Software Security

Project 2

Dion van de Vooren - s4256468
Natanael Adityasatria - s4417992
Sesaria Kikitamara - s4561414
Sudhakarreddy Gottam - s4657454
Sven Arissen - 4206363

Organisation

We managed to work on this project while sitting together every week. This made it easy to organize who did what. We split the work by the ASVS security requirements. We had an online spreadsheet, in which all requirements were listed, such that we had a nice overview of our progress and were not doing double work. All vulnerabilities were stored in an extra spreadsheet, including a severity column. Because we all worked together in one room, it was very easy to communicate severe vulnerabilities.
At the beginning, each of ASVS security requirements is done by one person as the first assigned. Then, another person as the second assigned will check what the first person did. In this way, we did a double-check on every ASVS security requirements.
The tools we used are Fortify and RIPS, but RIPS was not that useful, because it gave many false positives.
We did run the application, which was very useful for understanding how the code works, but also for finding vulnerabilities. In some cases, running the application was more helpful than reviewing the code.

Verdict

V2. Authentication

#	Description	Verdict
2.1	Verify all pages and resources by default require authentication except those specifically intended to be public (Principle of complete mediation)	FAIL, all files except for the files in the system folder are publicly availible
2.2	Verify that all password fields do not echo the user's password when it is entered	FAIL, the generated admin password is echoed to the user at the end of the installation of the cms
2.4	Verify all authentication controls are enforced on the server side	PASS
2.6	Verify all authentication controls fail securely to ensure attackers cannot log in	FAIL, login.php sends unvalidated data to a web browser, which can result in the browser executing malicious code
2.7	Verify password entry fields allow, or encourage, the use of passphrases, and do not prevent long passphrases/highly complex passwords being entered	PASS

2.8	Verify all account identity authentication functions (such as update profile, forgot password, disabled / lost token, help desk or IVR) that might regain access to the account are at least as resistant to attack as the primary authentication mechanism	FAIL, any user can change the password of any other user. Apart from that, the “forgot password” function is as secure as the primary identity authentication, as long as the email address belongs to owner of the account. This however only holds if no user can change the email address of an other user, but that does not hold at the moment.
2.9	Verify that the changing password functionality includes the old password, the new password, and a password confirmation	FAIL, The user only needs to type the new password
2.12	Verify that all suspicious authentication decisions are logged. This should include requests with relevant metadata needed for security investigations.	FAIL, only errors can be logged, but is deactivated by default
2.13	Verify that account passwords make use of a sufficient strength encryption routine and that it withstands brute force attack against the encryption routine.	FAIL, they use MD5 for password encryption which is not FIPS approved and they use the password as a salt, which makes the salt completely useless as a defense against brute-force attacks
2.16	Verify that credentials are transported using a suitable encrypted link and that all pages/functions that require a user to enter credentials are done so using an encrypted link	NOT RELEVANT, a ssl certificate should be mostly configured in the webserver
2.17	Verify that the forgotten password function and other recovery paths do not reveal the current password and that the new password is not sent in clear text to the user	PASS
2.18	Verify that information enumeration is not possible via login, password reset, or forgot account functionality	PASS
2.19	Verify there are no default passwords in use for the application framework or any components used by the application (such as “admin/password”)	PASS
2.20	Verify that request throttling is in place to prevent automated attacks against common authentication attacks such as	FAIL, there is no throttling, however this could also be done on a webserver level

	brute force attacks or denial of service attacks	
2.21	Verify that all authentication credentials for accessing services external to the application are encrypted and stored in a protected location.	FAIL, there is no location to stored the authentication for external services
2.22	Verify that forgotten password and other recovery paths use a soft token, mobile push, or an offline recovery mechanism	FAIL, there is no reset password token
2.23	Verify that account lock out is divided into soft and hard lock status, and these are not mutually exclusive. If an account is temporarily soft locked out due to a brute force attack, this should not reset the hard lock status.	FAIL, there is currently no way to lock a user account
2.24	Verify that if knowledge based questions (also known as "secret questions") are required, the questions should be strong enough to protect the application	NOT RELEVANT, there is no “secret question” feature provided by testcms even if the users forgot their password
2.25	Verify that the system can be configured to disallow the use of a configurable number of previous passwords.	FAIL, using previous passwords is always allowed
2.26	Verify re-authentication, step up or adaptive authentication, two factor authentication, or transaction signing is required before any application-specific sensitive operations are permitted as per the risk profile of the application.	FAIL, a user only needs to be logged in in order to do sensitive operations
2.27	Verify that measures are in place to block the use of commonly chosen passwords and weak passphrases	FAIL, no such measures
2.30	Verify that if an application allows users to authenticate, they use a proven secure authentication mechanism	FAIL, It is used salt technique on password field. So the attacker can simply use the known salt when attempting to crack the password
2.31	Verify that if an application allows usersto authenticate,they can authenticate using two-factor authentication or other strong authentication, or any similarscheme	FAIL, strong authentication is not supported

	that provides protection against username + password disclosure.	
2.32	Verify that administrative interfaces are not accessible to untrusted parties	FAIL, every user has access to the admin pages that edit a user or the website

V3. Session management

#	Description	Verdict
3.1	Verify that there is no custom session manager, or that the custom session manager is resistant against all common session management attacks.	PASS, a pass for now, however there is a comment in the code about using a custom session manager in the future
3.2	Verify that sessions are invalidated when the user logs out.	FAIL, only the user property of the session is removed from the session, but the session is not invalidated
3.3	Verify that sessions timeout after a specified period of inactivity.	DEPENDS ON CONFIG, there is no explicit timeout, but since the php session manager is used, the timeout in php.ini will be used this is normally set to about 20 minutes but can be changed
3.5	Verify that all pages that require authentication have easy and visible access to logout functionality.	PASS
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.	PASS
3.7	Verify that all successful authentication and re-authentication generates a new session and session id.	FAIL, no new session is created, the existing session is used
3.10	Verify that only session ids generated by the application framework are recognized as active by the application.	CONFIG, this depends on whether use_strict_mode is on or off in the php.ini
3.11	Verify that session ids are sufficiently long, random and unique across the correct active session base.	PASS
3.12	Verify that session ids stored in cookies	FAIL, no httponly cookies, the domain is set

	have their path set to an appropriately restrictive value for the application, and authentication session tokens additionally set the “HttpOnly” and “secure” attributes	(should be empty), and the path is very unrestrictive
3.16	Verify that the application limits the number of active concurrent sessions.	FAIL, no such limit set by the application
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.	FAIL, no active session list
3.18	Verify the user is prompted with the option to terminate all other active sessions after a successful change password process.	FAIL, no such prompt

V4. Access control

#	Description	Verdict
4.1	Verify that the principle of least privilege exists - users should only be able to access functions, data files, URLs, controllers, services, and other resources, for which they possess specific authorization. This implies protection againstspoofing and elevation of privilege	FAIL, when a user is logged in he can access all files. Not logged in users can also access files (see 2.1). A normal user can also start the upgrade function by going to /upgrade
4.4	Verify that access to sensitive records is protected, such that only authorized objects or data is accessible to each user (for example, protect against users tampering with a parameter to see or alter another user's account).	NOT RELEVANT, every user has as much rights as an administrator. If they would fix that it wouldbe a PASS, because every request contains a session id
4.5	Verify that directory browsing is disabled unless deliberately desired. Additionally, applications should not allow discovery or disclosure of file or directory metadata, such as Thumbs.db, .DS_Store, .git or .svn folders.	FAIL, everyone can browse through directories
4.8	Verify that access controls fail securely.	PASS, If you want to access the admin controllers without Login, its redirected to Login page to enter admin userid and password

4.9	Verify that the same access control rules implied by the presentation layer are enforced on the server side.	NOT RELEVANT, there are no access control rules on the presentation layer
4.10	Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.	FAIL, the only two things that can give access to an account are e-mail address and password. Both can be changed by any other user, given that he has an account
4.12	Verify that all access control decisions can be logged and all failed decisions are logged.	PASS, it creates error.log
4.13	Verify that the application orframework usesstrong randomanti-CSRF tokens or has another transaction protection mechanism.	FAIL, no transaction protection mechanisms used
4.14	Verify the system can protect against aggregate or continuous access of secured functions, resources, or data. For example, consider the use of a resource governor to limit the number of edits per hour or to prevent the entire database from being scraped by an individual user.	FAIL, there is for example no limit for how many times a user can change his password
4.15	Verify the application has additional authorization (such as step up or adaptive authentication) for lower value systems, and / or segregation of duties for high value applications to enforce anti-fraud controls as per the risk of application and past fraud.	FAIL, the application has no implemented adaptive authentication
4.16	Verify that the application correctly enforces context-sensitive authorisation so asto not allow unauthorised manipulation by means of parameter tampering.	PASS, every request contains a session id

V5. Malicious input handling

#	Description	Verdict
5.1	Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows	PASS, php is not susceptible to buffer overflow, there is still a risk with the underlying code of php, but aside from regular patching there is nothing the creator of the testcms can do about that

5.3	Verify that server side input validation failures result in request rejection and are logged	PASS, there is a code log.php which create error log automatically but when trying to test it, for example create new post with incorrect input, it just shows the alert box and not logged,disabled by default
5.5	Verify that input validation routines are enforced on the server side	PASS
5.10	Verify that all SQL queries, HQL, OSQL, NOSQL and stored procedures, calling of stored procedures are protected by the use of prepared statements or query parameterization, and thus not susceptible to SQL injection	FAIL, there is a line of code which executes SQL from untrusted source
5.11	Verify that the application is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection	PASS, LDAP is not used
5.12	Verify that the application is not susceptible to OS Command Injection, or that security controls prevent OS Command Injection	PASS, there is no OS command statement in the source code (Exec, system, passthru, shell_exec, proc_open, pcntl_exec)
5.13	Verify that the application is not susceptible to Remote File Inclusion (RFI) or Local File Inclusion (LFI) when content is used that is a path to a file	NOT RELEVANT, There is no including files mechanism in this application
5.14	Verify that the application is not susceptible to common XML attacks,such as XPath query tampering, XML External Entity attacks, and XML injection attacks	NOT RELEVANT, there is no XML parser mechanism in this application. Its used sql query directly.
5.15	Ensure that all string variables placed into HTML or other web client code is either properly contextually encoded manually, or utilize templates that automatically encode contextually to ensure the application is not susceptible to reflected, stored and DOM Cross-Site Scripting (XSS) attacks.	FAIL, string variable placed into HTML is not encoded contextually
5.16	If the application framework allows automatic mass parameter assignment	PASS, the application only uses mass assignment for the default user when the website is created. If

	(also called automatic variable binding) from the inbound request to a model, verify that security sensitive fields such as “accountBalance”, “role” or “password” are protected from malicious automatic binding.	an attacker manages to alter the data, the admin would notice right away
5.17	Verify that the application has defenses against HTTP parameter pollution attacks, particularly if the application framework makes no distinction about the source of request parameters (GET, POST, cookies, headers, environment, etc.)	PASS
5.18	Verify that client side validation is used as a second line of defense, in addition to server side validation	FAIL, there is no client side validation
5.19	Verify that all input data is validated, not only HTML form fields but all sources of input such as REST calls, query parameters, HTTP headers, cookies, batch files, RSS feeds, etc; using positive validation (whitelisting),then lesserforms of validation such as greylisting (eliminating known bad strings), or rejecting bad inputs (blacklisting).	FAIL, many HTML form fields are not validated, like the login form
5.20	Verify that structured data is strongly typed and validated against a defined schema including allowed characters, length and pattern (e.g. credit card numbers or telephone, or validating that two related fields are reasonable, such as validating suburbs and zip or post codes match).	FAIL, you can enter numbers as your real name. The e-mail address however is being checked for having a correct pattern
5.21	Verify that unstructured data is sanitized to enforce generic safetymeasuressuch as allowed characters and length, and characters potentially harmful in given context should be escaped (e.g. natural names withUnicode or apostrophes,such as ね or O'Hara)	FAIL, characters as ねor こ are not escaped. The character ‘ however gives an internal error
5.22	Make sure untrusted HTML from WYSIWYG editors or similar are properly sanitized with an HTML	FAIL, HTML input is not filtered in the posts section, one can enter <script>alert(document.cookie);</script> and see

	sanitizer and handle it appropriately according to the input validation task and encoding task	the session id. The search bar is not affected, because certain characters are escaped
5.23	For auto-escaping template technology, if UI escaping is disabled, ensure thatHTML sanitization is enabled instead.	NOT RELEVANT, there is no auto-escaping template technology is implemented
5.24	Verify that data transferred from one DOM context to another, uses safe JavaScript methods, such as using .innerText and .val.	NOT RELEVANT, there is no data transferred from one DOM context to another
5.25	Verify when parsing JSON in browsers, that JSON.parse is used to parse JSONon the client. Do not use eval() to parse JSON on the client.	PASS, there is no eval() used nor JSON.parse
5.26	Verify that authenticated data is cleared from client storage, such as the browser DOM, after the session is terminated.	PASS, no authentication data is saved

V7. Cryptography at rest

#	Description	Verdict
7.2	Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable oracle padding.	PASS
7.6	Verify that all random numbers, random file names, random GUIDs, and random strings are generated using the cryptographic module's approved random number generator when these random values are intended to be not guessable by an attacker.	FAIL, the program makes used of PHP's mt_rand, which is used for genarating the site's password and is not save enough (given the state of the generator, the password can be guessed). Furthermore it uses math.random, which also is not a cryptographically-secure random number generator
7.7	Verify that cryptographic algorithms used by the application have been validated against FIPS 140-2 or an equivalent standard.	FAIL, The algorithm used for hashing passwords is MD5 which is not FIPS 140-2 approved
7.9	Verify that there is an explicit policy for how cryptographic keys are managed (e.g., generated, distributed,revoked, and expired). Verify that this key lifecycle is properly enforced.	FAIL, only session management uses keys, and has no properly enforced key lifecycle

7.12	Personally Identifiable Information should be stored encrypted at rest and ensure that communication goes via protected channels.	FAIL, only the password is stored hashed (not securely, see 2.30)
7.13	Verify that where possible, keys and secrets are zeroed when destroyed.	FAIL, for reauthentication the old session key is not destroyed.
7.14	Verify that all keys and passwords are replaceable, and are generated orreplaced at installation time	PASS, there is no default password and all passwords can be changed later

V8. Error handling and logging

#	Description	Verdict
8.1	Verify that the application does not output errormessages orstack traces containing sensitive data that could assist an attacker, including session id,software/framework versions and personal information	FAIL, contains sensitive information, for example: PHP version, server information, OS version, file path, and request URI.
8.2	Verify that error handling logic in security controls denies access by default.	DON'T KNOW, there is curl.php which called in functions.php, but not sure if it is also used for error handling or not.
8.3	Verify security logging controls provide the ability to log success and particularly failure events that are identified as security-relevant.	FAIL, only errors can be logged, but is deactivated by default. Furthermore exeptions that deal with security are not logged, it are mainly navigation errors
8.4	Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens.	PASS, only errors logged events infomation (date)stored, but is deactivated by default
8.6	Verify that security logs are protected from unauthorized access and modification.	PASS, The log file(\system\classes\log.php) acces by autorized only
8.7	Verify that the application does not log sensitive data as defined underlocal privacy laws or regulations, organizational sensitive data as defined by a risk assessment, or sensitive authentication data that could assist an attacker, including user's session identifiers, passwords, hashes, or API tokens.	PASS, only errors can be logged, but is deactivated by default
8.10	Verify that an audit log or similar allows for non-repudiation of key transactions.	NOT RELEVANT, there are no such

		transactions

V9. Data protection

#	Description	Verdict
9.1	Verify that all forms containing sensitive information have disabled client side caching, including autocomplete features	PASS, contains no-store, no-cache, must-revalidate, post-check=0, pre-check=0. Autocomplete is off for sensitive information
9.3	Verify that all sensitive data is sent to the server in the HTTP message body or headers (i.e., URL parameters are never used to send sensitive data)	PASS, for sensitive data POST is used
9.4	Verify that the application sets appropriate anti-caching headers as per the risk of the application, such as the following: Expires: Tue, 03 Jul 2001 06:00:00 GMT Last-Modified: {now} GMT Cache-Control: no-store, no-cache, must-revalidate, max-age=0 Cache-Control: post-check=0, pre-check=0 Pragma: no-cache	PASS, contains no-store, no-cache, must-revalidate, post-check=0, pre-check=0 and expires after browser session
9.5	Verify that on the server, all cached ortemporary copies ofsensitive data stored are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data.	CONFIG/FAIL, the main sensitive data is the database data, the access of this database is mostly determined by the configuration of the server and mysql. The only glaring mistake is that the install folder should probably be automatically purged after installation, this has to be done manually now
9.7	Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values.	PASS
9.9	Verify that data stored in client side storage - such as HTML5 local storage, session storage, IndexedDB, regular cookies or Flash cookies - does not contain sensitive or PII)	PASS, regular cookies only contain sessid, there is no other data stored locally.
9.10	Verify accessing sensitive data is logged, if the data is collected under relevant data	FAIL, accessing sensitive data is not logged

	protection directives or where logging of accesses is required.	
9.11	Verify that sensitive data is rapidly sanitized from memory as soon as it is no longer needed and handled in accordance to functions and techniques supported by the framework/library/operating system.	FAIL, after login the username and password variables are unset, but not cleared from physical memory

Reflection

The ASVS

The ASVS is a useful standart to test the application security requirements.
Some verification requirements easier to check than the others. It is because some of them are already given some explanation about the standart in OWASP documentation and the others are not. Therefore, the ASVS document can be improved by giving some relevant explanations in the documentation about the standart item and put the reference documentation in ASVS document. In that way, the developers / testers can easily see the more detail information on standart item if they are not understand.

The Tools

We did most of the checking manually through looking directly in the code after running the application. However, doing the manual checks requires not only advance knowledge in php programming but also in finding security vulnerabilities. Therefore, in order to support the manual code review, we used the tools (Fortify and RISP) and found some more vulnerabilities. Fortify is very useful, but on the other hand, RISP gave many false negatives. We think RISP has to be improved a lot especially in reducing false negatives and create better software security intelligence report.

The process

In our experience, the difficulties in doing the security review are checking a whole codes againts one verification requirement item.
We think the best approach to do the security review is by dividing the verification requirements over the team members and working in pairs where one person confirms the findings of the other. In that way, all team members are working with a focus on their own work instead of all team members working on the same thing. In addition, by working in pairs, we make sure that every verification requirements are at least completed by two person.
If we will develop an application that will need to be subjected to a security review, we will make sure that every developers are understand the application security standart. So, the developers can produce the code that conforms to the security standart. It means we were implementing the security standart from the beginning of the application development.

Furthermore, we will conduct a security review in every development phases, so every vulnerabilities found in the beginning of the development can be fixed earlier.

The TestCMS code

There are two things that should be changed in order to make TestCMS usable. First of all, not all users should be able to see or edit data from other users. The second thing is, that more input should be validated. TestCMS as it is now accepts scripts in the forum, which executes when a user goes to that page. This makes it very easy for an attacker to retrieve a session cookie.

Appendix: vulnerabilities

We have a list of vulnerabilities found during the security review. Please see the attached spreadsheet file.