Radboud University of Nijmegen

Software Security

OWASP
Open Source Review Project

Arne Aarts (s4667271)
Michail Kapsalakis (s4667700)
Sander Peters (s4677595)
Le Phuoc Tho (s4674936)

1. Organization

Initial scanning was performed using the Fortify tool, and performing a scan on every single piece of code. Upon having a brief look at all the error categories as a group, the errors were divided amongst the group members based on what sort of error it was (SQL injection, cross-site scripting, etc.). These errors were then located and double-checked by the group members, and, where relevant, the corresponding checkbox in the OWASP Verification Standard was marked as a Fail.

It was quickly found out that after a round of static analysis with Fortify only about 20% of the checkboxes were filled in. To fill in the rest of them, the decision was made to divide the categories of the OWASP Verification Standard (V2-V9) among the group members. These could then be filled in after a manual code review or through use of the web application. In case of doubt or uncertainty, questionable points would be discussed with the rest of the group, and if no clear consensus could be reached, they could be marked as such on the verification standard.

At times, inconclusive statements were made as a result of sampling. There were situations where it was not feasible to go through all the code to look for the absence of something. In these cases, certain pieces of code were reviewed, and the results of this were (perhaps incorrectly) extrapolated to the entire code.

In the following weeks, group members reviewed each other's verdicts in pairs, to see if there weren't any mistakes or points of disagreement. This led to some minor revisions in the verification standard.

To conclude the project, we worked on this report. In the final weeks, the group members also performed some final revisions on the checklist of verdicts, as well as rewriting them in a more clear and consistent way, making them readable to others. There was no strict division in who wrote what part of the report, people volunteered for parts they wanted to write, and others added onto it and made changes as necessary.

2. Verdict

Verification	Verdict	Explanation
2.1	Pass	The testcms.php parses requested URL and redirect to login page if the request relate to admin actions.
2.2	Pass	Login.php and reset.php use input type for password.
2.4	Pass	All authentications are handled at server side by users.php code.
2.6	Pass	All login failures are safely handled when running the application for checking purpose.
2.7	Pass	Password fields allow everything. They do however not encourage anything.
2.8	Fail	As indicated from Fortify, login.php and reset.php files use autocompletion in the form, which allows browsers to retain information in their history.
2.9	Fail	As indicated from the code of edit.php file, the user can change the password, however there is only one field for the password input and there is no password confirmation field.
2.16	Fail	All communication are transported unencrypted (unless the server is configured with a valid certificate to secure communication via HTTPS).
2.17	Pass	Password recovery is conducted in a way that the user will receive an email to follow a link for password reset.
2.18	Pass	Limited information prompted to input, such as "Incorrect pass", "Account not found". Hence information enumeration is unlikely possible.
2.19	Pass	There is no default passwords involved in the operation. After the installation of TestCMS the default password of database in config.php file has been changed.
2.20	Fail	There is no control to check automatic password attacks such as captcha.
2.22	Pass	The application uses a hash value in the link for password

		reset.
2.24	Fail	There are no secret questions.
2.27	Fail	There are no checks whatsoever on passwords.
2.30	Fail	The authentication process is primitive, it works by comparing input user and password (hash) from database.
2.32	Pass	As 2.1 passed, all admin pages are authentication required, therefore untrusted parties cannot access them.
3.1	Don’t know - Security requirement unclear	It is not clear from the requirement what it is expected to be checked.
3.2	Pass	Cookie is set to expired when logout, hence If the admin logged out, the attacker doesn't have access with the stealing cookies.
3.3	Fail	By the use of the application and code review, particularly in session.php, there is no function to manage timeout and invalidate the session.
3.5	Pass	It is technically a Pass, but there is a problem with the redirection in the front-end pages, because it is in the same tab and the user cannot see the logout functionality.
3.6	Pass	Session id is not accessed by any function to display on URL, error message or logs (only user.php, configuration.php and functions.php gets session value). Furthermore, URL rewrite depends on hosting environment (rewriting rule in Apache for example).
3.7	Fail	The web app doesn't generate a new session_id after the authentication.
3.11	NA - check is beyond scope of code	The length of session_id depends on the php.ini's parameter session.hash_bits_per_character, which is on the server-side.
3.12	Fail	With the use of Fortify and review of the code, it is obvious that the HttpOnly and Secure Attributes are not set.
3.16	NA - other	The requirement is not relevant to the content management

		web application.
3.17	NA-other	This requirement is not relevant to this web application. Maybe, only the administrator would be able to see the active session list.
3.18	Fail	User changes his password and remains connected to the web application.
4.1	Fail	In the web application a “User” account has the same privileges with an “Administrator” account. A “User” can also delete the account of an “Administrator”.
4.4	Don’t know - Other	It does not seem intended for a user to access all other information in the system, yet the system allows this. All sensitive data is however only available in the admin area which is protected by sessions, so a guest could not see any data of a user.
4.5	Fail	The default htaccess file does not forbid directory browsing.
4.8	Don't know - not clear how to check this	We did not observe any failing, nor were we able to find where this happens so we don’t know if it fails securely.
4.9	Trivial pass	There is no access control client side.
4.13	Fail	The website doesn't use CSRF tokens at all.
4.16	Pass	All access control on the admin area seems to be based on cookie sessions.
5.1	NA- check is beyond scope of code	Buffer overflows in PHP are not common, but they are possible. This is a fault of bugs in PHP, and not in the code. However, the application does not do any input verification checks on, for example, length, that might help combat these bugs.
5.3	Fail	Requests are properly denied, but they are not logged.
5.5	Pass	All user input goes through a cleaning function in input.php.
5.10	Fail	As indicated by Fortify, db.php and installer.php execute an SQL query built from untrusted input without performing and validation or input sanitizing.

5.11	Trivial pass	LDAP is not used.
5.12	Trivial pass	There is no direct access to the OS command line.
5.13	Pass	The template.php class parses a file path, but this does not use data from any LFI-vulnerable functions, so it is not a problem.
5.14	Don’t Know - Other	The rss.php class makes an XML DOM tree. It does not seem to be injectable, but our knowledge of XML injection is not enough to know if this is really true.
5.15	Fail	The input/output does not get properly validated according to fortify. This is a systematic fault that happens throughout the project.
5.22	Fail	system/classes/post/php does no sanitization checks on the posted or edited HTML.
7.2	Pass	The program uses crypt(), which is based on DES. DES is not vulnerable to oracle padding.
7.7	Fail	The program uses PHP's default crypt() function in installer.php and users.php, which uses DES, considered a very insecure algorithm. They are also not salted.
8.1	Pass	The amount of information given when an error occurs is settable in an option file, the default setting is to return an 500 error page back. Thus by default no information can leak through errors.
9.1	Pass	We were unable to find any forms that cache.
9.3	Pass	Sampling did not find any URL parameters used for sensitive input.
9.4	Pass	The cache settings use no-store by default, though there is no specification for this in the PHP files themselves.
9.9	Pass	Only cookies are used for session management. The cookies are randomized, and they do not contain any sensitive data.

3. Reflection

In general, the ASVS list was well-understood. We recognize its usefulness, as it is quite exhaustive and, in most cases, clear. However, we found certain exceptions in which the ASVS was confusing. One of such verifications is 3.16 which does not clearly indicate whether the numbers of site visitors should be limited or once a user log in, there should not be another log in with the same credential. We were confused by 3.1, 3.16 and 4.4. This is an inevitable result due to the restrictions of natural language, which may sometimes be interpreted differently by different people. Hence making a clear reference context for each verification could be an enhancement for ASVS.

We found the usefulness of Fortify and RIPS to be lacking with regards to OWASP. While it is useful to get a good first glance of vulnerabilities in the web application with minimal effort, it was not especially useful with regards to ticking things off on the checklist. There is no integration between the ASVS and these static analysis tools, and as a whole the checkmarks gathered from these tools covered less than 20% of the ASVS list.

Although the time investment required for using these tools is very small, they still don't seem to be especially useful for this purpose. A static analysis tool specifically developed for security standard verification would be more helpful. Though this will most likely increase the complexity of the tool and increase the amount of false positives. Thus reducing the usability of said tool. Therefore, a manual code review will probably remain inevitable.

We experienced the manual code review to be the most difficult and time-intensive part. This was partially as a result of inexperience, as many of us had little to no experience with PHP, and of course none of us had ever seen this particular web application before. It certainly was the bottleneck, as we spent by far the most time on it. Finding out how the pieces of code worked and interlinked was a difficult pursuit and often required different team members to look at it together. It would also have helped a lot if we had some more documentation for the web application, such as a class diagram or other design documents, so we'd have a better understanding of the underlying software architecture.

The security review process itself could also be improved, as simply dividing people among different chapters of the ASVS checklist proved to be fairly arbitrary and not that efficient. A better strategy might have been to divide the code instead. This way the reviewer only needs to concern itself with that piece of code and which points could be applicable. This approach would most likely scale better since the amount of requirements in the ASVS is static. We think it might be even better to integrating periodic security reviews during development. This way compliance

with the ASVS can be documented piecemeal, hopefully reducing the complexity of the task. And might provide useful documentation for later stages of the applications life cycle.

Finally, we suggest the following improvements to TestCMS code: All points that failed the ASVS need to be fixed. Most important of which are the missing permission management system and any failures that directly relate to the top 10 security holes: Injection, weak session management, XSS and CSRF. It is important for there to be a good collaboration between the designer, developer and code reviewer to improve efficiency and accuracy. The code has to be more secure in order to avoid the malicious users to attack the application.