionality.

Looking for third party components we execute a `grep` operation on the whole FluxBB's folder using as keywords "`require`" and "`include`". Analyzing the results we noticed that all the components included in each file are defined directed by the developers.

To validate our consideration we verified that there is no packet manager, like `composer` or `packagist`, to manage dependencies. Note that developers can also decide to not use a packet manager and incorporate the file implementing the third party component directly into the `include` folder of the application. However, this is not the case since the files present into that directory are written by the developers.

Thus, to verify whether there is dead code or not we have installed a PHP dead code detector named **phpdcd**. The output shows a list of 63 functions presumably useless.



```
phpdcd 1.0.2 by Sebastian Bergmann.

  - DBLayer::DBLayer()
    LOC: 4, declared in /home/serenaferracci/Desktop/ssa/fluxbb-1.5.10/include/dblayer/mysqli.php:58

  - DBLayer::affected_rows()
    LOC: 4, declared in /home/serenaferracci/Desktop/ssa/fluxbb-1.5.10/include/dblayer/mysqli.php:141

  - DBLayer::alter_field()
    LOC: 12, declared in /home/serenaferracci/Desktop/ssa/fluxbb-1.5.10/include/dblayer/mysqli.php:341

  - DBLayer::close()
    LOC: 12, declared in /home/serenaferracci/Desktop/ssa/fluxbb-1.5.10/include/dblayer/mysqli.php:187

  - DBLayer::drop_index()
    LOC: 7, declared in /home/serenaferracci/Desktop/ssa/fluxbb-1.5.10/include/dblayer/mysqli.php:373

  - DBLayer::fetch_row()
    LOC: 4, declared in /home/serenaferracci/Desktop/ssa/fluxbb-1.5.10/include/dblayer/mysqli.php:129

  - DBLayer::free_result()
    LOC: 4, declared in /home/serenaferracci/Desktop/ssa/fluxbb-1.5.10/include/dblayer/mysqli.php:165

  - DBLayer::get_names()
    LOC: 5, declared in /home/serenaferracci/Desktop/ssa/fluxbb-1.5.10/include/dblayer/mysqli.php:201

  - DBLayer::get_num_queries()
    LOC: 4, declared in /home/serenaferracci/Desktop/ssa/fluxbb-1.5.10/include/dblayer/mysqli.php:153

  - DBLayer::get_saved_queries()
    LOC: 4, declared in /home/serenaferracci/Desktop/ssa/fluxbb-1.5.10/include/dblayer/mysqli.php:159

  - DBLayer::insert_id()
    LOC: 4, declared in /home/serenaferracci/Desktop/ssa/fluxbb-1.5.10/include/dblayer/mysqli.php:147
```

We tried to manually check if all the hits were actually dead code. The first of them, that is `DBLayer::DBLayer()`, is only defined, but never used. Nonetheless, already the second result could be a false positive. Indeed, the class of object on which this function is called depends on the underling DBMS.

The same function is defined in more than one class, one for each type of DBMS. During the execution of the application only one of them will be used, while the others are never called. The functions cannot be considered to be dead code because the intent of the developer was to make the application compatible with multiple DBMS.

Considering that there were too many functions to check, we decide to test the soundness of the tool by randomly selecting some functions and search if they were effectively used or not. Concluding, we realized that some false positives are present.

FluBB's site has a section in which developers maintain a changelog of the the development (i.e. Development → Changelog), annotating the removal of some bugs and unused features, as well, as some further additions on the main project. Moreover, there is a page in which are described the features offered by the standard installation of FluxBB (i.e. About → Feature List).

For these reasons, the requirement is *satisfied.*

1.2

`Verify that all components, such as libraries, modules, and external systems, that are not part of the application but that the application relies on to operate are identified.`

FluxBB has no additional plug-ins in its standard installation - but they can be inserted by other developers that may borrow the application for their interests. The only components which are not part of the application but on which the application relies on are:

- Server in which the application runs;
- Database that stores information about the users;
- Browser, obviously.

During the analysis, we run the application using **XAMPP**, a free and open-source cross-platform web server solution stack package developed by Apache consisting mainly of the **Apache HTTP Server**, **MariaDB database**, and interpreters for scripts written in the PHP and Perl programming languages. We used both **Mozilla Firefox** and **Google Chrome** to test directly forum functionality. Since these tools are well documented, there is no inheritance of well-know vulnerabilities from these external systems.

Obviously, the developers are responsible for the tools used to correctly run the application: the installation page (i.e. Documentation → FluxBB v1.5 → Installing) gives some information on the required tools necessary to run it.

Thus, the requirement is *satisfied.*

1.3

`Verify that a high-level architecture for the application has been defined.`

Since the requirement was not precise in describing what degree of high-level architecture should be defined in order to fulfill the analysis, we assume that the definition of the high-level architecture of the application consists of providing a detailed and understandable documentation.

`FluxBB v1.5` documentation is present in the website (i.e. Documentation → FluxBB v1.5), giving some useful information about what the application does, how to install it and how to set up a forum. There is also a section that briefly explains some functions of the code (i.e. Documentation → FluxBB v1.5 → Functions) that may help in understanding at high-level what some sections of the code should do. On the contrary, `FluxBB v2.0` is not rightly documented, presumably because this last version of the forum is still under development.

Since there was not any example about how this requirement should be generally satisfied, we decided that FluxBB website was sufficiently good in describing the main functionalities of the application without going into unnecessary details.

Hence, the requirement is *satisfied*.

1.4

`Verify that all application components are defined in terms of the business functions and/or security functions they provide.`

FluxBB documentation explains functionality of the application, however it does not cover all available functionality, and often it does not go deeper on the security considerations and impact of the functionality. For what concerns business logic, the documentation appears to be more structured. Indeed there are multiple pages that:

- explain precisely the data that are stored in the database (i.e. Documentation → FluxBB v1.5 → Database structure);
- cover the most important global variables used across the code (i.e Documentation → FluxBB v1.5 → Global variables);
- describe some the behavior of some functions that are implemented in `common.php` and `functions.php` (i.e. Documentation → FluxBB v1.5

$\rightarrow$ Functions). Note that those files represent the core of the application business logic.

Since the requirement is satisfied uniquely with respect to the business logic, leaving the security aspect only slightly considered in the documentation, the requirement is *partially satisfied.*

1.5

`Verify that all components that are not part of the application but that the application relies on to operate are defined in terms of the functions, and/or security functions, they provide.`

Checking whether this requirement is satisfied or not is pretty challenging for outsider analysts, not only because we clearly do not know the design choices made by developers but also because the application does not seem to use any external components despite the ones described in `1.2` requirement analysis. Moreover, OWASP is very vague on what should be practically checked in these extra components.

For the sake of simplicity (and in order to give our interpretation on what it is required) we will consider the requirement as fully satisfied if these additional components are well-described in terms of their functionality in their respective web pages and/or manual. Assumed this, both Apache HTTP server[1] and MariaDB database [2] have a clear documentation in their respective web pages that well described their functionality, security aspects and functions/logic.

Since the requirement was interpreted according to our conjunctures we cannot safely mark this as “satisfied”. In order to keep track of the ambiguity that arose in the analysis, we decided to mark the result as *partially satisfied.*

1.6

`Verify that a threat model for the target application has been produced and covers off risks associated with Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of privilege (STRIDE).`

Threat modelling works to identify, communicate, and understand threats and mitigation within the context of protecting something of value.

[1]`https://httpd.apache.org/docs/2.4/`

[2]`https://mariadb.com/kb/en/library/documentation/`

Threat modelling can be applied to a wide range of things, including software, applications, systems, networks, distributed systems, things in the internet of things, business processes, etc. There are very few technical products which cannot be threat modelled; more or less rewarding, depending on how much it communicates, or interacts, with the world. Threat modelling can be done at any stage of development, preferably early - so that the findings can inform the design.

A threat model includes:

- A description/design/model of what developers are worried about;
- A list of assumptions that can be checked or challenged in the future as the threat landscape changes;
- A list of potential threats to the system;
- A list of actions to be taken for each threat;
- A way of validating the model and threats, and verification of success of actions taken.

Woefully, examining the documentation, there is not any information related to the adopted development process. Consequently, it is not possible to state whether there exists a STRIDE threat model that take into account security risks to which the application is vulnerable.

For this reason, the requirement is *not satisfied.*

1.7

`Verify all security controls (including libraries that call external security services) have a centralized implementation.`

Considering the available documentation, there is not any section dedicated to security services. Hence, we decided to give our interpretation to what it should be guaranteed.
In `php.ini` configuration file, there are all the security functionalities used by the server and that application inherits at start-up. This means that, since there is not a specific security module in FluxBB code, most of the security mechanisms are handled by the server.

For this reason, since Apache HTTP server has a centralized implementation that comes from XAMPP stack package installation, the requirement can be viewed as “satisfied”. However, since (i) we do not know how deeper the standard wants to verify the requirement and (ii) our considerations are based on an interpretation of what should be checked, we decided to mark the verification as *partially satisfied.*

1.8

```
Verify that components are segregated from each other via a
defined security control, such as network segmentation, firewall
rules, or cloud based security groups.
```

The requirement has to guarantee that there is a proper segregation between components from a network point of view, achieving data protection.

Network segmentation is the act or practice of splitting a computer network into subnetworks, each being a network segment. Advantages of such splitting are primarily for boosting performance and improving security.

Cloud resource segmentation is a process by which separate physical and virtual IT environments are created for different users and groups. The cloud-based resource segmentation process creates cloud-based security group mechanisms that are determined through security policies. Networks are segmented into logical cloud-based security groups that form logical network perimeters. Each cloud-based IT resource is assigned to at least one logical cloud-based security group. Each logical cloud-based security group is assigned specific rules that govern the communication between the security groups.

Considering the examined application and the absence of something related to these aspects both in code and documentation, it is not possible to verify whether such requirement is satisfied or not.

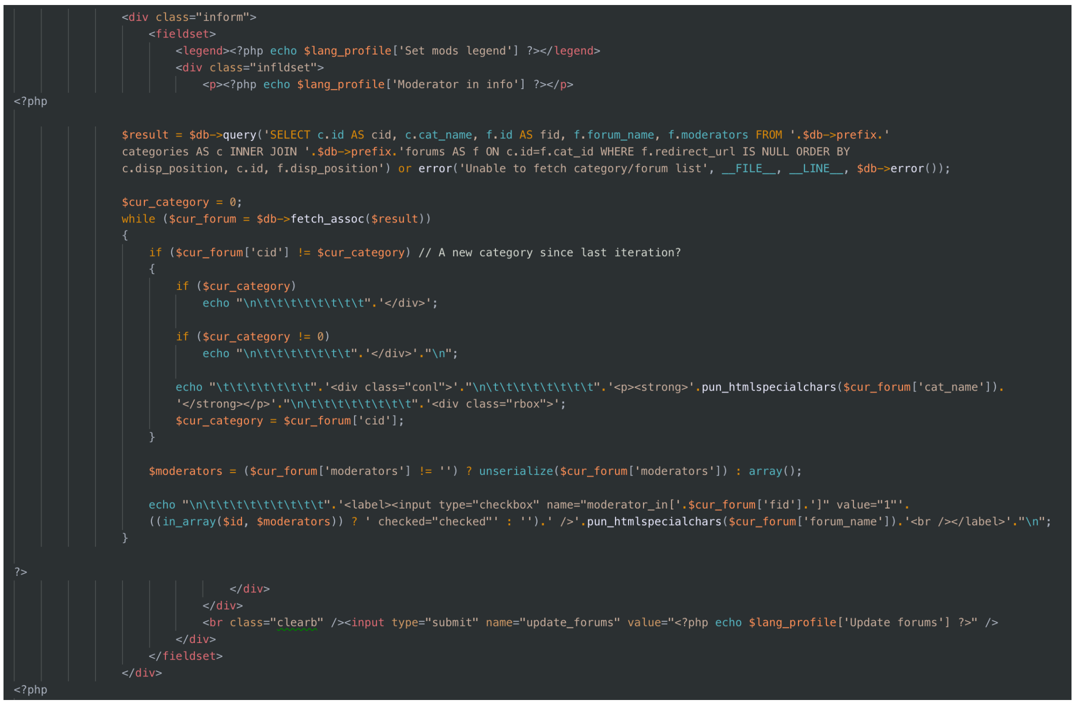
Thus, the requirement is *not applicable.*

1.9

```
Verify the application has a clear separation between the data
layer, controller layer and the display layer, such that security
decisions can be enforced on trusted systems.
```

The application does not enforce any modularity according to the *Model View Controller* (MVC) paradigm. Most of the functionalities related to the interaction with the user (View) and its management (Controller) are not clearly separated, with files containing both HTML code and the business logic.

The data layer (Model) is better isolated than the others: all the configuration files are confined into `dblayer` folder. Despite this, some of the queries on the database are tied into `.php` files that manage the View layer, such as `login.php`. In this file there is interleaving between HTML code and SQL queries without any controller in middle. An example is the following:



```
                <div class="inform">
                    <fieldset>
                        <legend><?php echo $lang_profile['Set mods legend'] ?></legend>
                        <div class="infldset">
                            <p><?php echo $lang_profile['Moderator in info'] ?></p>
<?php

                $result = $db->query('SELECT c.id AS cid, c.cat_name, f.id AS fid, f.forum_name, f.moderators FROM '.$db->prefix.'
                categories AS c INNER JOIN '.$db->prefix.'forums AS f ON c.id=f.cat_id WHERE f.redirect_url IS NULL ORDER BY
                c.disp_position, c.id, f.disp_position') or error('Unable to fetch category/forum list', __FILE__, __LINE__, $db->error());

                $cur_category = 0;
                while ($cur_forum = $db->fetch_assoc($result))
                {
                    if ($cur_forum['cid'] != $cur_category) // A new category since last iteration?
                    {
                        if ($cur_category)
                            echo "\n\t\t\t\t\t\t\t\t\t".'</div>';

                        if ($cur_category != 0)
                            echo "\n\t\t\t\t\t\t\t\t".'</div>'."\n";

                        echo "\t\t\t\t\t\t\t\t".'<div class="conl">'."\n\t\t\t\t\t\t\t\t\t".'<p><strong>'.pun_htmlspecialchars($cur_forum['cat_name']).
                        '</strong></p>'."\n\t\t\t\t\t\t\t\t\t".'<div class="rbox">';
                        $cur_category = $cur_forum['cid'];
                    }

                    $moderators = ($cur_forum['moderators'] != '') ? unserialize($cur_forum['moderators']) : array();

                    echo "\n\t\t\t\t\t\t\t\t\t\t".'<label><input type="checkbox" name="moderator_in['.$cur_forum['fid'].']" value="1"'.
                    ((in_array($id, $moderators)) ? ' checked="checked"' : '').' />'.pun_htmlspecialchars($cur_forum['forum_name']).'<br /></label>'."\n";
                }

?>
                                </div>
                            </div>
                            <br class="clearb" /><input type="submit" name="update_forums" value="<?php echo $lang_profile['Update forums'] ?>" />
                        </div>
                    </fieldset>
                </div>
<?php
```

For this reason, the requirement is only *partially satisfied*.

1.10

`Verify that there is no sensitive business logic, secret keys or other proprietary information in client side code.`

The requirement is asking to check whether there is an unwanted data-flow of sensible information to the client side code. The analyzed application does not require the client to install any additional software in order to use it properly. This because FluxBB is a web forum that is hosted in a server, accessible by any user's browser. Manually checking inside HTML pages for unwanted sensible data exposure does not produce any result.

For this reason, there is not any sensitive content (related to the application) that appears in the client side code. So the requirement is *satisfied*.

1.11

`Verify that all application components, libraries, modules, frameworks, platform, and operating systems are free from known vulnerabilities.`

A vulnerability is a hole or a weakness in the application, which can be a design flaw or an implementation bug, that allows an attacker to cause harm to the stakeholders of an application. Stakeholders include the application owner, application users, and other entities that rely on the application.

FluxBB has a changelog that is constantly updated by the developers (i.e. Development → Changelog), describing what types of modifications they do in the code. These can be bugs removal, new feature additions, updates of some deprecated functions and fixes of some vulnerabilities that were found.
A good addition that developers did in order to keep users aware of any possible security flaws that may arise in the future, was the creation of some sort of "security mailing-list" used by developers to notify users about new security-relevant releases as fast as possible.

Despite the constant update of the application, we installed and used different tools in order to check whether there were some known vulnerabilities not previously spotted by the developers. The tools that we used are:

- **RIPS** (free version)[3], a static code analysis software for the automated detection of security vulnerabilities in PHP applications;
- **progpilot**, a vulnerability scanner for PHP;
- **VisualCodeGrepper**, is an automated code security review tool that handles PHP, C/C++, Java, C#, VB and SQL;
- **phpcs-security-audit**, is a tool that tokenizes PHP, JavaScript and CSS files and detects violations of a defined set of coding standards.

Surprisingly, they produced an **enormous** amount of warnings and possible errors in the code (e.g. $1000 - 1200$ hits for each tool!), related to multiple types of vulnerabilities such as Cross-Site Scripting (XSS), HTTP Response Splitting and SQL Injection. These static checkers are well-known to produce a large amount of false positives but manually search for them inside the results would be very demanding and time consuming. Thus, the tools were

[3]We contacted RIPS's developers in order to get a free trial of the commercial version but that was denied.

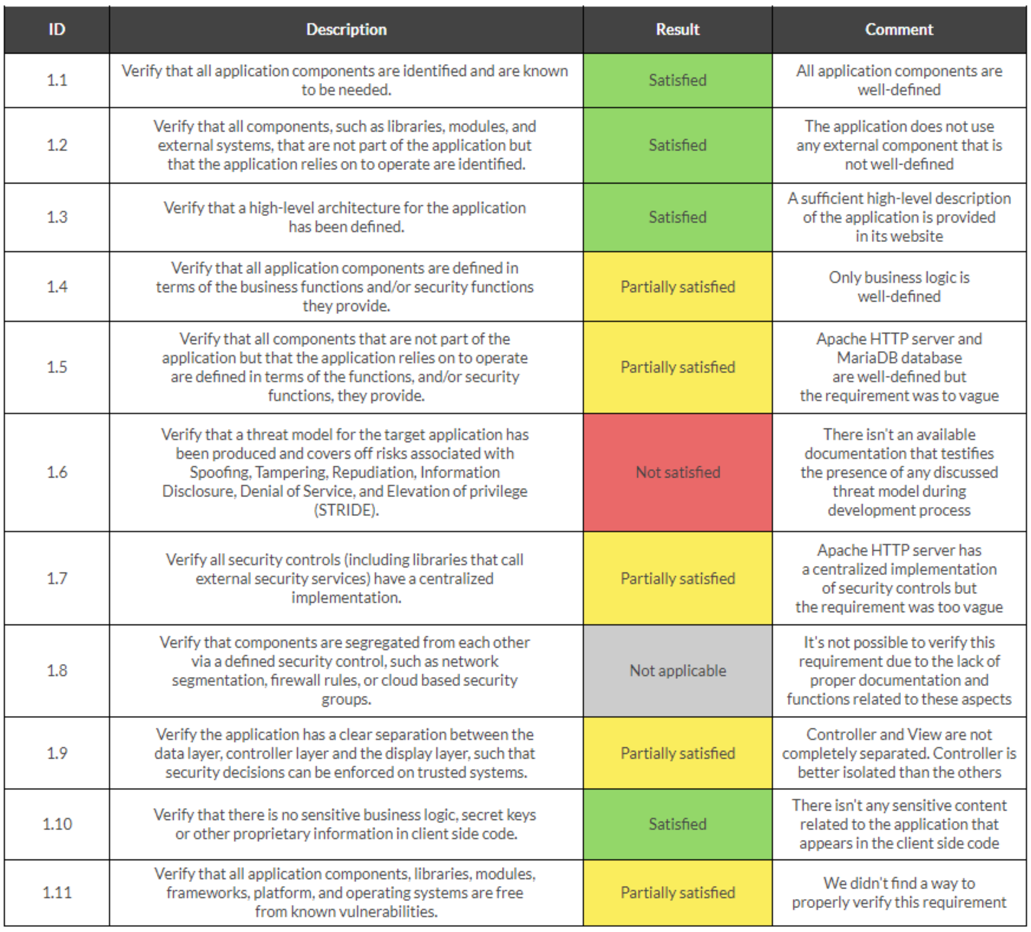
completely useless due to **unmanageable** results.

Additionally, we checked that also extra components (in our case, Apache HTTP Server and MariaDB database) were free from know vulnerabilities. Since these were third parties tools, it was up to their developers to update them periodically. The only thing that FluxBB's system administrator had to do is to keep them up to date.

Since we did not find a suitable way to check in depth whether this requirement is satisfied or not, we decided to mark it as *partially satisfied*.

3.2 Summary

The following table summarizes our considerations about v1 category, with a brief description for each requirement and a concise comment of the result achieved.



ID	Description	Result	Comment
1.1	Verify that all application components are identified and are known to be needed.	Satisfied	All application components are well-defined
1.2	Verify that all components, such as libraries, modules, and external systems, that are not part of the application but that the application relies on to operate are identified.	Satisfied	The application does not use any external component that is not well-defined
1.3	Verify that a high-level architecture for the application has been defined.	Satisfied	A sufficient high-level description of the application is provided in its website
1.4	Verify that all application components are defined in terms of the business functions and/or security functions they provide.	Partially satisfied	Only business logic is well-defined
1.5	Verify that all components that are not part of the application but that the application relies on to operate are defined in terms of the functions, and/or security functions, they provide.	Partially satisfied	Apache HTTP server and MariaDB database are well-defined but the requirement was to vague
1.6	Verify that a threat model for the target application has been produced and covers off risks associated with Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of privilege (STRIDE).	Not satisfied	There isn't an available documentation that testifies the presence of any discussed threat model during development process
1.7	Verify all security controls (including libraries that call external security services) have a centralized implementation.	Partially satisfied	Apache HTTP server has a centralized implementation of security controls but the requirement was too vague
1.8	Verify that components are segregated from each other via a defined security control, such as network segmentation, firewall rules, or cloud based security groups.	Not applicable	It's not possible to verify this requirement due to the lack of proper documentation and functions related to these aspects
1.9	Verify the application has a clear separation between the data layer, controller layer and the display layer, such that security decisions can be enforced on trusted systems.	Partially satisfied	Controller and View are not completely separated. Controller is better isolated than the others
1.10	Verify that there is no sensitive business logic, secret keys or other proprietary information in client side code.	Satisfied	There isn't any sensitive content related to the application that appears in the client side code
1.11	Verify that all application components, libraries, modules, frameworks, platform, and operating systems are free from known vulnerabilities.	Partially satisfied	We didn't find a way to properly verify this requirement

4 v3 Session Management

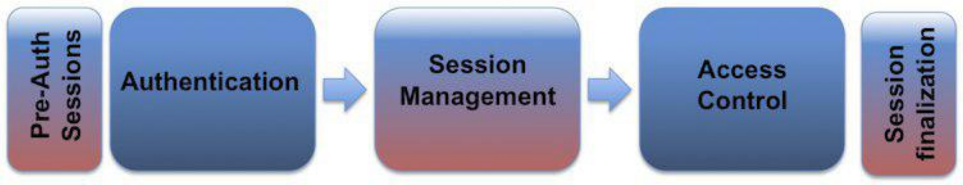
Session Management OWASP category regards the entire mechanism by which any web-based application controls and maintains the state for a user interacting with it. Session Management defines the set of all controls governing state-full interaction between a user and the web-based application.

The **goal** of V3 requirements is to minimize the risk of leaking users' information.

A web session is a sequence of network HTTP request and response transactions associated to the same user. Modern and complex web applications require the retaining of information or status about each user for the duration of multiple requests. Therefore, sessions provide the ability to establish variables – such as access rights and localization settings – which will apply to each and every interaction a user has with the web application for the duration of the session.

Once an authenticated session has been established, the session ID (or token) is temporarily equivalent to the strongest authentication method used by the application.

HTTP is a stateless protocol, where each request and response pair is independent of other web interactions. Therefore, in order to introduce the concept of a session, it is required to implement session management capabilities that link both the authentication and access control (or authorization) modules.



The session ID or token binds the user authentication credentials (in the form of a user session) to the user HTTP traffic and the appropriate access controls enforced by the web application.

The disclosure, capture, prediction, brute force, or fixation of the session ID will lead to session hijacking (or sidejacking) attacks, where an attacker is able to fully impersonate a victim user in the web application.

4.1 Requirements

3.1

`Verify that there is no custom session manager, or that the custom session manager is resistant against all common session management attacks.`

The session manager is the module in charge of securing multiple requests to a service from the same user or entity. A session manager can be either a default built-in save handler declared in `php.ini` configuration file or a custom handler defined by a call to `session_set_save_handler()`. Since this component is in charge of a critical task, implementing a custom session manager may introduce exploitable vulnerabilities.

Using `grep` and searching for session-related keywords inside the code of the application, no custom session manager was found. We discovered that `session` keyword was used few times and not for useful functions. Indeed, the first result of `grep` shows a call to `session.start()` that wraps the default server's session management initialization. Indeed, when `session_start()` is called or when a session automatically starts, PHP will call the session save handlers.

```
session_start();
...
if ($start_at == 0)
   $_SESSION['dupe_users'] = array();
...
```

Since there is no call to `session_set_save_handler()`, the default one is used. The default session manager is well-defined by PHP standard, thus we considered that as resistant against all common session management attacks.

The requirement is *satisfied.*

3.2

`Verify that sessions are invalidated when the user logs out.`

Even after the session has been closed, it might be possible to access the private or sensitive data exchanged within the session through the web browser cache. Therefore, web applications must use restrictive cache directives for all the web traffic exchanged through HTTP and HTTPS on all or (at least) sensitive web pages. Independently of the cache policy defined by the web application, if caching web application contents is allowed, the session IDs must never be cached.

According to `grep` there is not any explicit mechanism to invalidate a session. However, testing FluxBB application locally, it invalidates the old session and creates a new one. To test whether the application correctly handles session invalidation, we tried a simple test: once logged out and pressed *Back* in the web browser, we were correctly not logged anymore.

Regardless the positive result of the test, we noticed a possible security flaw: accessing an older page in the web browser history in which we were still logged in, we were able to read sensible information belonging to a previous session. Fortunately, the application denied any attempt to modify disclosed information, prompting the authentication page as soon as we moved in the website from there.

Since there is the possibility to have a sensible information disclosure despite the invalidation of the sessions, the requirement is only *partially satisfied.*

3.3

`Verify that sessions timeout after a specified period of inactivity.`

In order to minimize the time period an attacker can launch attacks over active sessions and hijack them, it is mandatory to set expiration timeouts for every session, establishing the amount of time a session will remain active. Insufficient session expiration by the web application increases the exposure of other session-based attacks, as for the attacker to be able to reuse a valid session ID and hijack the associated session.

The shorter the session interval is, the lesser the time an attacker has to use the valid session ID. The session expiration timeout values must be set accordingly with the purpose and nature of the web application, and balance security and usability, so that the user can comfortably complete the operations within the web application without his session frequently expiring. Both the idle and absolute timeout values are highly dependent on how critical the web application and its data are. Common idle timeouts ranges are 2-5 minutes for high-value applications and 15- 30 minutes for low risk applications.

When a session expires, the web application must take active actions to invalidate the session on both sides, client and server. The latter is the most relevant and mandatory from a security perspective

In order to close and invalidate the session on the server side, it is mandatory for the web application to take action either when the session expires or the user actively logs out, by using the functions and methods offered by PHP session management mechanisms, such as `session_destroy()/unset()`.

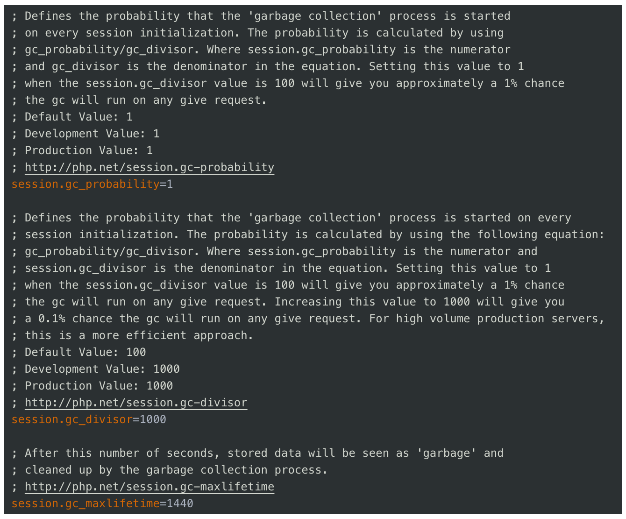
All sessions should implement an idle or inactivity timeout. This timeout defines the amount of time a session will remain active in case there is no activity in the session, closing and invalidating the session upon the defined idle period since the last HTTP request received by the web application for a given session ID.

The idle timeout limits the chances an attacker has to guess and use a valid session ID from another user. However, if the attacker is able to hijack a given session, the idle timeout does not limit the attacker's actions, as he can generate activity on the session periodically to keep the session active for longer periods of time.

Session timeout management and expiration must be enforced server-side. If the client is used to enforce the session timeout, for example using the session token or other client parameters to track time references (e.g. number of minutes since login time), an attacker could manipulate these to extend the session duration.

Considering FluxBB, each time `session_start()` is called the session files' timestamp gets updated. Note that this is used to calculated if `session.gc_maxlifetime` has been exceeded.

`session.gc_maxlifetime` defines the amount of time ($\sim$ 1440 seconds) after which the stored data will be seen as "garbage" and cleaned up by the garbage collection process. Garbage collection may occur during session start, depending on `session.gc_probability` and `session.gc_divisor`.



```
; Defines the probability that the 'garbage collection' process is started
; on every session initialization. The probability is calculated by using
; gc_probability/gc_divisor. Where session.gc_probability is the numerator
; and gc_divisor is the denominator in the equation. Setting this value to 1
; when the session.gc_divisor value is 100 will give you approximately a 1% chance
; the gc will run on any give request.
; Default Value: 1
; Development Value: 1
; Production Value: 1
; http://php.net/session.gc-probability
session.gc_probability=1

; Defines the probability that the 'garbage collection' process is started on every
; session initialization. The probability is calculated by using the following equation:
; gc_probability/gc_divisor. Where session.gc_probability is the numerator and
; session.gc_divisor is the denominator in the equation. Setting this value to 1
; when the session.gc_divisor value is 100 will give you approximately a 1% chance
; the gc will run on any give request. Increasing this value to 1000 will give you
; a 0.1% chance the gc will run on any give request. For high volume production servers,
; this is a more efficient approach.
; Default Value: 100
; Development Value: 1000
; Production Value: 1000
; http://php.net/session.gc-divisor
session.gc_divisor=1000

; After this number of seconds, stored data will be seen as 'garbage' and
; cleaned up by the garbage collection process.
; http://php.net/session.gc-maxlifetime
session.gc_maxlifetime=1440
```

PHP runs the garbage collector on expired sessions after the new session is loaded and only with a certain probability, computed using `session.gc_-`

`probability` and `session.gc_divisor`. The probability is calculated by using `gc_probability/gc_divisor`, (e.g. 1/100 means that there is a 1% chance that the garbage collection process starts on each request). By default, `session.gc_divisor` is set to 100 and `gc_probability` is set to 1.

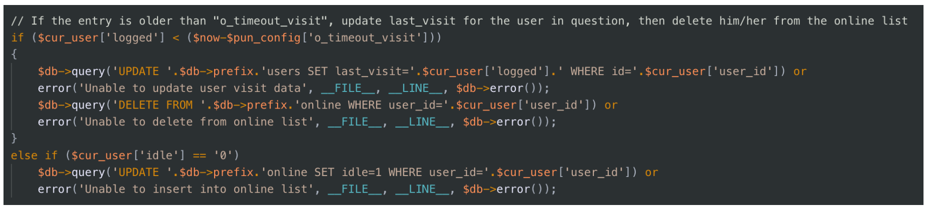
For this reason, the requirement is *satisfied*.

3.4

`Verify that sessions timeout after an administratively configurable maximum time period regardless of activity (an absolute timeout).`

All sessions should implement an absolute timeout, regardless of session activity. This timeout defines the maximum amount of time a session can be active, closing and invalidating the session upon its expiration. After invalidating the session, the user is forced to (re)authenticate again in the web application and establish a new session.
The absolute timeout limits the amount of time an attacker can use a hijacked session and impersonate the victim user.

Considering the examined application, associated to each registered user there is a timer called `o_timeout_visit` that is stored in `cache_config.php`. The timer determines the amount of time (expressed in seconds) that a user can remain logged into the application. After its expiration, the user must authenticate again if he wants to continue the browsing. The default value of `o_timeout_visit` is 1800.



```
// If the entry is older than "o_timeout_visit", update last_visit for the user in question, then delete him/her from the online list
if ($cur_user['logged'] < ($now-$pun_config['o_timeout_visit']))
{
    $db->query('UPDATE '.$db->prefix.'users SET last_visit='.$cur_user['logged'].' WHERE id='.$cur_user['user_id']) or
    error('Unable to update user visit data', __FILE__, __LINE__, $db->error());
    $db->query('DELETE FROM '.$db->prefix.'online WHERE user_id='.$cur_user['user_id']) or
    error('Unable to delete from online list', __FILE__, __LINE__, $db->error());
}
else if ($cur_user['idle'] == '0')
    $db->query('UPDATE '.$db->prefix.'online SET idle=1 WHERE user_id='.$cur_user['user_id']) or
    error('Unable to insert into online list', __FILE__, __LINE__, $db->error());
```

Despite this approach, if the user decides to save the password in the authentication phase, the expiration time changes: the user can remain logged in for a huge amount of time ($\sim$ 336 hours).

For these reasons, the requirement is *satisfied*.

3.5

`Verify that all pages that require authentication have easy and visible access to logout functionality.`

Web applications should provide mechanisms that allow aware users to actively close their session once they have finished using the web application. Web applications must provide a visible an easily accessible logout (logoff, exit, or close session) button that is available on the web application header or menu and reachable from every web application resource and page, so that the user can manually close the session at any time. As described above, the web application must invalidate the session at least on server side. Woefully, not all web applications facilitate users to close their current session.

Fortunately, FluxBB has a *Logout* button that appears in all pages that require authentication.

Thus, the requirement is *satisfied*.

3.6

`Verify that the session id is never disclosed in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies.`

The usage of specific session ID exchange mechanisms, such as those where the ID is included in the URL, might disclose the session ID (in web links and logs, web browser history and bookmarks, the referrer header or search engines), as well as facilitate other attacks, such as the manipulation of the ID or session fixation attacks

A web application should make use of cookies for session ID exchange management. If a user submits a session ID through a different exchange mechanism, such as a URL parameter, the web application should avoid accepting it as part of a defensive strategy to stop session fixation.

The analyzed application does not expose any session in URLs, error messages, or logs. Indeed, manually searching for the keyword "session" in `php.ini` file, among several configuration parameters, the `session.use_trans_sid` defines whether URLs can contain (active) session IDs. By default this functionality is disabled, meaning that the parameter is set to 0. Since it is not possible to inject any session ID in URL, such that they can be interpreted as a valid input, the application does not support URL rewriting of session cookies.

Hence, the requirement is *satisfied*.

3.7

`Verify that all successful authentication and re-authentication generates a new session and session id.`

The session ID must be renewed or regenerated by the web application after any privilege level change within the associated user session. The most common scenario where the session ID regeneration is mandatory is during the authentication process, as the privilege level of the user changes from the unauthenticated (or anonymous) state to the authenticated state. Other common scenarios must also be considered, such as password changes, permission changes or switching from a regular user role to an administrator role within the web application. For all these web application critical pages, previous session IDs have to be ignored, a new session ID must be assigned to every new request received for the critical resource, and the old or previous session ID must be destroyed.

The session ID regeneration is mandatory to prevent session fixation attacks, where an attacker sets the session ID on the victim users web browser instead of directly gathering them, as in most of the other session-based attacks (independently from using HTTP or HTTPS). This protection mitigates the impact of other web-based vulnerabilities that can also be used to launch session fixation attacks, such as HTTP response splitting or XSS.

In `db_update.php` the user ID, used to identify the session, is generated only if the session is valid.

The function that generates the ID is:

```
$uid = pun_hash($req_db_pass.'|'.uniqid(rand(), true));
```

For this reason, the requirement is *satisfied*.

3.10

`Verify that only session ids generated by the application framework are recognized as active by the application.`

Since PHP 5.5.2, `session.use_strict_mode` is available. When it is enabled (and the session save handler supports it), an uninitialized session ID is rejected and a new session ID is created. This prevents an attack that may force users to use known session IDs.
Attacker may paste the session ID in URL or send an email that contains the session ID (e.g. `http://example.com/page.php?PHPSESSID=123456789`).
If `session.use_trans_sid` is enabled, the victim may start a session using the

session ID provided by the attacker. To summarize, `session.use_strict_mode` mitigates the risk.

Some application frameworks will accept any old session ID, but this is not the case because it is not possible to insert any ID in the URL in order to create a valid URL.

Hence, the requirement is *satisfied*.

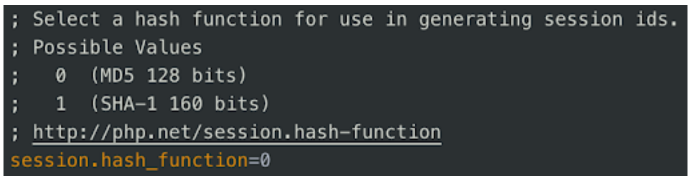
3.11

`Verify that session ids are sufficiently long, random and unique across the correct active session base.`

The session ID must be unpredictable (random enough) to prevent guessing attacks, where an attacker is able to guess or predict the ID of a valid session through statistical analysis techniques. For this purpose, a good PRNG (Pseudo Random Number Generator) must be used.

The session ID value must provide at least 64 bits of entropy: if a good PRNG is used, this value is estimated to be half the length of the session ID. Note that session ID entropy is really affected by other external and difficult to measure factors, such as the number of concurrent active sessions the web application commonly has, the absolute session expiration timeout, the amount of session ID guesses per second the attacker can make, etc. If a session ID with an entropy of 64 bits is used, it will take an attacker at least 292 years to successfully guess a valid session ID, assuming the attacker can try $10,000$ guesses per second with $100,000$ valid simultaneous sessions available in the web application.

In `php.ini` configuration file, the option `session.hash_function` allows to specify the hashing algorithm used to generate session IDs. By default, the parameter is set to 0, so it is used MD5 message-digest algorithm, that produce a 128-bit hash value.



```
; Select a hash function for use in generating session ids.
; Possible Values
;   0  (MD5 128 bits)
;   1  (SHA-1 160 bits)
; http://php.net/session.hash-function
session.hash_function=0
```

This is well-know to be a weak one-way hashing algorithm and cryptographically broken but it can still be considered fine for session IDs generation since they will have a short lifetime. If developers were paranoid enough they would have used SHA-1 hash algorithm, but realistically speaking nobody will ever try to break session ID with 30 minutes expiration timeout especially for the examined application.

Despite this requirement is marked to be an L1 recommendation, OWASP standard was not so precise in describing how 3.11 should be guaranteed. We would like to have seen a kind of list of sub-requirements that should be met according to the area of interest of the application.

For this reason, the requirement is *partially satisfied*.

3.12

`Verify that session ids stored in cookies have their path set to an appropriately restrictive value for the application, and authentication session tokens additionally set the "HttpOnly" and "secure" attributes.`

Considering the `php.ini` file, the path in which cookies are stored is the root file of the application (`session.cookie_path=\`). This means that it is up to the browser how to manage session cookies.

The tests were done using Google Chrome and Mozilla FireFox. The cookies of the application were found in the same folder that contains all the others coming from different sites. Despite this unsecured behavior, we noticed that the `HttpOnly` cookie attribute is enabled by the browser. HttpOnly is an additional flag included in `Set-Cookie` HTTP response header. Using the HttpOnly flag when generating a cookie helps to mitigate the risk of client side script accessing the protected cookie. As a result, even if a cross-site scripting (XSS) flaw exists, and a user accidentally accesses a link that exploits this flaw, the browser will not reveal the cookie to a third party.

Considering `php.ini` configuration file, the `HttpOnly` flag is not set (`false` by default), but FluxBB code overrides this behavior by manually enable the flag, using the following instructions:

```
include/functions.php:

if (version_compare(PHP_VERSION, '5.2.0', '>='))
    setcookie($name, $value, $expire, $cookie_path,
       $cookie_domain, $cookie_secure, true);
else
    setcookie($name, $value, $expire, $cookie_path.';
       HttpOnly', $cookie_domain, $cookie_secure);
```

In addition, `secure` attribute is used in the same operation.

The `secure` cookie attribute instructs web browsers to only send the cookie through an encrypted HTTPS (SSL/TLS) connection. This session protection mechanism is mandatory to prevent the disclosure of the session ID through MitM (Man-in-the-Middle) attacks. It ensures that an attacker cannot simply capture the session ID from web browser traffic.

Forcing the web application to only use HTTPS for its communication (even when port TCP/80, HTTP, is closed in the web application host) does not protect against session ID disclosure if the `secure` cookie has not been set. The attacker can intercept and manipulate the victim user traffic and inject an HTTP unencrypted reference to the web application that will force the web browser to submit the session ID in the clear.

Unfortunately the application set the `cookie_secure` attribute to 0, leaving it as default value.

```
config.php:

$cookie_secure = 0;
```

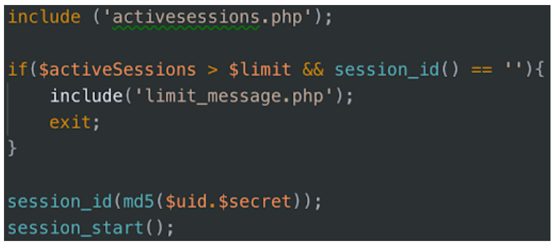
Hence, the requirement is *not satisfied*.

3.16

`Verify that the application limits the number of active concurrent sessions.`

Limiting the number of concurrent session is fundamental for security reasons, such as DoS attack avoidance.

A common way to limit this is to keep track of the number of active sessions in the system, comparing it with the maximum number of admissible sessions. An example of this mechanism can be the following:



```
include ('activesessions.php');

if($activeSessions > $limit && session_id() == ''){
    include('limit_message.php');
    exit;
}

session_id(md5($uid.$secret));
session_start();
```

Woefully, searching for keywords like `active`, `limit`, `concurrent`, gave no interesting result. Hence, we have assumed that there is no limit of concurrent sessions at application level, but there are some *physical restrictions*. The random id generated for each session is of a fixed length, ergo there are only a limited number of random ids available. But that number is ridiculously large, and will likely by far exceed the number of files which can be stored within a single directory, which is limited by the filesystem. Since all sessions are stored as files by default in a single directory, eventually the limit will occur. Another restriction is given by the limited size of the disk on which the session data is stored.

For all these reasons, the requirement is *not satisfied.*

3.17

`Verify that an active session list is displayed in the account profile or similar of each user. The user should be able to terminate any active session.`

It is recommended for web applications to add user capabilities that allow checking the details of active sessions at any time, monitor and alert the user about concurrent logons, provide user features to remotely terminate sessions manually, and track account activity history (logbook) by recording multiple client details such as IP address, User-Agent, login date and time, idle time, etc.

Regrettably, as a user, it is impossible to access these information through the web site's offered services. Thus, the requirement is *not satisfied.*

3.18

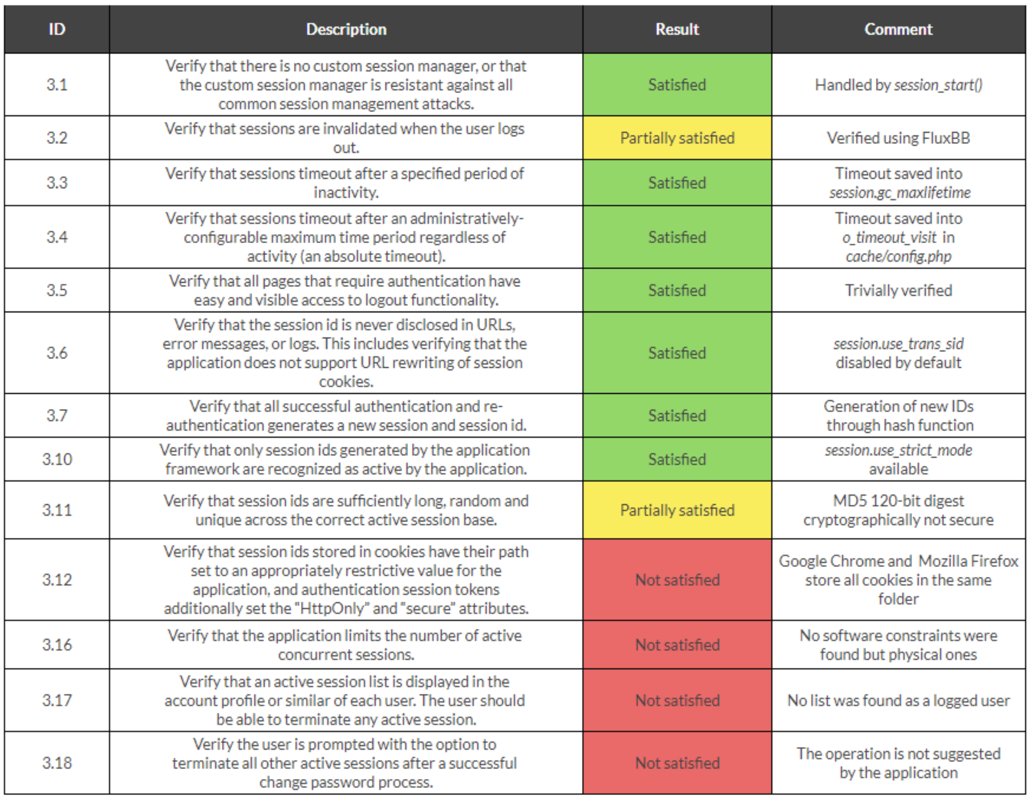
`Verify the user is prompted with the option to terminate all other active sessions after a successful change password process.`

Once the password is changed, all the pages in which the user is logged in do not force the same user to authenticate again.
Running the application on different pages with the same logged user, it is possible to perform sensible operations on data (e.g. changing password, changing email, etc.) without oblige the user to log in again.

Hence, the requirement is *not satisfied.*

4.2 Summary

The following table summarizes our consideration about v3 category.



ID	Description	Result	Comment
3.1	Verify that there is no custom session manager, or that the custom session manager is resistant against all common session management attacks.	Satisfied	Handled by *session_start()*
3.2	Verify that sessions are invalidated when the user logs out.	Partially satisfied	Verified using FluxBB
3.3	Verify that sessions timeout after a specified period of inactivity.	Satisfied	Timeout saved into *session.gc_maxlifetime*
3.4	Verify that sessions timeout after an administratively-configurable maximum time period regardless of activity (an absolute timeout).	Satisfied	Timeout saved into *o_timeout_visit* in *cache/config.php*
3.5	Verify that all pages that require authentication have easy and visible access to logout functionality.	Satisfied	Trivially verified
3.6	Verify that the session id is never disclosed in URLs, error messages, or logs. This includes verifying that the application does not support URL rewriting of session cookies.	Satisfied	*session.use_trans_sid* disabled by default
3.7	Verify that all successful authentication and re-authentication generates a new session and session id.	Satisfied	Generation of new IDs through hash function
3.10	Verify that only session ids generated by the application framework are recognized as active by the application.	Satisfied	*session.use_strict_mode* available
3.11	Verify that session ids are sufficiently long, random and unique across the correct active session base.	Partially satisfied	MD5 120-bit digest cryptographically not secure
3.12	Verify that session ids stored in cookies have their path set to an appropriately restrictive value for the application, and authentication session tokens additionally set the "HttpOnly" and "secure" attributes.	Not satisfied	Google Chrome and Mozilla Firefox store all cookies in the same folder
3.16	Verify that the application limits the number of active concurrent sessions.	Not satisfied	No software constraints were found but physical ones
3.17	Verify that an active session list is displayed in the account profile or similar of each user. The user should be able to terminate any active session.	Not satisfied	No list was found as a logged user
3.18	Verify the user is prompted with the option to terminate all other active sessions after a successful change password process.	Not satisfied	The operation is not suggested by the application

5 Conclusions

Despite some problems were related to our inexperience with PHP language, applying OWASP principles in order to check whether some requirements were fulfilled of not by FluxBB application was not an easy task. In our opinion, the main obstacles that we faced were related to three **limitations**:

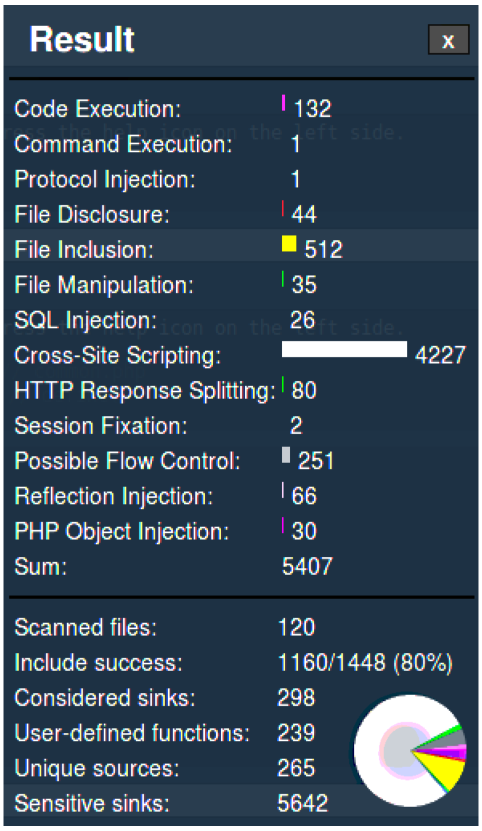
1. most of the OWASP principles were not so precise in describing what should be checked, resulting to be too vague for our need and knowledge of the examined application, especially the ones related to V1 category. We would like to have seen some kind of list of sub-requirements, categorizing what should be checked in base of the area of interest of the examined application. An additional improvement would be to further divide the requirements according to the programming language used by the analyzed application. This would be useful since some features can be worth to be checked only if some particular language is used. All of these improvements would help a lot in the analysis, especially for some analysts that did not take part in the development of the application taken into exam;

2. some of the OWASP requirements should be split in multiple parts. Sometimes, a requirement was partially satisfied only because it asked to check different things that were not complete verified. Divide them would help to reach a finer analysis and more comprehensive results;

3. sometimes, applying ASVS Level 3 requirements to FluxBB application seems to be a bit excessive. Despite the necessity to do that in order to fulfill our project assignment, we think that that some requirements should be considered only for very critical applications (i.e. health-care or military ones) instead for an application like FluxBB that can be classified into Level 2 category: an application that contains sensitive data which requires protection.

As anticipated throughout the entire analysis, most of the **tools** were useless. Static checkers produced a huge amount of warnings that cannot be used to find some possible flaws without investing a lot of energies in filtering thousands (and thousands) of false positives.

One of the tools that has been used during the analysis is *RIPS*, but, as can be seen from the image below, the number of warnings is exaggeratedly high.



Result

```
Code Execution:            132
Command Execution:         1
Protocol Injection:        1
File Disclosure:           44
File Inclusion:            512
File Manipulation:         35
SQL Injection:             26
Cross-Site Scripting:      4227
HTTP Response Splitting:   80
Session Fixation:          2
Possible Flow Control:     251
Reflection Injection:      66
PHP Object Injection:      30
Sum:                       5407

Scanned files:             120
Include success:           1160/1448 (80%)
Considered sinks:          298
User-defined functions:    239
Unique sources:            265
Sensitive sinks:           5642
```

We tried to use more than one tool to obtain more significant result (to understand if the results were correct or if they were mostly false positives), but we did not get the desired outcome given the unusability of the tools.