DNTP security requirements and testing plan

27th August 2015

Chapter 1

Security requirements

1.1 Introduction

Spring Boot and Angular offer out-of-the-box protection against SQL injection attacks, cross-site scripting attacks, and more. Spring Boot supports role-based permissions, such that, e.g., only administrators can add and edit user accounts. On top of that, we implemented data-driven access control: users should only be able to access and modify the requests that they own (have created) or are assigned to (e.g., PALGA employees, scientific council, etc.). We will use an SSL certificate for securing the communication between user and server.

Possible vulnerabilities/attack scenario's, if the application is not properly secured, include:

- Someone gains access to the server and finds/accesses the database;
- Someone finds a vulnerability in Spring Boot/other libraries/our application and exploits it to get access to the data;
- Someone sets up a man-in-the-middle attack and can act to the system as the user whose connection is compromised;
- Someone acquires authentication credentials of a user (possibly the administrator) and can login and act as the user.

These and other vulnerabilities/attacks are within the scope of the OWASP Application Security Verification Standard (ASVS) standard. Compliance with the standard should address all of these issues. Still, a user may have exposed user credentials to an attacker (e.g., using the same password on a compromised system). Mitigation of such attacks (e.g., using cryptocards for authentication) is out of scope for this system. Included is, however, logging of user activity (enabling analysis of the damage when an intruder has been detected).

We aim at compliance with level 2 of the OWASP Application Security Verification Standard (ASVS), 2014 version [3]. Level 2 of the standard contains security verification requirements on several categories: authentication, session management, access control, malicious input handling, cryptography, error handling and logging, data protection, communications, HTTP, business logic, and files and resources. In the next section we summarise these requirements. In Chapter 2 we describe the measures taken to meet those requirements and list the known weaknesses of the system.

1.2 Security guidelines

We aim at compliance with level 2 from the OWASP Application Security Verification Standard (ASVS) [3]. We give a summary of the requirements below. OWASP also provides a security testing guide [1] and a Top 10 [2] of security risks.

1.2.1 Application specific requirements

The client prioritises security of data over availability of the application.

1.2.2 Summary of security requirements from ASVS, level 2

V2: Authentication Requirements:

- Non-public pages require authentication (enforced server-side);
- Strong passwords;
- No weaknesses in account update/creation functionality;
- Enable to securely update credentials;
- Logging of authentication decisions (for security investigations);
- Secure password hashing and storage;
- No leaking of credentials;
- No harvesting of user accounts possible through login functionality;
- No default passwords active;
- Blocking attacks that try multiple passwords and multiple accounts (V2.20);
- 'Forgot password' with expiring unique link, not sending an actual password;
- No 'secret questions';
- Possibility to disallow a number of previously chosen passwords (V2.25).

V3: Session Management Requirements:

- Proper session management (invalidation, timeout, new session id after login, only accept framework generated sessions; session tokens long enough and securely generated, limited cookie domain);
- Logout links;
- No leaking of the session id in URLs;
- HttpOnly (V3.14?);
- Secure properties for session cookies (V3.15);
- Disallow duplicate sessions for a user originating from different machines (V3.16).

V4: Access Control Requirements:

- Allow users only to secured functions, pages, files that they are authorised to use;
- Allow users only to directly access and manipulate objects (with id reference), for which they have permission;
- Access controls fail securely;
- Directory browsing disabled;
- Verify consistency of client-side and server-side access control; all access control enforced server-side;
- Access control rules and data not changeable by users (e.g., changing ownership of data);
- Logging of access control decisions;
- Strong random tokens against Cross-site Request Forgery (CSRF/XSRF);
- Aggregate access control protection (V4.17), e.g., scraping by generating a large sequence of requests (a possibly authenticated user looking for security holes).

V5: Malicious Input Handling Requirements:

- No buffer overflow vulnerabilities (V5.1);
- UTF8 specified for input;
- Input validation on server-side. Failure rejects input;
- Protection agains injection attacks (SQL, OS);
- Escape untrusted client data;
- Protect sensitive fields (e.g., role, passwords) from automatic binding (V5.17);
- Protection against parameter pollution attacks (V5.18).

V7: Cryptography at Rest Requirements:

- Cryptography performed server-side;
- Cryptography functions fail securely;
- Access to master secret protected;
- Random numbers generated for security (session id, password hashing, XSRF tokens) generated by secure cryptographic functions;
- Policy for managing cryptographic keys (creating, distributing, revoking, expiry).

V8: Error Handling and Logging Requirements:

- No logging of sensitive data, session id, personal details, credentials, stack traces;
- Logging implemented server-side, only on trusted devices;
- Error handling logic should deny access by default (e.g., throw exception which is not caught).

V9: Data Protection Requirements:

- No client side caching, autocomplete features on sensitive fields;
- Sensitive data in body, not in URL parameters;
- `no-cache` and `no-store` Cache-Control headers for sensitive data (V9.4);
- Protection of cached data server side.

V10: Communications Security Requirements:

- Valid SSL certificate chain (V10.1);
- Use SSL for all connections (V10.3);
- SSL connection failures are logged (V10.4);
- Connections to external systems are authenticated (V10.6);
- Connections to external systems use an account with minimum privileges (V10.7).

V11: HTTP Security Requirements:

- Only accept defined set of request methods;
- Response headers contain content type header with safe character set;
- Headers contain only printable ASCII characters (V11.6?);
- Use X-Frame-Options header to prevent clickjacking;
- Do not allow HTTP headers to be spoofed (V11.9?);
- Do not expose system information in HTTP headers.

V15: Business Logic Requirements:

- Business logic handled in a protected environment;
- Do not allow spoofing of actions as another user, e.g., hijacking another user's session, or spoofing a user id;
- Do not allow tampering high value parameters (prices, balances, etc.);
- Audit trail, system logs, monitoring of user activity;
- Protect against disclosure attacks, such as direct object reference, session brute force, etc.;
- Detection of and protection against brute force and denial of service attacks (V15.6);
- Access control to prevent elevation of prililege attacks;
- Process steps in correct order;
- Process steps taken in realistic human time (V15.8: Won't verify, would require different setup for automated testing.);
- Business limits enforced, such as maximum number of requests.

V16: Files and Resources Requirements:

- URL redirects do not include unvalidated data;
- Canonise file names and paths;
- Files from untrusted sources scanned by antivirus scanner (V16.3: won't implement, client side concern);
- No untrusted parameters used in lookup of file locations;
- Parameters from untrusted sources canonicalised, input validated, output encoded to prevent file inclusion attacks (V16.5);
- Use X-Frame-Options header to prevent inclusion of arbitrary remote content;
- Files from untrusted sources (uploads) stored outside webroot;
- By default access to remote resources or systems is denied by the application;
- The application does not execute data from untrusted sources.

Chapter 2

Security measures

Most requirements are met by the built-in security of the Spring and Angular frameworks and by our server configuration. In Section 2.1 we list the security measures that have been taken and cover most of the requirements. In Section 2.2 we list the open issues that need to be evaluated. In Section 2.3 we list the tests that are being performed on code and deployment. Section 2.4 lists the known vulnerabilities.

2.1 Implementation and frameworks

The measures that have been taken to secure the application:

- Spring security for authentication and authorization;
- Spring and Angular prevent Cross-site request forgery (CSRF);
- Access to functionality is protected using Spring routing rules;
- Access to data in controller logic and using security annotations.
- Using JPA derived queries in the repository classes (SQL queries are derived from interface method names), preventing SQL injection.
- Input sanitisation by Spring (and Angular).

For securing the infrastructure:

- Application is run in user space;
- Apache proxy in between, application not directly approachable;
- Only ports 22, 80, and 443 open from the outside (firewall rules);
- On port 80, Apache should only forward to https (rewrite rule, status 301 (Moved permanently)) [TODO];
- `mod_security` has been added to Apache;
- Access log by Apache, logging all incoming requests (URIs, not data), `mod_security` and by the application.

2.2 Open issues

There is a list of requirements for which it is not clear if the they are met and they are not included in the checks in the next section.

- Blocking attacks that try multiple passwords and multiple accounts (V2.20); (*Trying multiple passwords will result in blocking; trying multiple accounts not, but there won't be many accounts, so this is OK.*)
- Possibility to disallow a number of previously chosen passwords (V2.25); (*Actually, this requirement does not add security.*)

- HttpOnly (V3.14);
- Secure properties for session cookies (V3.15);
- Disallow duplicate sessions for a user originating from different machines (V3.16); (*Won't implement.*)
- Aggregate access control protection (V4.17), e.g., scraping by generating a large sequence of requests (a possibly authenticated user looking for security holes); (*Is not automatically detected. May be detected based on the logs.*)
- No buffer overflow vulnerabilities (V5.1);
- Protect sensitive fields (e.g., role, passwords) from automatic binding (V5.17); (*Should be provided by Spring and Angular.*)
- Protection against parameter pollution attacks (V5.18). (*Should be provided by Spring and Angular.*)
- `no-cache` and `no-store` Cache-Control headers for sensitive data (V9.4);
- SSL connection failures are logged (V10.4);
- Headers contain only printable ASCII characters (V11.6); (*For cookies, this is enforced by `mod_security`.*)
- Do not allow HTTP headers to be spoofed (V11.9);
- Detection of and protection against brute force and denial of service attacks (V15.6); (*Is not automatically detected. May be detected based on the logs.*)
- Parameters from untrusted sources canonicalised, input validated, output encoded to prevent file inclusion attacks (V16.5); (*Should be provided by Spring and Angular.*)

2.3 Security testing

2.3.1 Code checks

The analysis that has been done on code/application level:

- Security code review:
  - Proper routes, method annotation on all REST controllers?
  - How are user roles assigned? Who can change them? Should only be administrators.
  - No custom queries or all nicely prepared using prepared statements (JPA provides this);
  - Check that user input is not used for: demermining filesystem paths, checking for access, determining user id or roles, database key in a query.[1]
- Test injection:
  - Test for XSS attacks: try to insert HTML and Javascript in input fields; this should be handled automagically by Spring.
- Regularly check `https://pivotal.io/security` for vulnerabilities in Spring.

[1] A common error: checking access based on the *id* in the url (`@PathVariable`), but fetching the database object based on the *id* in the request body.

2.3.2 Deployment checks

Disable dev/test behaviour on the production environment. Checking this list is part of deployment to production.

- Default users disabled (e.g., `palga` – check startup log of application and user administration);
- No action `/test/clear/` available;
- Check account blocking period after *n* failed login attempts.
- Check for SSL certificate, make sure key chain is secure and valid.
- Check that requests on port 80 are forwarded to https.
- Check that the database is not accessible (except from localhost) and passwords are properly set.
- Check that database password in the application configuration is not exposed (e.g., in `puppet`).
- Perhaps, write a script that checks if the current deployment uses the newest versions of included frameworks.

2.4 Known weaknesses

- Information sent plain text by email: password recovery link.

Bibliography

[1] OWASP. *Testing Guide*. 4.0 edition, 2013. https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf.

[2] OWASP. *Top 10 (2013)*. 2013. http://owasptop10.googlecode.com/files/OWASPTop10-2013.pdf.

[3] OWASP. *Application Security Verification Standard (2014)*. 2014. https://www.owasp.org/images/5/58/OWASP_ASVS_Version_2.pdf.