Software Security
Web application audit

Tim Janssen Gerdriaan Mulder Thom Wiggers

January 2015

Chapter 1

Organisation

We divided the initial work with regards to the audit itself amongst four people. We decided to divide the work into the ASVS chapters instead of groups of files or folders. This allowed us to think about the possible security vulnerabilities from a certain 'attacker' viewpoint. When dividing the chapters over the team members we were taking into account the knowledge of the various ASVS chapters per person. The various other chapters were divided ad-hoc. We used ShareLaTeX as a collaboration tool to create the final report.

We used the static analysis tools Fortify, RIPS and RATS to review the source code. Next to that, we looked through the source code ourselves to spot (obvious) security flaws. The manual check was done on individual basis. When anyone of the group had doubts regarding the verdict of a requirement, he would ask another person of the group to have a look as well.

The three static code analysers were all used. Results from some of those were used in determining whether a security requirement was satisfied. In the Reflection chapter, we explain why we think that these tools did not really contribute to the audit.

We used Docker[1] to run the application in a *container* such that existing configuration on our systems was not contaminated. In the Reflection chapter, there are more details about this system.

During the course, we were notified that one of our group members, Matthijs Lavrijsen, was ill and could not complete the project with us. We did not include him in the authors list, because we had to redistribute his ASVS chapters quite early in the project. He has, however, contributed to (parts of) some ASVS chapters in this project. We would like to thank him for that, and wish him a speedy recovery.

[1]`https://www.docker.com/what-docker`, "Docker allows you to package an application with all of its dependencies into a standardized unit for software development."

Chapter 2

Verdict

This chapter presents the results of our code review based on the OWASP requirements. We adapted our template from Jakob Bleier, Moritz Neikes, Martijn Terpstra and Patrick Verleg who took this course last year. The results are our own.

For each requirement, we will indicate whether the requirement is met by adding either a ✗ FAILED or a ✓ PASSED tag. A requirement is passed if both, we and the tools, could not find a violation of the requirement. If either we or the tools found a violation, the final verdict is 'failed'.

In case that a requirement is not met, we will also indicate by which method we spotted the vulnerability. For example, if Fortify found a vulnerability, we will indicate this with the FORTIFY tag. Finally, translucent tags indicate that no vulnerability was spotted with this tool.

Chapter 2: Authentication Verification Requirements

2.1 ✓ PASSED 👁 MANUAL FORTIFY RATS RIPS	*Verify all pages and resources require authentication except those specifically intended to be public.* All admin pages require authorization, and there are no other non-public pages.
2.2 ✗ FAILED 👁 MANUAL 👁 FORTIFY RATS RIPS	*Verify all password fields do not echo the user's password when it is entered.* The reset password field echoes the password back if it for some reason does not pass validation.
2.7 ✓ PASSED 👁 MANUAL FORTIFY RATS RIPS	*Verify that password entry fields allow, or encourage, the use of passphrases, and do not prevent long passphrases/highly complex passwords being entered.* They do allow long passwords with special characters to be entered, but do not encourage it.
2.8 ✗ FAILED 👁 MANUAL FORTIFY RATS RIPS	*Verify that all account identify 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.* There are no resistances against attacks (nor are there on passwords).
2.9 ✗ FAILED 👁 MANUAL FORTIFY RATS RIPS	*Verify that password changing functionality includes the old password, the new password, and a password confirmation* No password confirmation or old password is requested, ever.

2.12

✖ FAILED
MANUAL
FORTIFY
RATS
RIPS

Verify that all suspicious authentication decisions are logged. This should include requests with relevant metadata needed for security investigations.
While there is a log, it is not used in the authentication module.

2.13

✖ FAILED
MANUAL
FORTIFY
RATS
RIPS

Verify that account passwords make use of a sufficient strength encryption routine and that it withstands brute force attack against the encryption routine
The application uses unsalted `crypt()` (DES) for the password. This is neither strong nor brute force resistant.

2.16

~ IRRELEVANT
~ MANUAL
~ FORTIFY
~ RATS
~ RIPS

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.
This is a configuration issue: the application should be run on an HTTPS-only server.

2.17

✖ FAILED
MANUAL
FORTIFY
RATS
RIPS

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 the clear to the user.
The new password isn't sent in the clear, however, the current password is part of the hash that is sent to the user by email.

2.18

✖ FAILED
MANUAL
FORTIFY
RATS
RIPS

Verify that information enumeration is not possible via login, password reset, or forgot account functionality.
It is possible to figure out what accounts exist via the password reset form. There is also a timing difference between non-existing and existing accounts in password validation.

2.19 ✓ PASSED MANUAL FORTIFY RATS RIPS	*Verify there are no default passwords in the application framework or any components used by the application* The application generates a fresh, random password for the administrator account on install. This password is however not generated with secure randomness.
2.20 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that request throttling is in place to prevent automated attacks against common authentication attacks suchs as brute force attacks or denial of service attacks.* The application does not support this.
2.21 ~IRRELEVANT ~MANUAL ~FORTIFY ~RATS ~RIPS	*Verify that all authentication credentials for accessing services external to the application are encrypted and stored in a protected location.* The application does not access external services.
2.22 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that forgotten password and other recovery paths use a soft token, mobile push, or an offline recovery mechanism.* The application does not support these mechanisms.
2.23 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that account logout 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.* The application does not support locked accounts.

2.24

✓ PASSED
MANUAL
FORTIFY
RATS
RIPS

Verify that if knowledge based questions (also known as "secret questions") are required, the questions should be strong enough to protect the application.
No such questions are asked.

2.25

✗ FAILED
MANUAL
FORTIFY
RATS
RIPS

Verify that the system can be configured to disallow the use of a configurable number of previous passwords.
There is no such functionality.

2.26

~ IRRELEVANT
~ MANUAL
~ FORTIFY
~ RATS
~ RIPS

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.
There are no such sensitive operations, other than changing passwords, for which see also 2.9.

2.27

✗ FAILED
MANUAL
FORTIFY
RATS
RIPS

Verify that measures are in place to block the use of commonly chosen passwords and weak passphrases.
There are no such measures in place.

2.28

✗ FAILED
MANUAL
FORTIFY
RATS
RIPS

Verify that all authentication challenges, whether successful or failed, should respond in the same average response time.
As noted before, this is not the case.

2.29

~ IRRELEVANT
~ MANUAL
~ FORTIFY
~ RATS
~ RIPS

Verify that secrets, API keys, and passwords are not included in source code, or online source code repositories.
This is not relevant for our case.

2.30	*Verify that if an application allows users to authenticate, they use a proven secure authentication mechanism.*
✓ PASSED MANUAL FORTIFY RATS RIPS	Password authentication is a secure mechanism.
2.31	*Verify that Verify that if an application allows users to authenticate, they can authenticate using two-factor authentication or other strong authentication, or any similar scheme that provides protection against username + password disclosure.*
✗ FAILED MANUAL FORTIFY RATS RIPS	No two-factor mechanisms are supported.
2.32	*Verify that administrative interfaces are not accessible to untrusted parties.*
✓ PASSED MANUAL FORTIFY RATS RIPS	See also 2.1.

Chapter 3: Session Management Verification Requirements

3.1 ✖ FAILED MANUAL FORTIFY RATS RIPS	*Verify that there is no custom session manager, or that the custom session manager is resistant against all common session management attacks.* A custom session manager is found, which is for example not protected against session hijacking.
3.2 ✖ FAILED MANUAL FORTIFY RATS RIPS	*Verify that sessions are invalidated when the user logs out.* Sessions are not destroyed when the user logs out e.g. session_destroy() is not called.
3.3 ✖ FAILED MANUAL FORTIFY RATS RIPS	*Verify that sessions timeout after a specified period of inactivity.* A specified expire period is found in the configuration but does not appear to be used anywhere.
3.5 ✖ FAILED MANUAL FORTIFY RATS RIPS	*Verify that all pages that require authentication have easy and visible access to logout functionality* All pages that require authentication have a link which allows the user to logout in the top-right corner.
3.6 ✖ FAILED MANUAL FORTIFY RATS RIPS	*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 application does not include the session identifier in the URLs. URL rewriting is supported by default in PHP, however.
3.7 ✖ FAILED MANUAL FORTIFY RATS RIPS	*Verify that all successful authentication and re-authentication generates a new session and session id* The application does not generates a new session after each successful (re-)authentication.

3.10 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that only session ids generated by the application framework are recognized as active by the application.* The application uses the default PHP session functionality, and thus can't check authenticity.
3.11 ✓ PASSED MANUAL FORTIFY RATS RIPS	*Verify that session ids are sufficiently long, random and unique across the correct active session base.* Session identifiers are generated by the default PHP session generator, this generates sufficient long, random and unique identifiers.
3.12 ✗ FAILED MANUAL FORTIFY RATS RIPS	*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.* The 'HttpOnly' setting is not enabled.
3.16 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that the application limits the number of active concurrent sessions.* The application does not limit the number of active concurrent sessions
3.17 ✗ FAILED MANUAL FORTIFY RATS RIPS	*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* The user is not able to terminate any active session.
3.18 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify the user is prompted with the option to terminate all other active sessions after a successful change password process.* The user does not have this option.

Chapter 4: Access Control Verification Requirements

4.1 ✗ FAILED MANUAL FORTIFY RATS RIPS	*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 against spoofing and elevation of privilege.* The user's role and status properties are never checked to verify whether they have the required authorization for an action. Therefore it is possible even for a deactivated user with role 'user' to log in and perform admin-level actions such as adding new admin users.
4.4 ✗ FAILED MANUAL FORTIFY RATS RIPS	*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).* It is possible for an attacker to overwrite an existing installation by manually crafting a POST request to the installer script with their desired configuration. This can be seen as a configuration issue because the install folder was not removed after installation. However, the installer does not warn the user that they should do this, and the site functions even when the installer script is present and executable.
4.5 ✗ FAILED MANUAL FORTIFY RATS RIPS	*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.* The application does not disable directory browsing. Within the `.htaccess` file there is a line that explicitly allows to access directories directly.
4.8 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that access controls fail securely.* The access control code does not fail securely: on line 62 of `testcms.php`, `User::authed()` could return something other than `false` in case of an error causing the strict check to fail and the template to still be rendered.
4.9 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that the same access control rules implied by the presentation layer are enforced on the server side.* When an users account is set to inactive, the user should not be able to log in. According to the presentation layer, however it is still possible for an inactive user to log in.

4.10 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that all user and data attributes and policy information used by access controls cannot be manipulated by end users unless specifically authorized.* A normal user has the same rights as an admin user, so this requirement is not satisfied. Because an unauthorized normal user can manipulate admin level data.
4.12 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that all access control decisions can be logged and all failed decisions are logged.* There is a logging system in place but it does not log all failed decisions regarding access control.
4.13 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that the application or framework uses strong random anti-CSRF tokens or has another transaction protection mechanism.* The application does not make use of anti-CSRF tokens.
4.14 ✗ FAILED MANUAL FORTIFY RATS RIPS	*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.* The system is not protected against aggregate or continuous access
4.15 ✗ FAILED MANUAL FORTIFY RATS RIPS	*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.* The application does not offer additional authorization nor segregation of duties.

4.16	*Verify that the application correctly enforces context-sensitive authorisation so as to not allow unauthorised manipulation by means of parameter tampering.*
✗ FAILED MANUAL FORTIFY RATS RIPS	Context-sensitive authorisation is lacking in this application.

Chapter 5: Access Control Verification Requirements

5.1

~ IRRELEVANT
~ MANUAL
~ FORTIFY
~ RATS
~ RIPS

Verify that the runtime environment is not susceptible to buffer overflows, or that security controls prevent buffer overflows.
The latest version of PHP 5.6 has no known buffer overflows in the core functionality. This is however out of scope.

5.3

✗ FAILED
MANUAL
FORTIFY
RATS
RIPS

Verify that server side input validation failures result in request rejection and are logged.
There is a logging system in the CMS that can be enabled through the `config.php` file. Input validation is done for some form fields (such as e-mail). When, for example, e-mail validation fails, there is no log entry, but the request is rejected nonetheless. Fail because not all premises are satisfied.

5.5

✗ FAILED
MANUAL
FORTIFY
RATS
RIPS

Verify that input validation routines are enforced on the server side.
All inputs are fed through the `Input` class, however, input validation is not done everywhere (for example: the `offset` parameter in the posts page should be an integer, but there is no check).

5.10

✗ FAILED
MANUAL
FORTIFY
RATS
RIPS

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
In `system/classes/pages.php` the sorting parameter in `list_all` is not fed through the prepared statement, but directly into the SQL. It must be noted that this sorting option does not seem to come from user input.

5.11

~ IRRELEVANT
~ MANUAL
~ FORTIFY
~ RATS
~ RIPS

Verify that the application is not susceptible to LDAP Injection, or that security controls prevent LDAP Injection.
The CMS does not use LDAP.

5.12 ✓ PASSED MANUAL FORTIFY RATS RIPS	*Verify that the application is not susceptible to OS Command Injection, or that security controls prevent OS Command Injection.* There are no calls to functions that involve OS Command Injection (e.g. `system`, `exec`)
5.13 ✗ FAILED MANUAL FORTIFY RATS RIPS	*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.* The CMS is susceptible to LFI in `system/classes/template.php`. The template filename in the `render()` function can contain directory separators (the use of the `basename()` function would prevent this). When the path exists, the file is given to the `parse()` function, which includes the file and puts the (interpreted) contents in the output buffer.
5.14 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that the application is not susceptible to common XML attacks, such as XPath query tampering, XML External Entity attacks, and XML injection attacks.* The RSS class does not seem to filter XML tags, so it is possible to inject XML through, for example, the configuration option `metadata.description`.
5.15 ✗ FAILED MANUAL FORTIFY RATS RIPS	*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.* The admin panel is susceptible to XSS attacks through the site name.

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 input validation (whitelisting), then lesser forms of validation such as reylisting (eliminating known bad strings), or rejecting bad inputs (blacklisting)*
✗ FAILED, 👁 MANUAL, FORTIFY, RATS, 👁 RIPS	There is barely any to no validation on any inputs.
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).*
✗ FAILED, 👁 MANUAL, FORTIFY, RATS, RIPS	There is barely any such validation.
5.21	*Verify that unstructured data is sanitized to enforce generic safety masures such as allowed characters and length, and characters potentially harmful in given context should be escaped*
✗ FAILED, 👁 MANUAL, FORTIFY, RATS, RIPS	There is mostly no sanitation.
5.22	*Make sure untrusted HTML from WYSIWYG editors or similar are properly sanitized with an HTML sanitizer and handle it appropriately according to the input validation task and encoding task.*
✗ FAILED, 👁 MANUAL, FORTIFY, RATS, RIPS	There is no sanitation on the fields that do expect html input.
5.23	*For auto-escaping template technology, if UI escaping is disabled, ensure that HTML sanitization is enabled instead.*
~ IRRELEVANT, ~ MANUAL, ~ FORTIFY, ~ RATS, ~ RIPS	There are no auto-escaping templates.

5.24 ✓ PASSED MANUAL FORTIFY RATS RIPS	*Verify that data transferred from one DOM context to another, uses safe JavaScript methods, such as `.innertext` and `.val`* The javascript actually seems decently written.
5.25 ✓ PASSED MANUAL FORTIFY RATS RIPS	*Verify when parsing JSON in browsers, that `JSON.parse` is used to parse JSON on the client. Do not use `eval()` to parse JSON on the client.* The javascript actually seems decently written.
5.26 ✓ PASSED MANUAL FORTIFY RATS RIPS	*Verify that authenticated data is cleared from client storage, such as the browser DOM, after the session is terminated.* The application does not seem to make use of localstorage or cookies to store authenticated information.

Chapter 7: Cryptography at rest verification requirements

7.2 ✓ PASSED MANUAL FORTIFY RATS RIPS	*Verify that all cryptographic modules fail securely, and errors are handled in a way that does not enable oracle padding.* There are no cryptographic modules in use other than `crypt`, which is safe.
7.6 ✓ PASSED MANUAL FORTIFY RATS RIPS	*Verify that all random number generators, 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 not to be guessable by an attacker.* The application does not use random numbers anywhere (for better or worse).
7.7 ~IRRELEVANT ~MANUAL ~FORTIFY ~RATS ~RIPS	*Verify that cryptographic algorithms used by the application have been validated against FIPS 140-2 or an equivalent standard.* The application uses the PHP `crypt` function, which has not been validated. However, this isn't really relevant to this application.
7.8 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that cryptographic modules operate in their approved modes according to their published security policies.* `crypt` is used, but largely deprecated over `password_hash` in PHP.
7.9 ~IRRELEVANT ~MANUAL ~FORTIFY ~RATS ~RIPS	*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.* There are no cryptographic keys in the application, and any HTTPS keys would be out of scope for this review.

7.11 ~IRRELEVANT ~MANUAL ~FORTIFY ~RATS ~RIPS	*Verify that all consumers of cryptographic services do not have direct access to key material. Isolate cryptographic processes, including master secrets and consider the use of a hardware key vault (HSM).* There is no key material in the web application.
7.12 ~IRRELEVANT ~MANUAL ~FORTIFY ~RATS ~RIPS	*Personally Identifiable Information should be stored encrypted at rest and ensure that communication goes via protected channels.* There is no such information in the web application and encrypted channels are a manner of server configuration (HTTPS/HSTS).
7.13 ~IRRELEVANT ~MANUAL ~FORTIFY ~RATS ~RIPS	*Verify that were possible, keys and secrets are zeroed out when destroyed.* Unclear, since this depends on the behaviour of the PHP Garbage Collector.
7.14 ✓PASSED ◉MANUAL FORTIFY RATS RIPS	*Verify that all keys and passwords are replaceable, and are generated or replaced at installation time.* The installation procedure generates fresh keys and is the only place that key material is used.
7.15 ~IRRELEVANT ~MANUAL ~FORTIFY ~RATS ~RIPS	*Verify that random numbers are created with proper entropy even when the application is under heavy load, or that the application degrades gracefully in such circumstances.* No random numbers are used.

Chapter 8: Error handling and logging verification requirements

8.1 ✗ FAILED MANUAL FORTIFY RATS RIPS	*1 Verify that the application does not output error messages or stack traces containing sensitive data that could assist an attacker, including session id,software/framework versions and personal information.* The application outputs a stack trace on the *posts* page when the `offset` parameter is not an integer (but, e.g., **false**).
8.2 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that error handling logic in security controls denies access by default.* It checks if it can find an error, such as wrong password or unknown user, but allows access by default.
8.3 ✓ PASSED MANUAL FORTIFY RATS RIPS	*Verify security logging controls provide the ability to log success and particularly failure events that are identified as security-relevant.* The application allows for writing success/failure events.
8.4 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that each log event includes necessary information that would allow for a detailed investigation of the timeline when an event happens.* The application does not log detailed information that would allow for a detailed investigation of the timeline, for example the exact time of the events is not written to the logs (only the date is written).
8.6 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that security logs are protected from unauthorized access and modification.* Every file on the server is accessible, as stated in the htaccess file.

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.*
✓ PASSED MANUAL FORTIFY RATS RIPS	No leaking of sensitive information to the log files is found.
8.10	*Verify that an audit log or similar allows for non-repudiation of key transactions.*
✗ FAILED MANUAL FORTIFY RATS RIPS	The application does not provide non-repudiation of key transactions with only the use of the log files.

Chapter 9: Data Protection Verification Requirements

9.1 ✓ PASSED MANUAL FORTIFY RATS RIPS	*Verify that all forms containing sensitive information have disabled client side caching, including autocomplete features.* There is no sensitive information, except for passwords, which use the proper (non-autocompleting) password fields.
9.2 ~ IRRELEVANT ~ MANUAL ~ FORTIFY ~ RATS ~ RIPS	*Verify that the list of sensitive data processed by the application is identified, and that there is an explicit policy for how access to this data must be controlled, encrypted and enforced under relevant data protection directives.* Irrelevant due to being a level 3 requirement.
9.3 ✓ PASSED MANUAL FORTIFY RATS RIPS	*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).* All forms use POST requests to send (sensitive) data to the server.
9.4 ✗ FAILED MANUAL FORTIFY RATS RIPS	*Verify that the application sets appropriate anti-caching headers as per the risk of the application* The application does not set anti-caching headers at all.
9.5 ~ IRRELEVANT ~ MANUAL ~ FORTIFY ~ RATS ~ RIPS	*Verify that on the server, all cached or temporary copies of sensitive data stored are protected from unauthorized access or purged/invalidated after the authorized user accesses the sensitive data* The scope of this audit is to look at the CMS, not the webserver configuration.

9.6 ~ IRRELEVANT ~ MANUAL ~ FORTIFY ~ RATS ~ RIPS	*Verify that there is a method to remove each type of sensitive data from the application at the end of the required retention policy.* Irrelevant due to being a level 3 requirement
9.7 ✓ PASSED 👁 MANUAL FORTIFY RATS RIPS	*Verify the application minimizes the number of parameters in a request, such as hidden fields, Ajax variables, cookies and header values.* Only the `PHPSESSID` cookie is stored. There are no hidden input fields in forms. JSON is used to transfer a limited amount of data.
9.8 ~ IRRELEVANT ~ MANUAL ~ FORTIFY ~ RATS ~ RIPS	*Verify the application has the ability to detect and alert on abnormal numbers of requests for data harvesting for an example screen scraping.* Irrelevant due to being a level 3 requirement.
9.9 ✓ PASSED 👁 MANUAL FORTIFY RATS RIPS	*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).* No cookies other than `PHPSESSID` are stored.
9.10 ~ IRRELEVANT ~ MANUAL ~ FORTIFY ~ RATS ~ RIPS	*Verify accessing sensitive data is logged, if the data is collected under relevant data protection directives or where logging of accesses is required.* The system does not contain sensitive data.

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.*
~IRRELEVANT ~MANUAL ~FORTIFY ~RATS ~RIPS	The system does not contain sensitive data.

Chapter 10: Communications security verfication requirements

10.X	*TLS requirements*
~IRRELEVANT ~MANUAL ~FORTIFY ~RATS ~RIPS	All requirements in Chapter 10 are out of scope as they cover server configuration.

Chapter 3

Reflection

The ASVS is a very comprehensive checklist to make sure you've thought about a lot of common security holes. The items on that list however often vary in terms of how clear they are, or where to check them. Some could span the entire code base. This makes verifying whether these requirements are met fairly difficult. The structure of a single level list also inflates how many items there should be checked, while some items are dependent on each other.

Because a lot of the things on the ASVS list require some reasoning, the applicability of static analysis tools remains limited to broad checking for some mistakes across the entire code base. These static analysis tools can, for the extent we used them at least, not differentiate between sensitive or insensitive information (except if something is named like `password`) or decide what pages should be protected by a session. Manual checking of the code (with some help from standard *NIX tools such as `grep` and `awk`) played a very important role in this audit.

Since we stopped probing after finding the first issue, static analysis tools did not really decide many issues between *passed* or *failed*. The types of issues the static analysis tools found were typically so glaring that a casual poke at the user interface also revealed them. The application also was small enough to casually read all code. If one would try to find more hidden issues in bigger code bases, SCA tools may prove more worthwhile.

RATS barely found anything worth mentioning. RIPS was able to find some cross-site scripting errors, but the more advanced modes popped up more false positives than useful results.

Matthijs ran the Fortify tool for us on his Windows PC, so that we did not all to have to try to get it working. The report Fortify produced however was not very useful, containing dozens of pages of example bugs, and only one tiny list with some actual issues. The actual user interface was a bit more useful. Fortify found one hard-to-spot issue, although we independently also found it by hand.

A Dockerfile [1] was set up to help easily run the web application. Running the

[1] `https://gitlab.science.ru.nl/twiggers/software-security-docker/`

application helped to gain some insight into how the application worked, and also revealed things about the authentication infrastructure that made marking some things on the checklist trivial as they were not present in the application.

We experienced that the convenience with verifying the code differed greatly amongst the group members. Some of us had lots of experience with PHP and reviewing code, and found this very easy, but others needed more time for this. While we started working on the review early, and had actually completed a large part of it by the end of November, progress then stalled. It was also unfortunate we did not hear from Matthijs himself that he would not be completing the course. In a next review, we would take care to keep ourselves on track such that we have time to get another pair of eyes to look at (parts of) the review. More frequent meeting could help with this: we only met when we needed to divide tasks.

TestCMS is largely designed from the ground up. It was constructed reasonably sensibly, but even they are reinventing the wheel. It is better practice to use established frameworks and perhaps even template engines in PHP, such that things like escaping query parameters and HTML output are handled in much more convenient ways. In some frameworks these things work opt-out instead of opt-in: this makes forgetting to escape something fairly difficult.

In TestCMS there is also a lot of dead code, which further distracts from the relevant parts of the application.

Handling the escaping and such in the framework and turning it on by default also simplifies an audit: instead of having to look at everything, a reviewer can simply focus on the output handling code and those few places where raw strings are explicitly printed.