LOG8371E
Software Quality Engineering

TP3: Security

December 2, 2022

Petr Smejkal
Mikulas Vesely
Lucia Cahojova
Marc Garcia Massaneda
Omar Tanveer Butt Ramirez

POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

Contents

1 Abstract

A good quality software that satisfies software quality standards secures meeting all the product specifications and requirements, saves a lot of money and time, is more maintainable and therefore results in happy and satisfied customers.

In our paper, we focus on the software security and further elaborate our two previous works, by performing security testing. We verify the Mango M2M software security through running static analysis using SonarQube platform and SonarScanner and performing penetration testing using the OWASP ZAP tool. We then compare the differences between the results of the two approaches. Our results show that todo.

2 Introduction

This paper is a direct elaboration of our two previous team works. In the first one, we defined software quality plans, software quality assurance activities and verified the methods for different quality characteristics of the Mango M2M software. In the second one, We verified the Mango M2M software performance through profiling with plugin in IntelliJ IDEA and performed load testing with JMeter. After that, we optimised the system and compared the results of the two versions.

In our third team work project, to further expand our previous work and learn more about the quality of a software, we will focus on the following objectives:

- Software security objectives and security assurance process.
- Identification of security vulnerabilities through static analysis of the source code.
- Identification of security vulnerabilities through penetration testing.

Throughout the project we will be working with open-source Mango software which is a crucial element of a machine to machine system. Its role is to gather and process data, present this raw data in a way which is comprehensible to a human operator who can, in turn, send commands and thus control the process. Mango is an "open source alternative for machine to machine(M2M) software. Mango is browser-based, Ajax-enabled M2M software that enables users to access and control electronic sensors, devices, and machines over multiple protocols simultaneously"[1].

Inseparable part of a good quality software is the system security. It is a set of policies and procedures that safeguard the integrity, confidentiality, and availability of computer systems and data. It's important because it helps protect information and computer systems from unauthorized access, use, disclosure, or destruction. It also helps ensure that authorized users have access to the information and resources they need, when they need it.

In this paper, we will focus on spotting software security vulnerabilities, which are weaknesses in design,implementation, or operations. There are many potential security vulnerabilities in software applications. Some of the most common include buffer overflows, SQL injection, cross-site scripting, command injection, file inclusion vulnerabilities, privilege escalation, path traversal or denial of service. These vulnerabilities can allow attackers to gain access to sensitive data, execute malicious code, or crash applications.

At first, we will try to spot the vulnerabilities by performing static analysis. To do so, we will use the SonarQube open-source web-based tool that helps developers produce code free from security issues, bugs, vulnerabilities, smells, and general issues [2] and also SonarScanner. We will try to spot eight different vulnerabilites, to which we will provide a short description and its potential risk. We will also provide a manual to run the SonarQube.

Secondly, we will perform penetration testing using the OWASP ZAP tool, which is an open-source web app scanner. We will again try eight different vulnerabilites, to which we will provide a short description and its potential risk. We will also provide a manual to run the OWASP ZAP tool.

At last, we will compare the results of the static analysis with the results of the penetration testing, discuss differences between the results of the two approaches in terms of the numbers and the CWE categories of the detected vulnerabilities.

3 Q1: Static Analysis

In the following section we will provide the detected vulnerabilities or security hotspots with more details and recommendations for solving the issues. To do so, we will be using SonarQube and Sonar Scanner.

3.1 Static Analysis User Manual

In this section, we are going to demonstrate how to set up and run the tool we used to perform static analysis on the whole code base of Mango. Also, we are going to explain why we chose this tool over the other alternative.

We chose SonarQube, which is an advanced tool that statically scans a code base in order to find bugs and possible error flows in the software system. There's also another version of this tool called SonarCloud. It works similarly but runs in a remote cloud, however, we found it not as suitable as SonarQube for our specific needs [3]. Now, let's present a manual on how to set up SonarQube and run static analysis on Mango.

Note that we are showing the steps on macOS 12.6 (21G115), but the set up and run process is going to be very similar on other operating systems (e.g. Linux or Windows).

3.1.1 Download and run SonarQube on localhost

At first, proceed to the official website of SonarQube [4], where you are going to find a link that will start the installation. We used the Community version of SonarQube.

After downloading the zip file, unpack it to any location you wish (for example to your Desktop). Then, open a terminal window and move to the following location (it can differ depending on where you put the unpacked SonarQube zip):

```
cd /Users/[your_username]/Desktop/sonarqube/bin/macosx-universal-64/
```

Now, run the **sonar.sh** script, which will start the program - following command [4]:

```
./sonar.sh console
```

In case there's no error and SonarQube starts running, you can skip to checking if SonarQube is running. But if an error occurs, it is most likely caused by your Java version. SonarQube require Java 11 or newer in order to run (before, we used Java 1.8 to run Mango itself, so you may have your Java set up to that old version).

To check your java version type **java -version** to your console. If your Java is not version 11 or newer, follow those step so set it up:

At first, check whether you have any of the new Java versions on your machine by this command:

```
/usr/libexec/java_home -V
```

It will list all versions that are installed on your local machine. If there's no new version, install java with brew, if there's a new Java version in the list, skip the next step:

```
brew install openjdk@17
```

Now, when there's a Java 11+ installed, copy its path from the list of all Javas. Then, proceed to your home directory **cd ˜** and open the file **.bash_profile** in any text editor. You will see a variable **JAVA_HOME** that you need to set up the the Java 11+ path. So, be sure that there's a line in your **.bash_profile** that looks something like this:

```
export JAVA_HOME= \
"/Users/[user_name]/Library/Java/JavaVirtualMachines/openjdk-17.0.1-1/Contents/Home"
```

Restart your terminal to load the new path. Now, you can check the java version by **java -version** and it should be the new one. Finally, run the command that we proposed before (it will start SonarQube and it will be accessable on localhost):

```
./sonar.sh console
```

3.1.2 Check if SonarQube is running

Open a browser and go to localhost, port 9000 (**http://localhost:9000/**. You should see a login form shown in Figure 1:

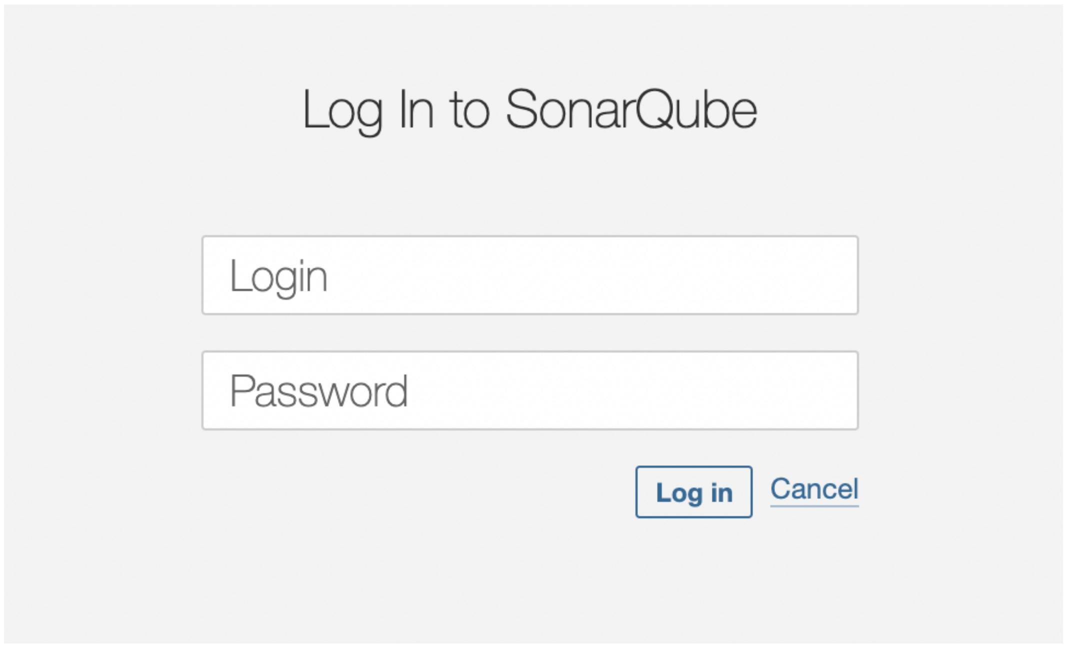


Figure 1: SonarQube login form

You will be able to log in using the following credentials:

```
login: admin
password: admin
```

On the first log in you will be asked to change your password - be sure to remember it for later. As you log in, you are able to use SonarQube to analyze a code base, which we are showing in the next section.

3.1.3 Run SonarScanner for Ant project

When you are sure that SonarQube is running well on your machine, we can continue by running the static analysis of the Mango code. We are going to use a jar file of SonarScanner for Ant projects.

At first, download the file from the official website [5] . Now, we need to integarte the jar into our Ant project. In a terminal window, move to the MangoSource directory (for ex. like this if you have your mangoSource on the Desktop):

```
cd /Users/[user_name]/Desktop/mangoSource
```

In any text editor open the **build.xml** configuration file, that will allow us to set up SonarScanner for Ant:

```
open cd /Users/[user_name]/Desktop/mangoSource/build.xml
```

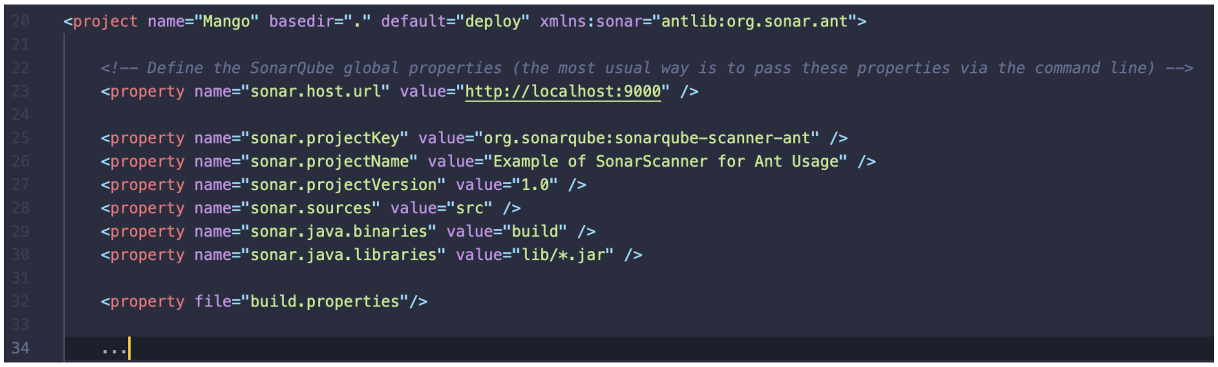
At first, change the very first line of the **build.xml** file to:

```
<project name="Mango" basedir="." default="deploy" xmlns:sonar="antlib:org.sonar.ant">
```

Then, add the global and project properties of **build.xml** (still at the top of the file):

```
<property name="sonar.host.url" value="http://localhost:9000" />

<property name="sonar.projectKey" value="org.sonarqube:sonarqube-scanner-ant" />
<property name="sonar.projectName" value="Example of SonarScanner for Ant Usage" />
<property name="sonar.projectVersion" value="1.0" />
<property name="sonar.sources" value="src" />
<property name="sonar.java.binaries" value="build" />
<property name="sonar.java.libraries" value="lib/*.jar" />
```

The **build.xml** file should look as shown in Figure 2 this after the previous changes:



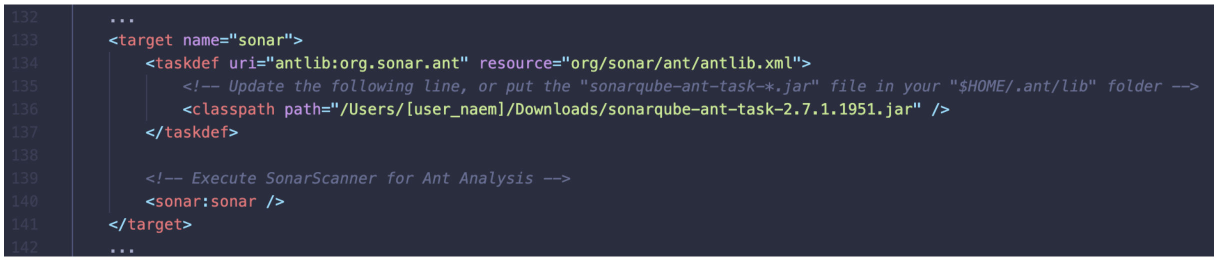
```
<project name="Mango" basedir="." default="deploy" xmlns:sonar="antlib:org.sonar.ant">

    <!-- Define the SonarQube global properties (the most usual way is to pass these properties via the command line) -->
    <property name="sonar.host.url" value="http://localhost:9000" />

    <property name="sonar.projectKey" value="org.sonarqube:sonarqube-scanner-ant" />
    <property name="sonar.projectName" value="Example of SonarScanner for Ant Usage" />
    <property name="sonar.projectVersion" value="1.0" />
    <property name="sonar.sources" value="src" />
    <property name="sonar.java.binaries" value="build" />
    <property name="sonar.java.libraries" value="lib/*.jar" />

    <property file="build.properties"/>

    ...
```

Figure 2: General and Project properties in build.xml

Lastly, move farther in the file (to the targets) and add a target for SonarScanner (note that you have to set the pathname for the jar file that you downloaded before):

```
<target name="sonar">
    <taskdef uri="antlib:org.sonar.ant" resource="org/sonar/ant/antlib.xml">
        <!-- Update the following line, or put the "sonarqube-ant-task-*.jar" file in your "$H
        <classpath path="path/to/sonar/ant/task/lib/sonarqube-ant-task-*.jar" />
    </taskdef>
    <sonar:sonar />
</target>
```

The **build.xml** file added target should look as shown in Figure 3.



```
...
<target name="sonar">
    <taskdef uri="antlib:org.sonar.ant" resource="org/sonar/ant/antlib.xml">
        <!-- Update the following line, or put the "sonarqube-ant-task-*.jar" file in your "$HOME/.ant/lib" folder -->
        <classpath path="/Users/[user_naem]/Downloads/sonarqube-ant-task-2.7.1.1951.jar" />
    </taskdef>

    <!-- Execute SonarScanner for Ant Analysis -->
    <sonar:sonar />
</target>
...
```

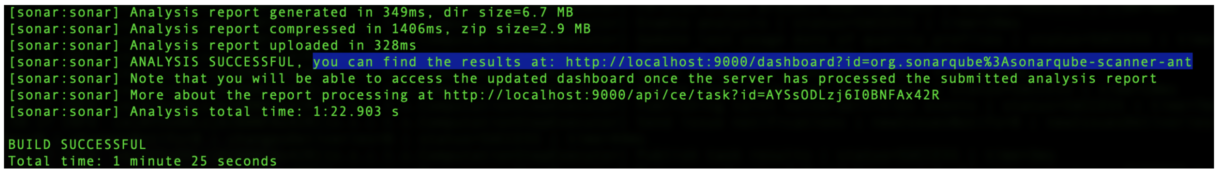
Figure 3: Sonar target in build.xml

Now, we are able to run SonarScanner for Mango and review results of static analysis. In a terminal window, change to the mangoSource directory and run ant with target sonar (the following command):

```
ant sonar -Dsonar.login="login_name" -Dsonar.password="your_password"
```

Where **”login_name”** is **admin** and **password** is **the password you chose when logging to localhost:9000**. This will initiate the static analysis (note that you have to have SonarQube running on your localhost when performing the analysis).

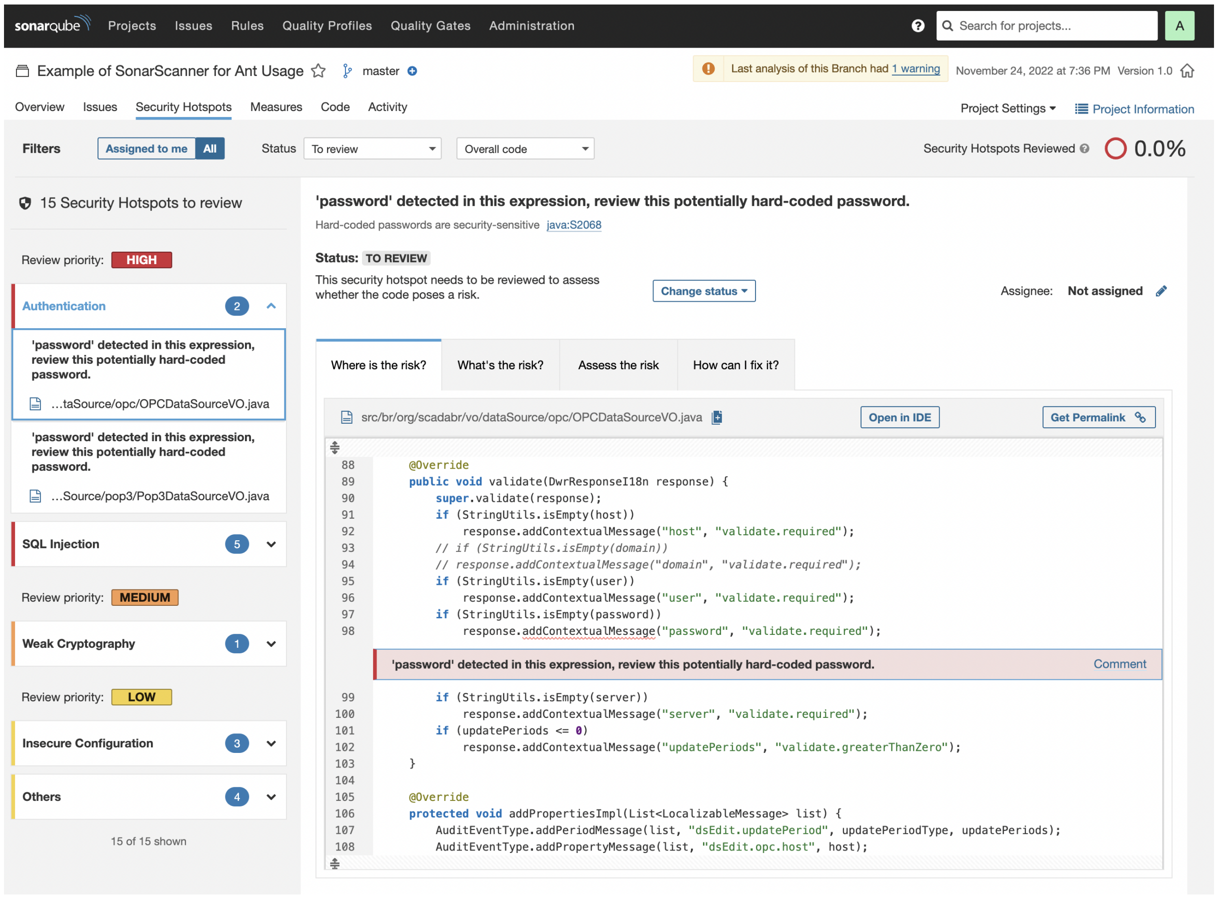
This may take a few minutes. When the analysis is finished, you will be provided a link that you can open in a browser to see the results (shown in Figure 4).



```
[sonar:sonar] Analysis report generated in 349ms, dir size=6.7 MB
[sonar:sonar] Analysis report compressed in 1406ms, zip size=2.9 MB
[sonar:sonar] Analysis report uploaded in 328ms
[sonar:sonar] ANALYSIS SUCCESSFUL, you can find the results at: http://localhost:9000/dashboard?id=org.sonarqube%3Asonarqube-scanner-ant
[sonar:sonar] Note that you will be able to access the updated dashboard once the server has processed the submitted analysis report
[sonar:sonar] More about the report processing at http://localhost:9000/api/ce/task?id=AYSsODLzj6I0BNFAx42R
[sonar:sonar] Analysis total time: 1:22.903 s

BUILD SUCCESSFUL
Total time: 1 minute 25 seconds
```

Figure 4: Result of Ant target Sonar

After opening the link in a browser, you will see all the results including Security Hostspots, various measurements results, other issues and much more. An example of a Security Hotspot found in Mango is demonstrated in Figure 5.



'password' detected in this expression, review this potentially hard-coded password.

'password' detected in this expression, review this potentially hard-coded password.

'password' detected in this expression, review this potentially hard-coded password.

Hard-coded passwords are security-sensitive java:S2068

Status: TO REVIEW

This security hotspot needs to be reviewed to assess whether the code poses a risk.

```java
    @Override
    public void validate(DwrResponseI18n response) {
        super.validate(response);
        if (StringUtils.isEmpty(host))
            response.addContextualMessage("host", "validate.required");
        // if (StringUtils.isEmpty(domain))
        // response.addContextualMessage("domain", "validate.required");
        if (StringUtils.isEmpty(user))
            response.addContextualMessage("user", "validate.required");
        if (StringUtils.isEmpty(password))
            response.addContextualMessage("password", "validate.required");
```

'password' detected in this expression, review this potentially hard-coded password.

```java
        if (StringUtils.isEmpty(server))
            response.addContextualMessage("server", "validate.required");
        if (updatePeriods <= 0)
            response.addContextualMessage("updatePeriods", "validate.greaterThanZero");
    }

    @Override
    protected void addPropertiesImpl(List<LocalizableMessage> list) {
        AuditEventType.addPeriodMessage(list, "dsEdit.updatePeriod", updatePeriodType, updatePeriods);
        AuditEventType.addPropertyMessage(list, "dsEdit.opc.host", host);
```

Figure 5: An example of Security Hotspot in Mango (result of SonarQube analysis)

In the next section, we closely described all results of the static analysis and chose the main vulnerabilities, which we then analyze in great detail as well as we propose a solution to each of the vulnerability.

3.2 Vulnerabilities

3.2.1 Authentication

Description: This is a security hotspot. `"password"` detected in an expression. It is easy to extract strings from an application source code or binary, so passwords should not be hard-coded (should not appear in the source code).
Potential Risk: Attackers could get access to sensitive credentials.
File Name: `src/br/org/scadabr/vo/dataSource/opc/OPCDataSourceVO.java`
Severity Level: High.
Type of Vulnerability: Identification and Authentication Failures (OWASP)/ Broken Authentication (OWASP) / Use of Hard-coded Credentials (CWE)
Recommendation for Solving the Vulnerability: Store the credentials in a database or in a configuration file not pushed to the code repository. If a password has been disclosed through the source code, it must be changed.

3.2.2 Weak Criptography

Description: This is a security hotspot. A pseudorandom number generator is used. Using pseudorandom number generators (PRNGs) is security-sensitive, so it must be made sure that it is safe.
Potential Risk: It may be possible for an attacker to guess the next value that will be generated, and use this guess to impersonate another user or access sensitive information.
File Name: `src/com/serotonin/mango/rt/dataSource/virtual/ChangeTypeRT.java`
Severity Level: Medium.
Type of Vulnerability: Cryptographic Failures (OWASP) / Sensitive Data Exposure (OWASP) / Insufficient Cryptography (OWASP) / Use of Insufficiently Random Values (CWE)
Recommendation for Solving the Vulnerability: Use a cryptographically strong random number generator (RNG) like "java.security.SecureRandom" in place of this PRNG. You should also not expose the generated the generated random value, i.e., store it in a database.

3.2.3 SQL Injection

Description: This is a security hotspot. There is a dynamically formatted SQL query in the code. It must be made sure that the use is safe.
Potential Risk: Formatted SQL queries can be difficult to maintain, debug and can increase the risk of SQL injection when concatenating untrusted values into the query.
File Name: `src/com/serotonin/mango/db/DBConvert.java`
Severity Level: High.
Type of Vulnerability: Injection (OWASP) / Improper Neutralization of Special Elements used in an SQL Command (CWE)
Recommendation for Solving the Vulnerability: Use parameterized queries, prepared statements, or stored procedures and bind variables to SQL query parameters.

3.2.4 Insecure configuration

Description: This is a security hotspot. There is debug feature in the code. It should be deactivated before delivering the code in production.

Potential Risk: These features enable developers to find bugs more easily and thus facilitate also the work of attackers. It often gives access to detailed information on both the system running the application and users.
File Name: `src/br/org/scadabr/rt/dataSource/dnp3/Dnp3DataSource.java`
Severity Level: Low.
Type of Vulnerability: Security Misconfiguration (OWASP) / Sensitive Data Exposure (OWASP) / Active Debug Code (CWE) / Information Exposure Through Debug Information (CWE)
Recommendation for Solving the Vulnerability: Do not enable debug features on production servers or applications distributed to end users. Loggers should be used (instead of printStackTrace used in this case) to print throwables.

3.2.5 Publicly writable directories

Description: This is a security hotspot. There is a publicly writable directory in the code. It must be made sure that the use is safe.
Potential Risk: Operating systems have global directories where any user has write access. A malicious user can try to create a file with a predictable name before the application does. A successful attack can result in other files being accessed, modified, corrupted or deleted.
File Name: `src/com/serotonin/mango/rt/maint/work/ReportWorkItem.java`
Severity Level: Low.
Type of Vulnerability: Broken Access Control (OWASP) / Sensitive Data Exposure (OWASP) / Sensitive Data Exposure (CWE) / Creation of Temporary File in Directory with Incorrect Permissions (CWE)
Recommendation for Solving the Vulnerability: Use a dedicated sub-folder with tightly controlled permissions or create secure-by-design APIs to create temporary files.

3.2.6 Handle an exception

Description: The signature of servlet methods tend to throw exceptions like IOException or ServletException, but it's not a good idea to let such exceptions be thrown.
Potential Risk: Failing to catch the exceptions of the servlet could leaves the system in a vulnerable state, even risking to have a denial-of-service attacks or exposing sensitive information, because after an exception is thrown the servlet sends debugging information back to the user that could be valuable to an attacker.
File Name: `src/com/serotonin/mango/web/servlet/ChartExportServlet.java`
Severity Level: Minor.
Type of Vulnerability: Sensitive Data Exposure (OWASP) / Uncaught Exception in Servlet (CWE)
Recommendation for Solving the Vulnerability: Handle the exception catches and avoid sending debugging information .

3.2.7 Authentication

Description: This is a security hotspot. `"password"` detected in an expression. It is easy to extract strings from an application source code or binary, so passwords should not be hard-coded (should not appear in the source code).

Potential Risk: Attackers could get access to sensitive credentials.
File Name: `src/com/serotonin/mango/vo/dataSource/pop3/Pop3DataSourceVO.java`
Severity Level: High.
Type of Vulnerability: Identification and Authentication Failures (OWASP)/ Broken Authentication (OWASP) / Use of Hard-coded Credentials (CWE)
Recommendation for Solving the Vulnerability: Store the credentials in a database or in a configuration file not pushed to the code repository. If a password has been disclosed through the source code, it must be changed.

3.2.8 SQL Injection

Description: This is a security hotspot. There is a dynamically formatted SQL query in the code. It must be made sure that the use is safe.
Potential Risk: Formatted SQL queries can be difficult to maintain, debug and can increase the risk of SQL injection when concatenating untrusted values into the query.
File Name: `src/com/serotonin/mango/db/dao/BaseDao.java`
Severity Level: High.
Type of Vulnerability: Injection (OWASP) / Improper Neutralization of Special Elements used in an SQL Command (CWE)
Recommendation for Solving the Vulnerability: Use parameterized queries, prepared statements, or stored procedures and bind variables to SQL query parameters.

3.3 Summary of the Results

Althought there are many vulnerabilities and security hotspots that need to be considered, there are not that many different types. As we have previously seen, the most dangerous ones that we have found are Authentication and SQL Injection, because they can expose and risk highly valuable information for the user, although the most common vulnerability found has been Exception Handling but its severity level is minor, meaning that it's not that dangerous for the user.

4 Q2: Penetration Testing

Second type of security assessment was penetration testing, which is a dynamic way to find out vulnerabilities in the system. It can discover even those security risks, that would not be possible to see during static analysis.

4.1 Penetration Testing User Manual

In this section, we are going to demonstrate how to set up and run the tool we used to perform penetration testing on the whole code base of Mango. We were instructed to use tool called OWASP ZAP (Zed Attack Proxy), the penetration testing was performed on Ubuntu 20.04.5 LTS.

4.1.1 Download and run OWASP ZAP

At first, proceed to the official website of the tool [6], where you are going to find links to download different versions of the tool, for all platforms (Windows, Linux, macOS). You can also choose wheter to download installer or package. We chose to download installer for Linux, one large (approx. 240 MB) **ZAP_2_12_0_unix.sh** file started to download. When the download is finished, locate the downloaded file, open terminal for that location and run command:

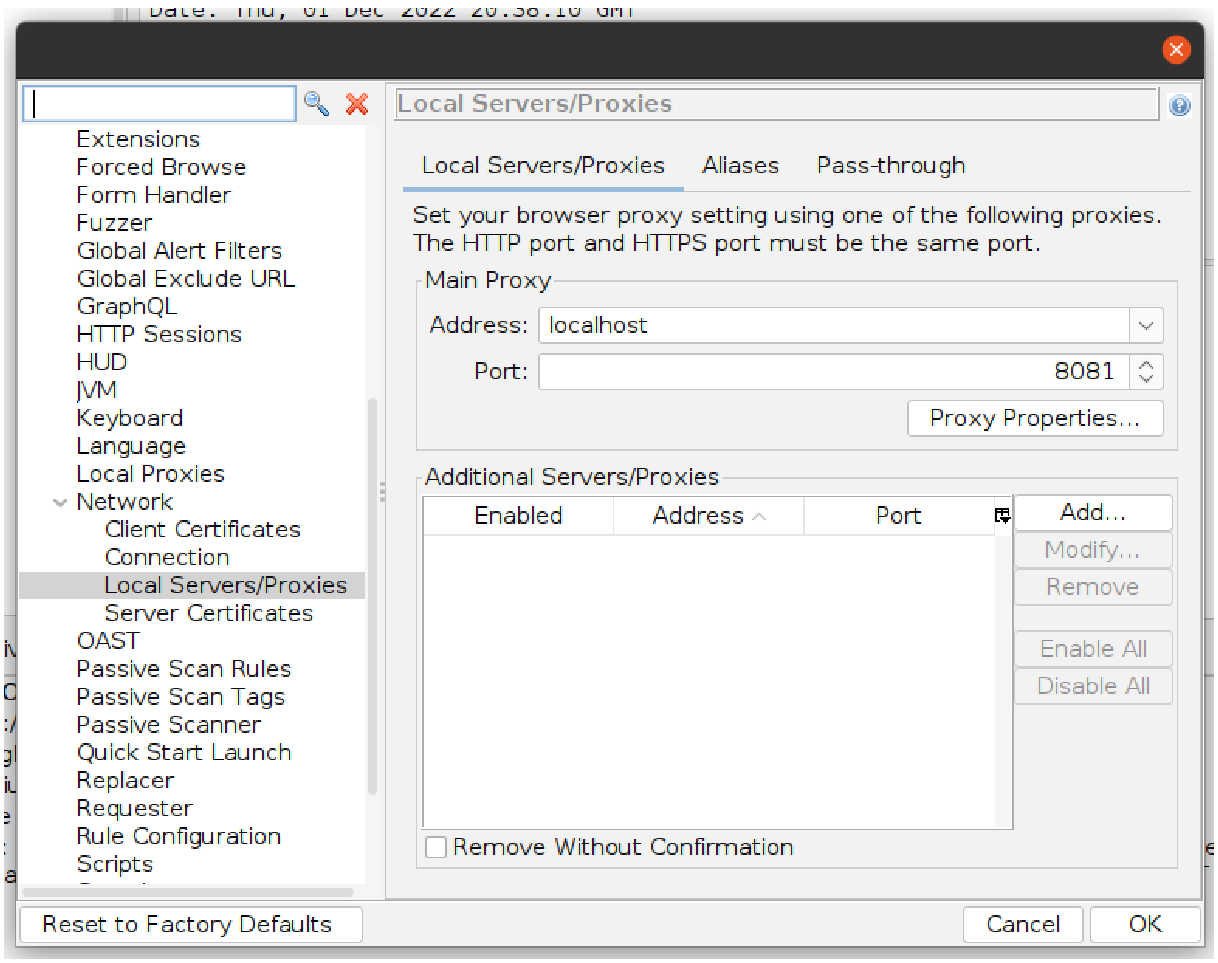
```
./ZAP_2_12_0_unix.sh
```

It will install the tool and after finishing the installation you will be able to find the tool in the Ubuntu dash under the name "OWASP ZAP".

4.1.2 Configure ZAP proxy

As a next step, we needed to configure proxy in our browser as a way for ZAP tool to intercept application communication. But as we tested an application running on tomcat server, using port **8080** and as the tool by default also uses port **8080**, as a very first thing we needed to change the port for local proxy in zap tool to some other, for example **8081**. To change port for the tool:

1. Open OWASP ZAP
2. Click on *Tools* - *Options*
3. Locate category *Network* - *Local servers/proxies*
4. Change the field *Port* to non-coliding port



Set your browser proxy setting using one of the following proxies.
The HTTP port and HTTPS port must be the same port.

Main Proxy

Address: localhost

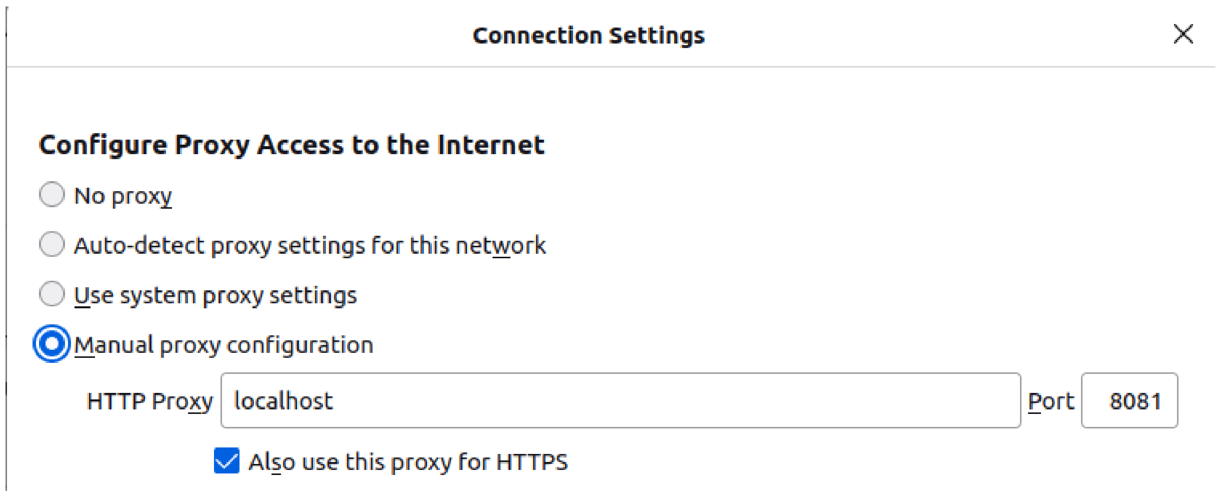
Port: 8081

☐ Remove Without Confirmation

Figure 6: Proxy port settings in ZAP tool

Now that we have ZAP tool running on a non-coliding port, we set a localhost proxy in the browser. We used Mozilla Firefox for our example. To set the proxy:

1. Open Firefox
2. Navigate to *about:preferences* url
3. Locate *Network settings* and press *Settings* button
4. Change the field *Port* to your ZAP port set in previous step (8081) and host to *localhost* for both HTTP and HTTPS


Connection Settings

Configure Proxy Access to the Internet

No proxy

Auto-detect proxy settings for this network

Use system proxy settings

[✓ Manual proxy configuration]

HTTP Proxy: localhost Port: 8081

[✓] Also use this proxy for HTTPS

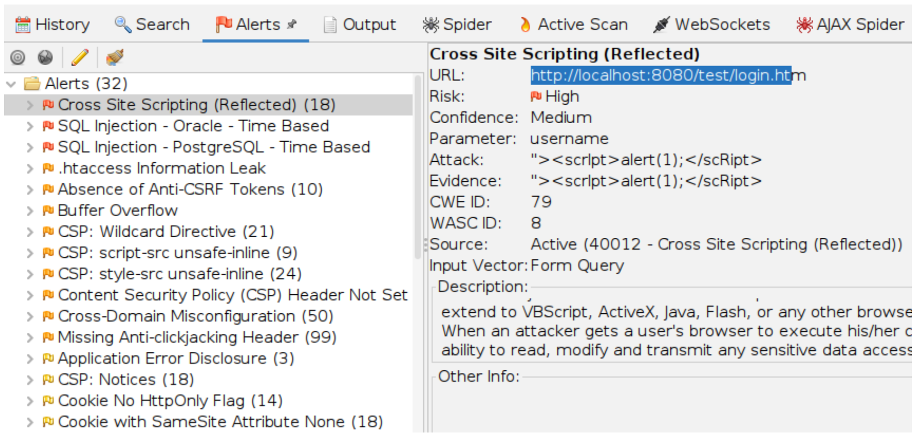
Figure 7: Proxy settings in Firefox browser

If everything worked okay, one should be able to see the ZAP overlay while have loaded any web page.

4.1.3 Prepare the test

Because the test is performed on a system, that has almost all the pages behind the authentication, we need to set it up in the tool. To set the user:

1. Start Mango app
2. Navigate to Mango login [7]
3. Login using username and password
4. In ZAP tool right click on root folder of website and choose "Include in context"
5. In history, find POST request to */test/login.shtm* and select *Flag as Context - Form-based Auth Login request*
6. Check settings of fields to parameters
7. Select some string typical for logged user in Response of logged in page, click right and click *Flag as Context - Logged in Indicator*
8. Enable *Force User Mode* clicking the small button in the second of options
9. Run spider by clicking right the root folder and clicking *Attack - Spider*
10. You are ready to run the penetration testing itself, similar to the Spider above, but selecting *Active Scan* instead of *Spider*



```
Alerts (32)
  Cross Site Scripting (Reflected) (18)
  SQL Injection - Oracle - Time Based
  SQL Injection - PostgreSQL - Time Based
  .htaccess Information Leak
  Absence of Anti-CSRF Tokens (10)
  Buffer Overflow
  CSP: Wildcard Directive (21)
  CSP: script-src unsafe-inline (9)
  CSP: style-src unsafe-inline (24)
  Content Security Policy (CSP) Header Not Set
  Cross-Domain Misconfiguration (50)
  Missing Anti-clickjacking Header (99)
  Application Error Disclosure (3)
  CSP: Notices (18)
  Cookie No HttpOnly Flag (14)
  Cookie with SameSite Attribute None (18)
```

```
Cross Site Scripting (Reflected)
URL:          http://localhost:8080/test/login.htm
Risk:         High
Confidence:   Medium
Parameter:    username
Attack:       "><scrIpt>alert(1);</scRipt>
Evidence:     "><scrIpt>alert(1);</scRipt>
CWE ID:       79
WASC ID:      8
Source:       Active (40012 - Cross Site Scripting (Reflected))
Input Vector: Form Query
Description:
 extend to VBScript, ActiveX, Java, Flash, or any other browse
 When an attacker gets a user's browser to execute his/her c
 ability to read, modify and transmit any sensitive data access
Other Info:
```

Figure 8: Penetration test results in ZAP tool

4.2 Vulnerabilities

4.2.1 Cross Site Scripting (Reflected)

Description: Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance.
Potential Risk: When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser
Url: http://localhost:8080/test/login.htm
Severity Level: High
Type of Vulnerability: CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
Recommendation for Solving the Vulnerability: Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

4.2.2 SQL Injection - Oracle - Time Based

Description: SQL injection, is a common attack that uses malicious SQL code for backend database manipulation to access or modify information that was not intended to be displayed.
Potential Risk: Modifying database records, leak of the database records
Url: http://localhost:8080/test/dwr/call/plaincall/MiscDwr.doLongPoll.dwr
Severity Level: High
Type of Vulnerability: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
Recommendation for Solving the Vulnerability: Do not trust client side input, even if there is client side validation in place. In general, type check all data on the server side.

4.2.3 SQL Injection - PostgreSQL - Time Based

Description: SQL injection, is a common attack that uses malicious SQL code for backend database manipulation to access or modify information that was not intended to be displayed.

Potential Risk: Modifying database records, leak of the database records
Url: http://localhost:8080/test/dwr/call/plaincall/MiscDwr.doLongPoll.dwr
Severity Level: High
Type of Vulnerability: CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
Recommendation for Solving the Vulnerability: Do not trust client side input, even if there is client side validation in place. In general, type check all data on the server side.

4.2.4 .htaccess Information Leak

Description: .htaccess files can be used to alter the configuration of the Apache Web Server software to enable/disable additional functionality and features that the Apache Web Server software has to offer.
Potential Risk: Modifying server configuration
Url: http://localhost:8080/test/dwr/call/plaincall/.htaccess
Severity Level: Medium
Type of Vulnerability: CWE-94: Improper Control of Generation of Code ('Code Injection')
Recommendation for Solving the Vulnerability: Ensure the .htaccess file is not accessible.

4.2.5 Absence of Anti-CSRF Tokens

Description: No Anti-CSRF tokens were found in a HTML submission form.
Potential Risk: A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim.
Url: http://localhost:8080/test/login.htm
Severity Level: Medium
Type of Vulnerability: CWE-352: Cross-Site Request Forgery (CSRF)
Recommendation for Solving the Vulnerability: Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid. For example, use anti-CSRF packages such as the OWASP CSRFGuard.

4.2.6 Buffer Overflow

Description: Buffer overflow errors are characterized by the overwriting of memory spaces of the background web process, which should have never been modified intentionally or unintentionally.
Potential Risk: Overwriting values of the IP (Instruction Pointer), BP (Base Pointer) and other registers causes exceptions, segmentation faults, and other process errors to occur. Usually these errors end execution of the application in an unexpected way.
Url: http://localhost:8080/test/data_source_edit.shtm?dsid=2
Severity Level: Medium
Type of Vulnerability: CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
Recommendation for Solving the Vulnerability: Rewrite the background program using proper return length checking. This will require a recompile of the background executable.

4.2.7 Content Security Policy (CSP) Header Not Set

Description: Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks.
Potential Risk: These attacks are used for everything from data theft to site defacement or distribution of malware.
Url: http://localhost:8080/test/login.htm
Severity Level: Medium
Type of Vulnerability: CWE-693: Protection Mechanism Failure
Recommendation for Solving the Vulnerability: Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header, to achieve optimal browser support: "Content-Security-Policy" for Chrome 25+, Firefox 23+ and Safari 7+, "X-Content-Security-Policy" for Firefox 4.0+ and Internet Explorer 10+, and "X-WebKit-CSP" for Chrome 14+ and Safari 6+.

4.2.8 Missing Anti-clickjacking Header

Description: The response does not include either Content-Security-Policy with 'frame-ancestors' directive or X-Frame-Options to protect against 'ClickJacking' attacks.
Url: http://localhost:8080/test/login.htm
Severity Level: Medium
Type of Vulnerability: CWE-1021: Improper Restriction of Rendered UI Layers or Frames
Recommendation for Solving the Vulnerability: Modern Web browsers support the Content-Security-Policy and X-Frame-Options HTTP headers. Ensure one of them is set on all web pages returned by your site/app. If you expect the page to be framed only by pages on your server (e.g. it's part of a FRAMESET) then you'll want to use SAMEORIGIN, otherwise if you never expect the page to be framed, you should use DENY. Alternatively consider implementing Content Security Policy's "frame-ancestors" directive.

4.3 Summary of the Results

The penetration testing took approximately one hour, so next time it would be probably wiser to cut test into separate categories by the features of the Mango system. Nevertheless, after taking top 8 vulnerabilities scanned by the OWASP ZAP tool, it is safe to say, that it is more than concerning, how many *High* and *Medium* severity level ones we could see present in the list. Especially the fact, that one of the most severe vulnerabilities, SQL injection, endangering the whole database is present is to be frowned upon. In combination with another huge security flaw, being able to perform Cross-Site Scripting, it is almost certain that such flawed system would never be usable in production and should be returned to developer team to a great portion of fixing and redesign.

5 Q3: Static Analysis and Penetration Testing Results Comparison

In the table below, we can see comparison of few hand-picked CWE categories of the vulnerabilities, found during Static analysis, Penetration testing or both:

CWE type	**Static analysis**	**Penetration testing**
CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	2	2
CWE-798: Use of Hard-coded Credentials	1	0
CWE-330: Use of Insufficiently Random Values	1	0
CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	0	1
CWE-352: Cross-Site Request Forgery (CSRF)	0	1

Table 1: Comparison of CWE categories

As we can see, SQL injection is found in both, so we can very basically guarantee the vulnerability actually exists. Other than SQL injection, vulnerabilities differ and are found only during one of the two security testing phases. That only emphasizes need to use both approaches, as choosing only one might not be enough.

Static analysis shows us flaws in code, such as hard-coded credentials, not enough random values, which might never be penetrated, but are still a high security risk. Static analysis shows us mostly points, where we can prevent being penetrated, and thanks to the fact that it is used in testing phase of development, it is hugely valuable, since it can help us fix security risk in pre-production time.

On the other only hand penetration testing showed us the possibility of Cross-Site Scripting attack, that poses a high risk too, and would not be discovered using static analyzer only. Even though lack of CSFR token is considered by OWASP ZAP tool as *Medium* severity flaw, it could still expose the application to severe consequences. It clearly shows that penetration testing on a pre-production (staging) environment of the application should be considered by every development team as well.

6 Conclusion

From the results of security testing, we can say with confidence that tested application (Mango) is very unsafe and should be hugely modified to be even considered ready for testing again. Huge amount of *High* and *Medium* severity security vulnerabilities is a big red flag, but also a sign that no matter how much we believe in our almighty developers, it is always very important to use a help of automated tools, to help us test the software. And it is more than clever to combine static and dynamic approach of testing the software, as we could experience how both of them can discover different important improvement points in the software.

References

[1] URL: https://helicaltech.com/introduction-to-mango-automationm2m/.

[2] URL: https://thenewstack.io/how-to-analyze-code-and-find-vulnerabilities-with-sonarqube/.

[3] URL: https://blog.sonarsource.com/sq-sc_guidance/.

[4] URL: https://docs.sonarqube.org/latest/setup/get-started-2-minutes/.

[5] URL: https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-ant/.

[6] URL: https://www.zaproxy.org/download/.

[7] URL: https://localhost:8080/test/login.shtm.