Practica Final — SEGURIDAD EN APLICACIONES

Pablo ORTOLAN

May 28^{th} 2023

Abstract

This document contains links to figures that are clickable. You can go back to the quote location by clicking the label underneath the figure.

Contents

1 Exercise 1 — Python application

1.1 What vulnerability or vulnerabilities can you find?

After analysing, the python application code contains vulnerabilities. here is a description of each one:

- Lack of data validation: The code below does not validate the input data. The variables are initialized without filtering and without examining the data provided by the user.

```
    first_name = request.POST.get('first_name')
    last_name = request.POST.get('last_name')
    password = request.POST.get('password')
    address = request.POST.get('address')
```

- Plain text and potentially weak password storage: The code uses set_password() to set the password but there is no encoding, encryption or complexity checking of it.

```
    # validacion de la contrasena de la peticion
    if password != None and password != "":
        customuser.set_password(password)
    customuser.save()
```

- Using the POST request method without CSRF protection: The code does not include protection against Cross-Site Request Forgery (CSRF) attacks. Indeed, the code below retrieves the "id" directly in the user's request without verification.

```
    customuser = CustomUser.objects.get(id=request.user.id)
```

- Error handling: The "except" block captures all exceptions without distinction and they are not saved in a suitable file.

```
    try:
        [...]
        messages.success(request, "Profile Updated Successfully")
        return redirect('student_profile')
    except:
        messages.error(request, "Failed to Update Profile")
        return redirect('student_profile')
```

1.2 What risks are involved?

The various risks associated with these vulnerabilities are:

- Lack of data validation: This can allow the injection of malicious or invalid data such as XSS. Sensitive information, such as session cookies could be stolen.

- Plain text and weak password storage: First the password is sent in plain text and stored as is in the database, so it is much more exposed to theft. Also, there is no password complexity check. An attacker could easily use brute force to crack this password.

- Using the POST request method without CSRF protection: The associated risk is that a malicious user could exploit the site's trust in a user's authentication and could change sensitive information without their knowledge.

- Error handling: The associated risk is to make debugging and fixing errors more difficult since no file holds the precise history of the activity.
  Moreover, if malicious users tried to force the website there will be no record of their tries and so it will be undetectable.

1.3 The developer does not know how to fix the problem, so please explain how to fix it

In order to help the developer here are some tips to solve these problems:

- Absence of data validation: To solve this problem, it is necessary to clean up these inputs. The use of libraries containing specific functions is common.
  For example, for Python the Validator Collection is a library that provides over 60 functions that can be used to validate the type and content of an input value.
  Here is an example implementation in pseudo code:

```
    import validator

    if request.method == "POST":
    data = validator.validate(request.POST)
    if form.is_valid():
        # The input data is valid, you can use it safely
        first_name = data.cleaned_data['first_name']
        last_name = data.cleaned_data['last_name']
        password = data.cleaned_data['password']
        address = data.cleaned_data['address']
```

- Plain text password storage: First the password should be complex enough not to be broken easily. A series of test should be performed as:
  - A minimum length of 16 characters.
  - Combination of numeric, punctuation, and alphabetic characters.
  - Do not allow anyone to submit a new password/phrase identical to one of the last four passwords/phrases used.
  - Black list of known passwords (from the dictionary)

  Next, the password must be encrypted in order to make it difficult to decrypt if intercepted. The technique to use is hashing (with salt for more security).
  Here is an example implementation for these two solutions:

```
    import password_validator
    # validacion de la contrasena de la peticion
    if password_validator.check(password):
        customuser.set_password(HASH(password))
    customuser.save()
```

- Use of the POST request method without CSRF protection: To reduce this risk, it is possible to introduce a CSRF token to be verified in the request.

- Error handling: In order to facilitate debugging, error correction and check activity, it is recommended to use a LOG file.
  Here is an implementation with some possible messages to insert in the agreed places:

```
    import logging

    logger = logging.get_logger(log_name)
    [...]
    logger.error(request, "Invalid Method")
    [...]
    logger.info(request, "Success update")
    [...]
    logger.error(request, "Failed update")
    [...]
    logger.error(request, "not valid password")
    [...]
```

2 Exercise 2 — Vuldroid.apk

In this exercise, I will study the vuldroid.apk file. It contains android application resources that are analysed with the tool MobSF.
All following questions are answered and justified thanks to this tool and the OWASP Mobile Application Security Framework and the OWASP Top Ten

2.1 Can you get the source code? Is the application properly obfuscated?

First, the source code of an APK file is the human-readable code, that defines the functionality and behavior of an Android application before it is compiled into an executable format.
In our case, MobSF allows us to open the application resources. The source code is composed of 18 java files and 1 XML file:

- The java files (fig.1) contain the instructions and logic that define the functionality and behavior of an Android application such as the authentication process, network communication etc.
- The AndroidManifest.xml file (fig.2) in an Android app provides essential information to the operating system, including the app's components, permissions, and configuration details, enabling proper installation, runtime behavior, and interaction with the Android platform and other apps.

Then, obfuscated code is intentionally modified code that is made difficult to understand and reverse-engineer, using techniques such as variable renaming and control flow obfuscation, to enhance code security and hinder reverse engineering efforts.
Here, the code provided is in plain text and easily understandable. For example, in the login.java files, variables like *inputEmail* and *inputPaassword* are undoubtedly receiving the user data input. More generally, everywhere in this project, functions name are describing their purpose.

```
public void firebaselogin(View view) {
    this.mauth = FirebaseAuth.getInstance();
    EditText inputEmail = (EditText) findViewById(R.id.loginemail_editText);
    EditText inputPaassword = (EditText) findViewById(R.id.login_password_editText);
    [...]
```

Finally, to check if the application is obfuscated, Proguard tool have to be used. In the file tree structure, no proguard or build.gradle file exists.

Consequently, the code is exposed to reverse engineering and not very secure. The application is not obfuscated.

2.2 Are all communications encrypted?

Encrypting communication is important to protect sensitive data from unauthorized access, maintain data integrity, comply with security regulations and unauthorized manipulation during transmission.

First, the NIAP Analysis (fig.3) assures that the application does encrypt some transmitted data with HTTPS/TLS/SSH between itself and another trusted IT product. The URLs and Database checker (fig.4) confirm that all known communications are using HTTPS and not HTTP.
But then the XML file contain a vulnerability concerning transmission.

The attribute *android:usesCleartextTraffic="true" (l.7)* is set in the <*application*> tag. This allows network communication over HTTP without encryption. It is generally recommended to use secure HTTPS connections to protect sensitive data during transmission. Setting this attribute to false enforces secure communication.
This vulnerability is considered "high severity" from MobSF as showed below (fig.5)

Finally, even though all URLs are using HTTPS protocol, the default behavior is not correctly configured and may lead to security exposure in the future. For instance, adding a new domain the app need to connect with will communicate with the non secure HTTP protocol.

2.3 Is sensitive information stored correctly?

According to OWASP Top 10, over the last few years, Sensitive Data Exposure has been the one of the most common impactful attack. A common flaw is simply not encrypting sensitive data. When crypto is employed, weak key generation and management, and weak algorithm, protocol and cipher usage is common, particularly for weak password hashing storage techniques. For data in transit, server-side weaknesses are mainly easy to detect, but hard for data at rest.

Concerning this application, transit data has already been investigated previously. // Here is a list of the other points at stake:

First, thanks to NIAP analysis (fig.3), some points testified of good practices such as:

- Not storing any credentials to non-volatile memory
- No access to sensitive information repositories.
- Implement functionality to encrypt sensitive data in non-volatile memory.

But, after deeper analysis, the application data is not secure properly because of a weak certificate encryption.
Indeed, the Certificate analysis exposes that the Application is signed with SHA1withRSA. (fig.7) SHA1 hash algorithm is known to have collision issues. (when two different inputs produce the same hash value.)

It is preferable to use SHA-256. It provides a higher level of security and is currently considered to be robust against collision attacks.

Then, the code analysis (fig.8) says that sensitive information is be logged and it can be a vulnerability. Moreover, the OWASP Mobile Application Security recommend to use a keyStore for Android application in order to ensure secure data storage. (fig.9)
However, the structure tree does not contain any file specifying a keyStore usage.

To conclude, any sensitive information is not guaranteed to be stored correctly.

2.4 Are there insecure permissions or settings in the MANIFEST file?

Examining the provided AndroidManifest.xml, here are some potential vulnerabilities or security concerns:

- Debuggable Application: The attribute *android:debuggable="true" (l.7)* is set in the *<application>* tag. This means that the application can be debugged, which could provide opportunities for attackers to inspect the application's code, uncover vulnerabilities, and extract sensitive information. In a production environment, it is recommended to set this attribute to false.
- Cleartext Traffic: As said in question 2 (fig.5), the attribute *android:usesCleartextTraffic="true" (l.7)* is set in the *<application>* tag. This allows network communication over HTTP without encryption. It is generally recommended to use secure HTTPS connections to protect sensitive data during transmission. Setting this attribute to false enforces secure communication.
- Permissions: The following permissions are declared: *android.permission.READEXTERNALSTORAGE*: This permission allows the application to read external storage. If not necessary, granting unnecessary file access permissions can pose a security risk.
- *android.permission.INTERNET*: This permission allows the application to access the internet. It is necessary if the app requires network connectivity, but it is essential to ensure that network requests are secure and handled properly to prevent unauthorized access or data leakage.
- Exported Components: Some activities and the receiver are marked as *exported="true"*. Exported components can be accessed by other applications or components if not properly secured or if the functionality is not intended to be publicly accessible. It is crucial to review whether these components need to be exported and, if not, set *exported="false"*.
- Implicit Intent Filters: Some activities have intent filters that specify actions and categories without explicit package names. Implicit intent filters can potentially allow other applications to invoke these activities, which can lead to security risks if not handled properly. It is recommended to specify the package name in explicit intent filters.
- Firebase-related Activities and Providers: The manifest includes activities and providers related to Firebase authentication and initialization. While these components may not inherently be vulnerable, it's crucial to ensure that Firebase configurations are properly secured and that relevant security best practices are followed.

To conclude, there are definitely insecure settings in the MANIFEST file. The problems found with high and medium (warning) criticality can be resolved following the recommendation above for each vulnerability.

3 Exercise 3 — Blues Breaker Bank Audit

This report describes the results of a thorough independent code audit of the Blues Breaker Bank application. In the course of the code review, Blues Breaker Bank provided full access to the source code of the app.

Overall, some high risk or critical vulnerabilities were discovered in Blues Breaker Bank app. Several low to medium risk issues were identified. In summary, Blues Breaker Bank need to mitigate the following vulnerabilities in order to increase its security and privacy features.

3.1 Executive summary content

3.1.1 Introduction

The audited application, Blues Breaker Bank app is a typical bank app where users, employees and admins can connect. Depending on their access authorization, they can access accounts and make transfers.
The audit objective is to reflects a adequate summary of the vulnerabilities found.

3.1.2 Methodology

During the audit, various methods and techniques is used.

First, a SAST — Static application security testing is conducted. Using the IDE ECLIPSE with Spotbugs and some interesting plugins, this methods tends to identify security vulnerabilities and weaknesses in the source code of an application.
Consequently, it is very efficient before an application is deployed or released because identifying and fixing security issues early in the software development life cycle can save significant costs.
Moreover, even when the app is released, it brings Continuous Security Improvement. As applications evolve and new features are added, SAST can be employed to identify security weaknesses introduced by changes to the codebase. This ongoing security assessment helps ensure that the application remains secure throughout its lifecycle.

Then, a SCA — Software Composition Analysis is conducted. Using Dependency Check, a report is produced allowing to identify and manage open source and third-party components used in the application. SCA is essential for identifying and managing security vulnerabilities, legal risks, and quality issues associated with open source and third-party components.

3.1.3 Key Findings

In the following part, the list of the major vulnerabilities categories is dressed. Both SAST and SCA are giving general insight on the security status of the application.

3.1.4 SAST — Static application security testing

After first Spotbugs analysis, they were 405 potential vulnerabilities identified. (fig.6).
In the following section, some of them are confirmed to be vulnerabilities and will be classified into the OWASP Top Ten categories. The rest are false positives.

In this report, they are classified according to the OWASP Top 10 Application Security Risks - 2017. Here is the summary of the identified vulnerabilities:

Vulnerability	Criticality
A1:2017-Injection	HIGH
A2:2017-Broken Authentication	LOW
A3:2017-Sensitive Data Exposure	HIGH
A4:2017-XML External Entities (XXE)	HIGH
A5:2017-Broken Access Control	HIGH
A6:2017-Security Misconfiguration	HIGH
A7:2017-Cross-Site Scripting (XSS)	LOW
A8:2017-Insecure Deserialization	MEDIUM
A9:2017-Using Components with Known Vulnerabilities	LOW
A10:2017-Insufficient Logging & Monitoring	LOW

3.1.5 SCA — Software Composition Analysis

After dependency-check analysis and dropping the false positive, here is a summary of the vulnerabilities. The Common Vulnerabilities and Exposures (CVE) is a standardized identifier used to uniquely identify a specific software vulnerability or security weakness.

Dependency name	CVE	Criticality	Main vulnerabilities
jackson-databind	2	HIGH	Deserialization of Untrusted Data
postgresql_jdbc_driver	2	HIGH-	Improper Neutralization of elements used in SQL Insecure Temporary File Exposure of Sensitive Information
snakeyaml	7	MEDIUM+	Deserialization of Untrusted Data Improper Input Validation Uncontrolled Resource Consumption
apache_tomcat	6	MEDIUM	Uncontrolled Resource Consumption Inconsistent Interpretation of HTTP Requests Improper Neutralization of Special Elements Unprotected Transport of Credentials Concurrent Execution using Shared Resource

3.2 Technical content

3.2.1 Introduction

The audited application, Blues Breaker Bank app is structured as a combination of folder.(fig.10)
Here is the main functions of them:

- db folder: Contain all the information about the database construction and configuration.

- Gradle folder: Gradle is a build automation tool known for its flexibility to build software. A build automation tool is used to automate the creation of applications. The building process includes compiling, linking, and packaging the code.
- src folder: Contain all the java source code of the application itself and the resources needed.
- target folder: It refers to the directory where the compiled and built files are generated during the build process. It is a standard convention used in many Java build tools and frameworks, such as Maven and Gradle.
- Various files can be found outside any folders helping understanding the application and setting the environment such as HELP.md, pom.xml, build_and_push.sh, Dockerfile and setting files.

3.2.2 Security Analysis

During the analysis, critical vulnerabilities are identified and false positive are reported. Below is a detailed explanation and potential impact of the application security.

3.2.3 SAST — Static application security testing

In this section, the Top 10 OWASP methodology is applied. All vulnerabilities found are reported in the right category.

3.2.3.1 A1:2017-Injection

CRLF Injection (Carriage Return Line Feeds)

- **Criticality**: Medium
- **Occurrence**: >1
- **Description**:
  When data from an untrusted source is put into a logger and not neutralized correctly, an attacker could forge log entries or include malicious content.
  If the log file is processed automatically, the attacker can render the file unusable by corrupting the format of the file or injecting unexpected characters.
- **Vulnerable File**:
  BluesBreakerBank/src/main/java/com/bluesbreaker/bank/mvc/services/UserServiceImpl.java
- **Vulnerable Code**:

```
1    public UserDetails loadUserByUsername(final String username)
2        throws UsernameNotFoundException {
3        User user = this.userRepository.getUserByUsername(username);
4        logger.debug("username:" + username.toString());
5        }
```

- **Bug info**:
  - *Rank*: Scary (5)

  - *Confidence*: High
  - *Pattern*: SPRING_CSRF_PROTECTION_DISABLED
  - *Type*: SECSPRCSRFPD
  - *Category*: SECURITY (Security)

- **Mitigation**:

  It is recommended to sanitize the input.

SQL Injection:

- **Criticality**: High
- **Occurrence**: >1
- **Description**:
  The vulnerability arises from the concatenation of the username parameter directly into the SQL query string.
  This means that if an attacker provides a malicious input as the username, they can manipulate the SQL query and potentially execute arbitrary SQL commands.
  For example, if an attacker provides the username value as ' OR 1=1 –, he can gain unauthorized access to sensitive information or perform other malicious actions.
- **Vulnerable File**:
  BluesBreakerBank/src/main/java/com/bluesbreaker/bank/mvc/services/UserServiceImpl.java
- **Vulnerable Code**:

```
1     public List<User> findByUsername(final String username) {
2         String sql = "SELECT user_id, enabled, name, password, path_avatar,
    surname,
3         username " + "FROM public.users where username='" + username + "';";
```

- **Bug info**:

  - *Rank*: Scary (7)
  - *Confidence*: Normal
  - *Pattern*: SQL_INJECTION_SPRING_JDBC
  - *Type*: SECSQLISPRJDBC
  - *Category*: SECURITY (Security)

- **Mitigation**:

  It is recommended to validate all input parameters to the application before processing, based on the use of white lists (accepting only valid data) through the use of regular expressions, patterns and/or validation schemes.
  Bind variables in prepared statements can be used to easily mitigate the risk of SQL injection.

3.2.3.2 A3:2017-Sensitive Data Exposure

Possible information exposure through error message

- **Criticality**: Low
- **Occurrence**: >1
- **Description**:
  The function prints the stack trace to the console.
  This is non configurable, and causes an application to look unprofessional.
- **Vulnerable File**:
  BluesBreakerBank/src/main/java/com/bluesbreaker/bank/mvc/controllers/MainController.java
- **Vulnerable Code**:

```
1    catch (FileNotFoundException e) {
2        e.printStackTrace();
3        logger.error("Error not found image");
4    }
```

- **Bug info**:
  - *Rank*: Of Concern (17)
  - *Confidence*: Normal
  - *Pattern*: IMC_IMMATURE_CLASS_PRINTSTACKTRACE
  - *Type*: IMC
  - *Category*: STYLE (Dodgy code)
- **Mitigation**:

  As logger is also used here so that users can control what is logged and where, it is recommended to delete that line.

Plain Text Password

- **Criticality**: High
- **Occurrence**: 1
- **Description**:
  Storing a password in plaintext may result in a system compromise.
  It is stored in plaintext in an application's properties or configuration file.
  It allows anyone who can read the file access to the password-protected resource.
- **Vulnerable File**:
  BluesBreakerBank/src/test/resources/application-test.properties
- **Vulnerable Code**:

```
spring.datasource.url = bluesjdbc:postgresql://localhost:5432/bluesbreakerbank
spring.datasource.username=bluesbreakerbankuser
spring.datasource.password=xRfUgQB26X7uyeB
```

- **Mitigation**:

  Good password management guidelines require that a password never be stored in plaintext. Encode the password in a remote database.

3.2.3.3 A5:2017-Broken Access Control

Post Authorization, incorrect authorization management:

- **Criticality**: High
- **Occurrence**: 1
- **Description**:
  Incorrect definition the moment of authorization of the resource, since it is carried out afterwards.
- **Vulnerable File**:
  BluesBreakerBank/src/main/java/com/bluesbreaker/bank/mvc/api/ApiController.java
- **Vulnerable Code**:

```
@PostMapping("/maketransfer")
@Operation(security = @SecurityRequirement(name = "basicAuth"))
@ResponseBody
@PostAuthorize("hasAnyRole('ADMIN','USER','EMPLOYEE')")
  public ResponseEntity<Map<String, String>> makeTransfer(@RequestBody final TransferDto transferdto) {
  [...]
  }
```

- **Mitigation**:

  The correction is simple, authorization must be made before the use of the resource, using '@PreAuthorize'

Path transversal:

- **Criticality**: Critical
- **Occurrence**: >1
- **Description**:
  Path Traversal refers to a security vulnerability where an attacker can access files or directories outside of the intended scope or designated directory structure.
  Here, the API reads a file whose location might be specified by user input.
  The filename comes from an input parameter. If an unfiltered parameter is passed to this file API, files from an arbitrary file system location could be read.

- **Vulnerable File**:
  BluesBreakerBank/src/main/java/com/bluesbreaker/bank/facade/StorageFacadeImpl.java
- **Vulnerable Code**:

```
    @Service
    public class StorageFacadeImpl implements StorageFacade {
        [...]
        @Override
        public boolean exists(final String fileName) {
            File file = new File(url + fileName);
            return file.exists();}
        [...]
    }
```

- **Bug info**:
  - *Rank*: Scary (7)
  - *Confidence*: Normal
  - *Pattern*: PATH_TRAVERSAL_IN
  - *Type*: SECPTI
  - *Category*: SECURITY (Security)
- **Mitigation**:
  Prefer working without user input when using file system calls.
  Ensure the user cannot supply all parts of the path.
  Validate the user's input by only accepting known good.
  If forced to use user input for file operations, normalize the input before using in file io API's, such as normalize().

3.2.3.4 A6:2017-Security Misconfiguration

Insecure Random Object:

- **Criticality**: High
- **Occurrence**: 1
- **Description**:
  Random() constructor without a seed is insecure because it defaults to an easily guessable seed.
- **Vulnerable File**:
  BluesBreakerBank/src/main/java/com/bluesbreaker/bank/utils/Utils.java
- **Vulnerable Code**:

```
    public static Integer getRandomNumberUsingNextInt(int min, int max) {
        Random random = new Random();
        return random.nextInt(max - min) + min;
    }
```

- **Bug info**:
  - *Rank*: Troubling (10)
  - *Confidence*: Low
  - *Pattern*: MDM_RANDOM_SEED
  - *Type*: MDM
  - *Category*: CORRECTNESS (Correctness)
- **Mitigation**:

  It is recommended to use 'new Random(new SecureRandom().nextLong())' or 'new SecureRandom()' instead.

Disabled CSRF protection

- **Criticality**: High
- **Occurrence**: 1
- **Description**:
  Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated.
  It can force the user to perform state changing requests like transferring funds, changing their email address, and so forth.
  If the victim is an administrative account, CSRF can compromise the entire web application.
- **Vulnerable File**:
  BluesBreakerBank/src/main/java/com/bluesbreaker/bank/config/ConfigSecurityWeb.java
- **Vulnerable Code**:

```
1    protected void configure(HttpSecurity http) throws Exception {
2    http
3    [...]
4    .and()
5        .csrf().disable();
6    [...]
7    }
```

- **Bug info**:
  - *Rank*: Scary (5)
  - *Confidence*: High
  - *Pattern*: SPRING_CSRF_PROTECTION_DISABLED
  - *Type*: SECSPRCSRFPD
  - *Category*: SECURITY (Security)

- **Mitigation**:

  It is recommended to implement a CSRF token solution.
  CSRF tokens should be: Unique per user session, Secret, Unpredictable (large random value generated by a secure method).
  CSRF tokens prevent CSRF because without a token, an attacker cannot create valid requests to the back-end server.
  For the Synchronised Token Pattern, CSRF tokens should not be transmitted using cookies.

Excessive Session Time

- **Criticality**: Medium
- **Occurrence**: 1
- **Description**:
  Session timeout represents the event occurring when a user does not perform any action on a web site during an interval. The event, on the server side, changes the status of the user session to 'invalid' and instructs the web server to destroy it. Here, the 7 days session time increase Exposure to Attacks such as Session Hijacking.
- **Vulnerable File**:
  BluesBreakerBank/src/main/java/com/bluesbreaker/bank/config/ConfigSecurityWeb.java
- **Vulnerable Code**:

```
1     private static final Integer SESSION_TIME = 7 * 24 * 60 * 60; // expiration
    time: 7 days
```

- **Bug info**:
  - *Rank*: Scary (5)
  - *Confidence*: High
  - *Pattern*: SPRING_CSRF_PROTECTION_DISABLED
  - *Type*: SECSPRCSRFPD
  - *Category*: SECURITY (Security)
- **Mitigation**:

  Exactly how long you should set an inactivity timeout again depends on the nature of the data your system holds, but OWASP gives some general advice below: "OWASP recommends application builders to implement short idle time outs (2-5 minutes) for applications that handle high-risk data, like financial information.

Disabled HSTS

- **Criticality**: High
- **Occurrence**: 1

- **Description**:
  The HTTP Strict Transport Security (HSTS) header is a mechanism that web sites have to communicate to the web browsers that all traffic exchanged with a given domain must always be sent over https, this will help protect the information from being passed over unencrypted requests.
- **Vulnerable File**:
  BluesBreakerBank/src/main/java/com/bluesbreaker/bank/config/ConfigSecurityWeb.java
- **Mitigation**:

  The HTTP strict transport security header uses two directives:

  - max-age: to indicate the number of seconds that the browser should automatically convert all HTTP requests to HTTPS.
  - includeSubDomains: to indicate that all web application's sub-domains must use HTTPS.

3.2.3.5 A8:2017-Insecure Deserialization

Insecure Deserialization

- **Criticality**: High
- **Occurrence**: 1
- **Description**:
  The application deserializes untrusted data without sufficiently verifying that the resulting data will be valid.
  Data which is untrusted cannot be trusted to be well formed. Malformed data or unexpected data could be used to abuse application logic, deny service, or execute arbitrary code, when deserialized.
- **Vulnerable File**:
  BluesBreakerBank/src/main/java/com/bluesbreaker/bank/mvc/controllers/MainController.java
- **Vulnerable Code**:

```
1     try {
2         Utils.writeResponse(new FileInputStream(file), response.getOutputStream()
    );
3         logger.info("Loaded image");
4     }
```

- **Bug info**:
  - *Rank*: Of Concern (18)
  - *Confidence*: Normal
  - *Pattern*: IOI_USE_OF_FILE_STREAM_CONSTRUCTORS
  - *Type*: IOI
  - *Category*: PERFORMANCE (Performance)
- **Mitigation**:

  A deserialization library could be used which provides a cryptographic framework to seal serialized data.

3.2.4 SCA — Software Composition Analysis

In this section, the tool Dependency-check allowed us to perform a Software Composition Analysis (SCA) in order to identify and manage third-party dependencies and their associated security vulnerabilities in our Blues Breaker Bank application.
Over the 133 dependency scanned, 45 potential vulnerabilities were found in 11 different ones. (fig.11)
For each potential vulnerability (CVE-YYYY-XXXXX), the process to identify false positive is identical. After researching the dependency name in the pom.xml or the dependency tree (fig.12.13) to confirm its utilisation, the provider is identified. Then, the comparison between the application CPE and the NIST CPE with the right version is performed. If the CPE is found in the Known Affected Software Configurations section, then the vulnerability exists, otherwise it is a false positive.
After reviewing the entire list, 18 vulnerabilities are confirmed and 27 are false positives.(fig.14) In total, 8 are considered High Vulnerability metric, 8 are Medium and 1 is Low.

Here is a description of the vulnerable dependency, their potential impact and mitigation:

- jackson-databind: CVE-2022-42003 / CVE-2022-42004
  - Dependency purpose: It provides functionalities for reading and writing JSON data. Its main purpose is to enable the conversion of Java objects to JSON and vice versa.
  - Threat: Deserialization of Untrusted Data
  - Mitigation: It is recommended to add check in ‘BeanDeserializer._deserializeFromArray()‘ to prevent the use of deeply nested arrays
- postgresql_jdbc_driver: CVE-2022-31197 / CVE-2022-41946
  - Dependency purpose: It enables Java applications to connect and interact with a PostgreSQL database.
  - Threat:
    - Improper Neutralization of Special Elements used in an SQL Command (’SQL Injection’). This allows a malicious table with column names that include statement terminator to be parsed and executed as multiple separate commands.
    - Insecure Temporary File and Exposure of Sensitive Information to an Unauthorized Actor. This allows to create a temporary file if the InputStream is larger than 51k which is readable by other users on Unix like systems.
  - Mitigation:
    - Fixing SQL generated to escape column identifiers so as to prevent SQL injection. Adding escapeIdentifier() function.
    - Update your java version up to 1.7 or specify the java.io.tmpdir system environment variable to a directory that is exclusively owned by the executing user.
- snakeyaml: CVE-2022-1471 / CVE-2022-25857 / CVE-2022-38749 / CVE-2022-38751 / CVE-2022-38752 / CVE-2022-41854 / CVE-2022-38750
  - Dependency purpose: It is an open-source library providing a simple and efficient way to parse, generate, and manipulate YAML data.

  - Threat:
    * Deserialization of Untrusted Data and Improper Input Validation: SnakeYaml's Constructor() class does not restrict types which can be instantiated during deserialization. Deserializing YAML content provided by an attacker can lead to remote code execution.
    * Uncontrolled Resource Consumption: Risk of DoS attack on the amount of aliases for collections (sequences and mappings)
    * Out-of-bounds Write and Stack-based Buffer Overflow for many open unmatched brackets
  - Mitigation:
    * It is recommended to: Sanitize, Use SafeConstructor if possible (or migrate to SnakeYAML Engine), Restrict data input by limiting size, anchors, etc.
    * It is recommended to restrict nested depth using *private int nestingDepthLimit = 50;*
    * It is recommended to upgrade the snakeyaml packages

- apache_tomcat: CVE-2022-29885 / CVE-2022-42252 / CVE-2022-45143 / CVE-2022-34305 / CVE-2023-28708 / CVE-2021-43980
  - Dependency purpose: It is an open-source web server and servlet container. Its main purpose is to provide a reliable and efficient environment for running Java web applications.
  - Threat:
    * Insufficient Information and Uncontrolled Resource Consumption about the EncryptInterceptor. It does not protect against all risks associated with running over any untrusted network, particularly DoS risks.
    * Inconsistent Interpretation of HTTP Requests ('HTTP Request/Response Smuggling'). Tomcat's default configuration allowed ignoring invalid headers, enabling request smuggling attacks if behind a misconfigured reverse proxy.
    * Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection'): The JsonErrorReportValve did not properly escape user-provided data in type, message, and description values, allowing potential manipulation of the JSON output.
    * The Form authentication example in the examples web application displayed user provided data without filtering, exposing a XSS vulnerability.
    * Unprotected Transport of Credentials: When the RemoteIpFilter is used with reverse proxy requests containing the X-Forwarded-Proto header set to "https", Tomcat's session cookies lacked the secure attribute, potentially causing the user agent to transmit the cookie over an insecure channel.
    * Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition'): The simplified implementation of blocking reads and writes in Tomcat 10 and Tomcat 9.0.47 introduced a concurrency bug that could cause client connections to share an Http11Processor instance, leading to responses or parts of responses being received by the wrong client.
  - Mitigation:
    * If it requires full protection then switch to an alternative solution such as running the clustering communication over a VPN.
    * Ensure rejectIllegalHeader is set to "true".

* Remove the examples web application as documented in the Tomcat security guide.
* Upgrade version.

4 Figures

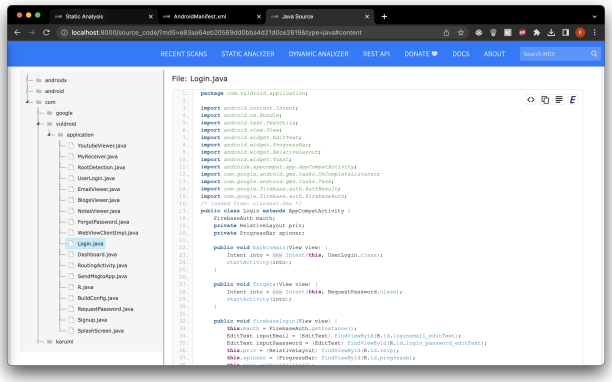

Figure 1: Source code Java files

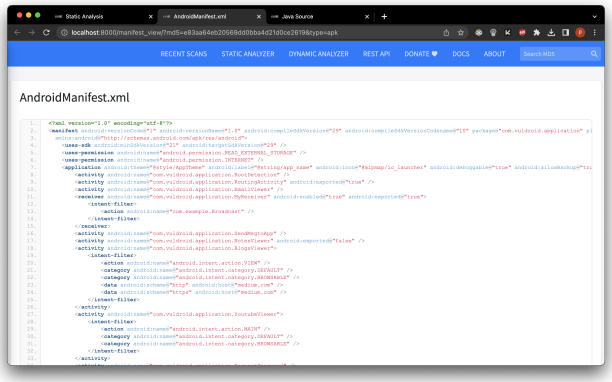

Figure 2: XML file

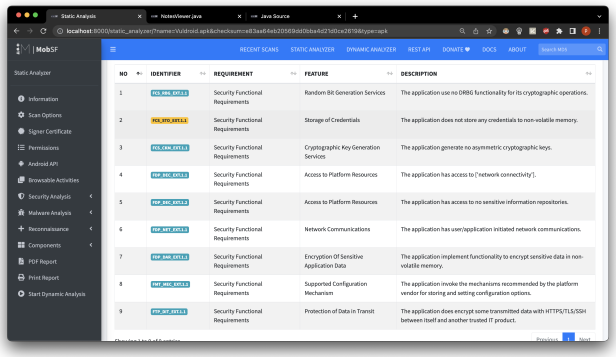

Figure 3: NIAP Analysis — Transmission

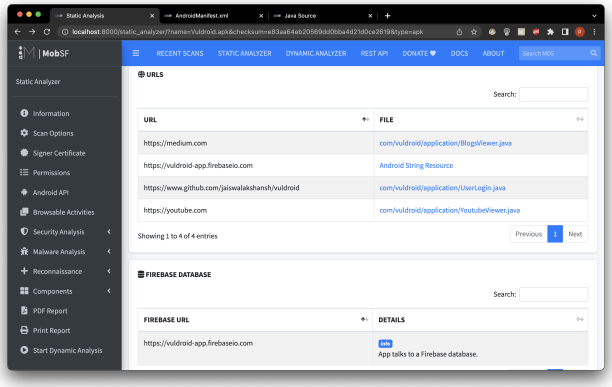

Figure 4: URLs and Database checker

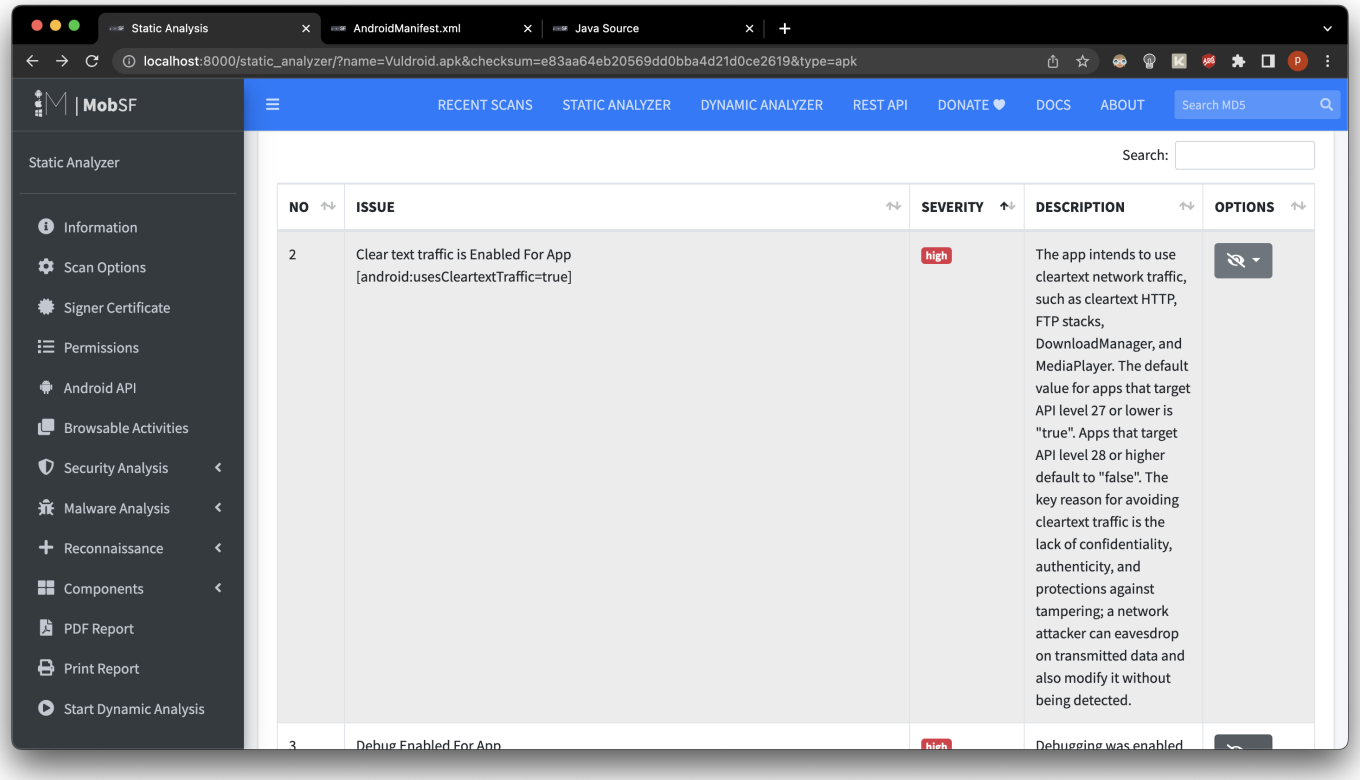

Figure 5: Clear Text Traffic vulnerability

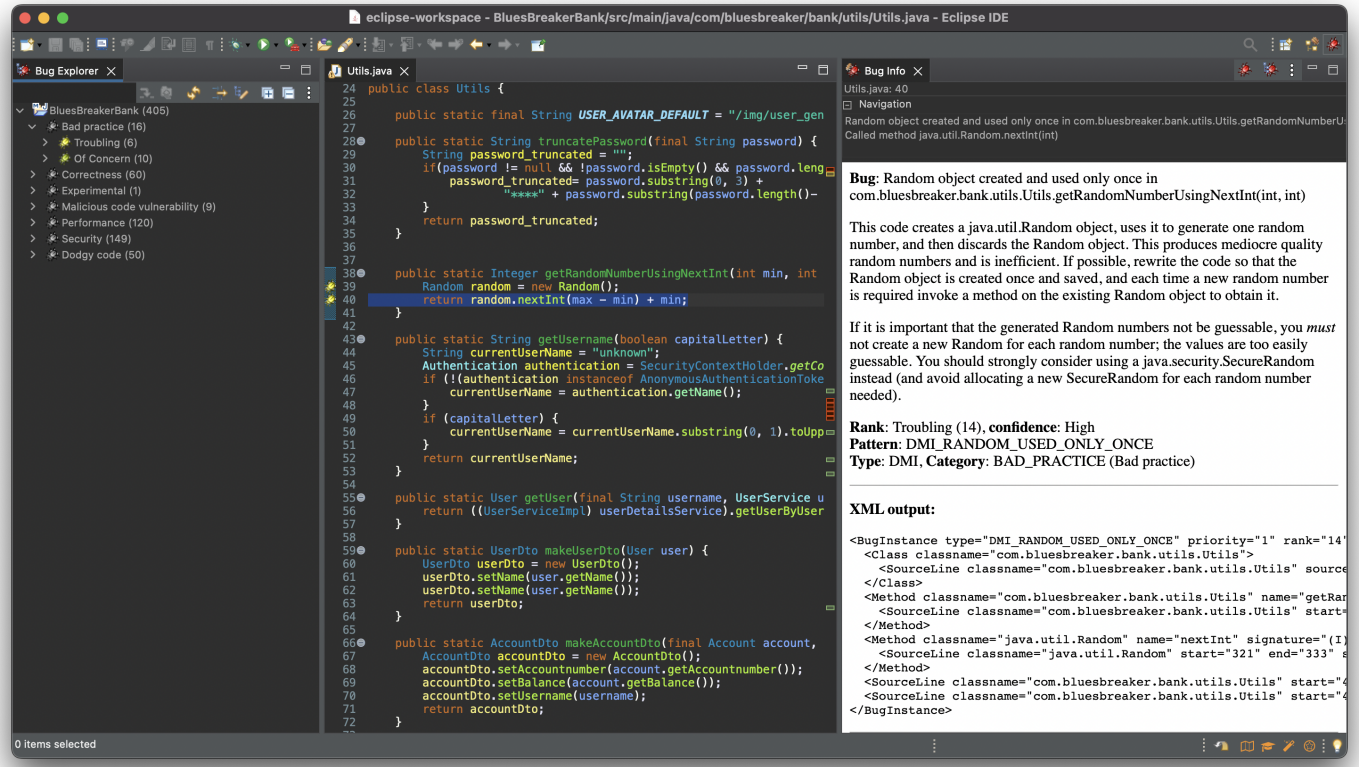

Figure 6: Eclipse IDE

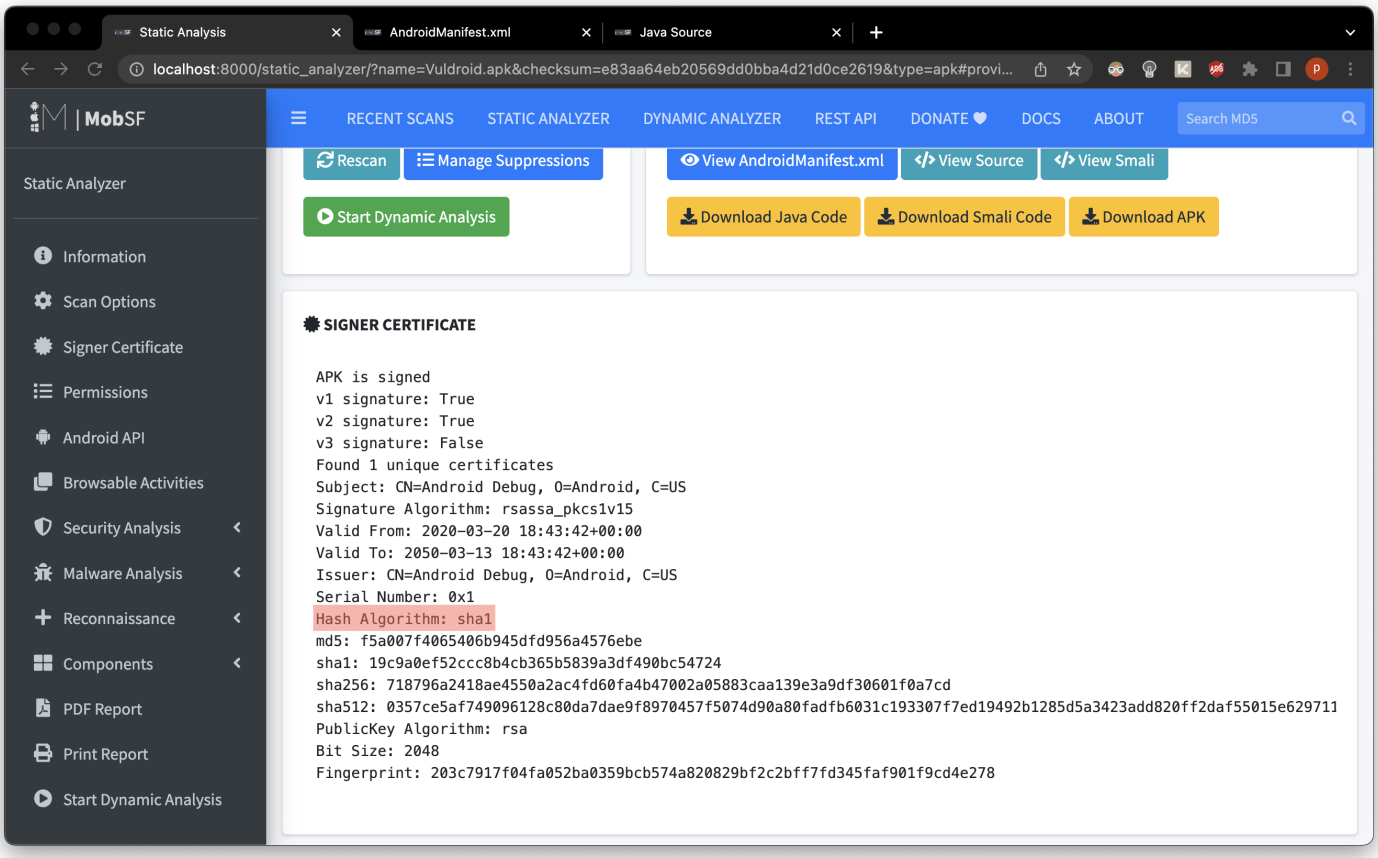

Figure 7: Certificate signed with SHA1

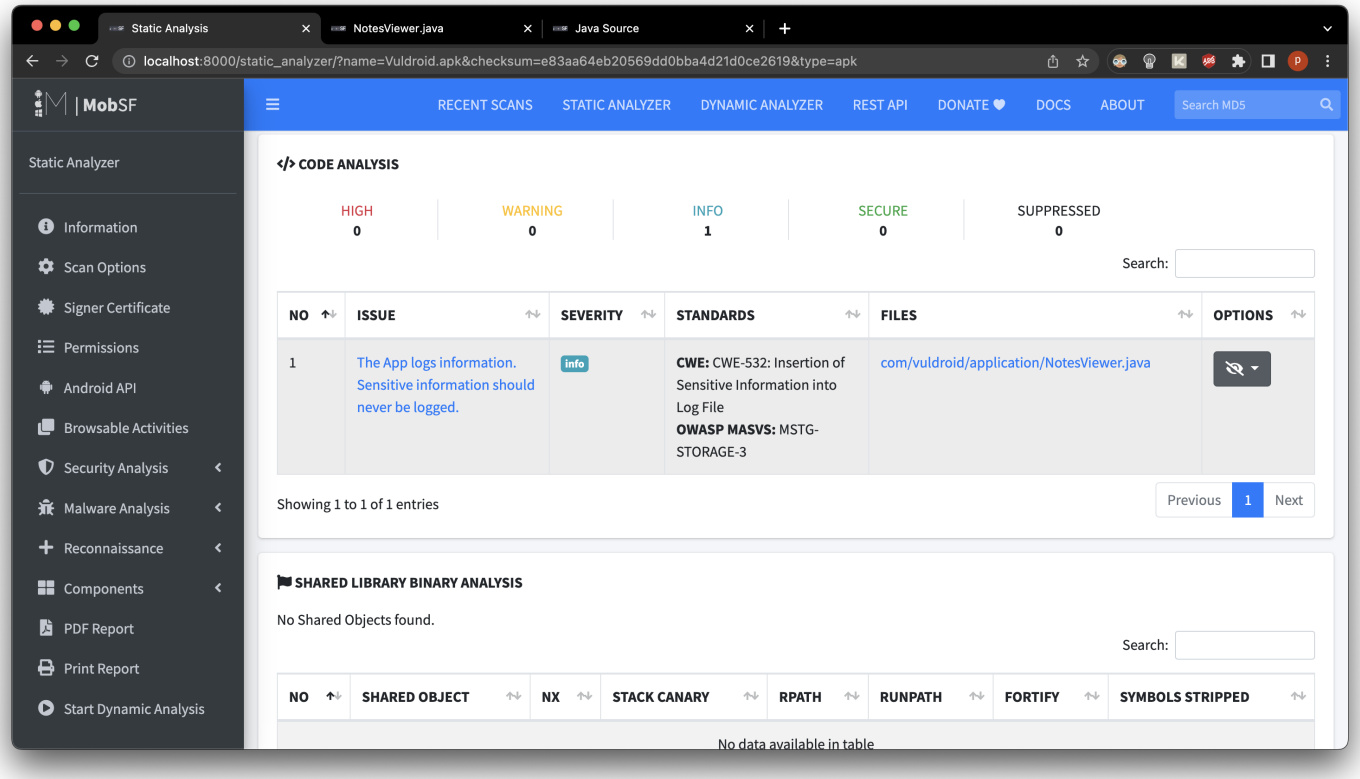

Figure 8: Code analysis — Sensitive information logged

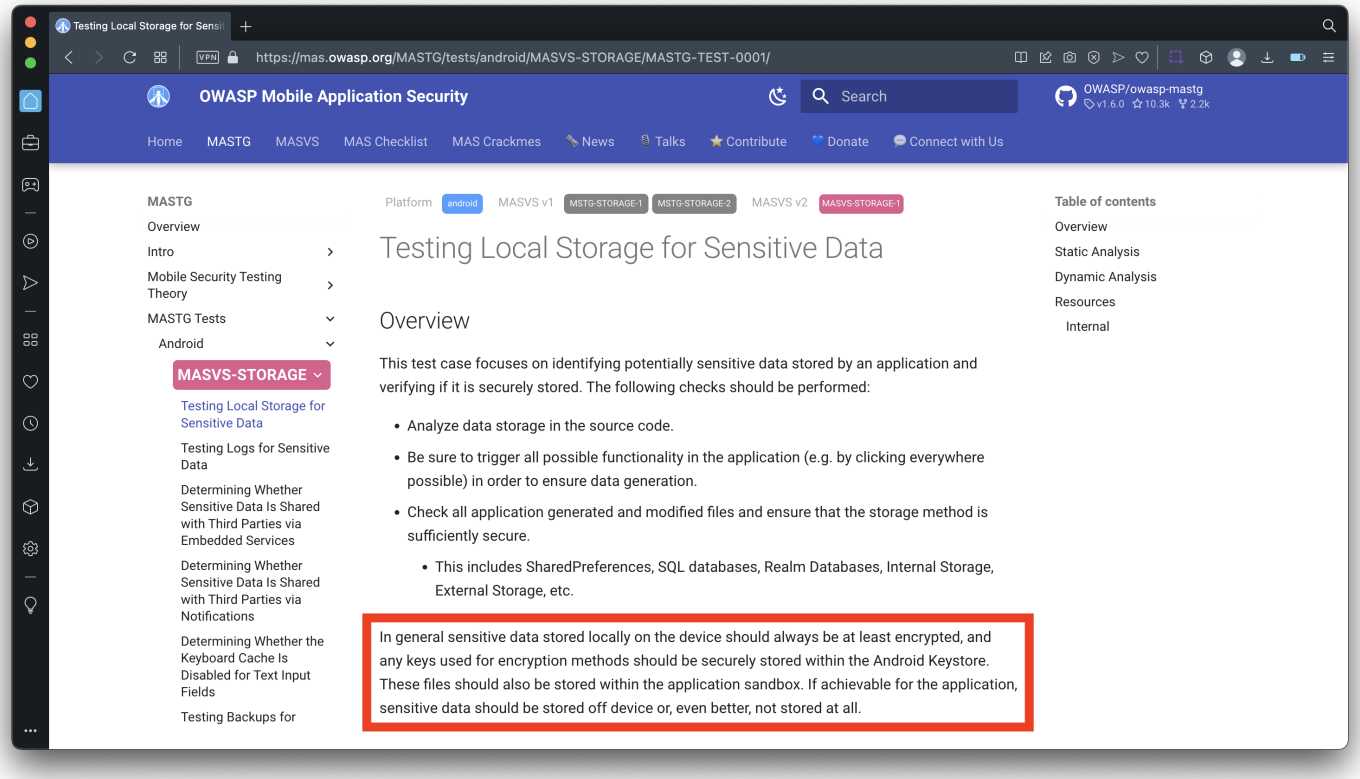

Figure 9: Sensitive data storage OWASP Mobile Application Security

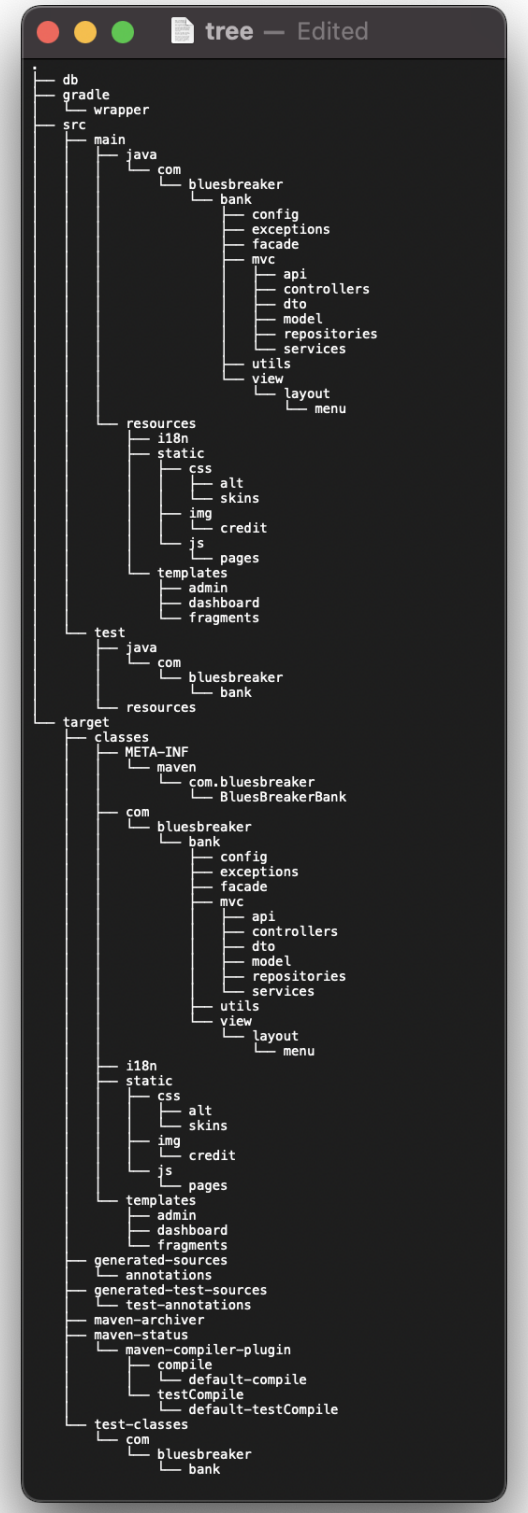

Figure 10: Application Folder tree

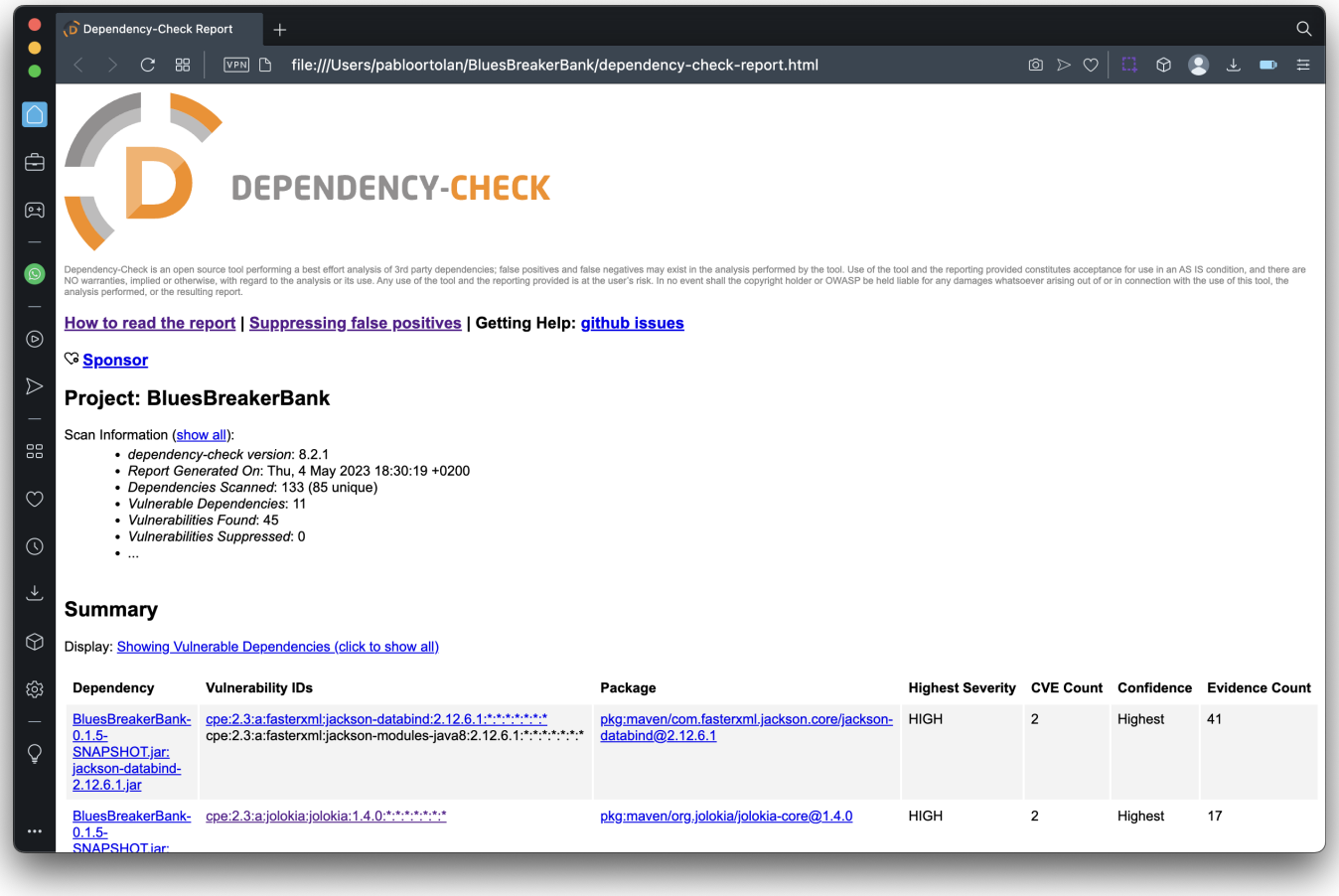

Figure 11: Dependency-Check report

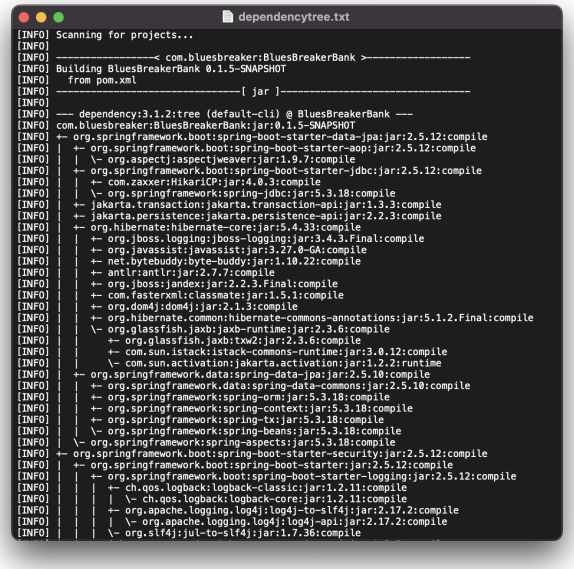

Figure 12: Dependency tree

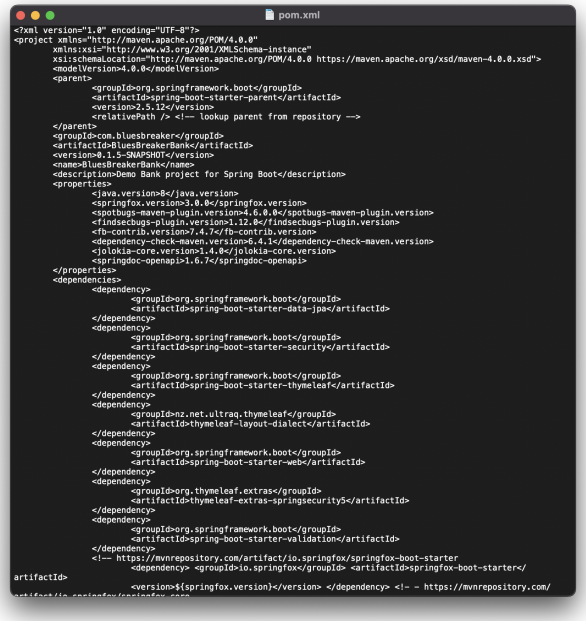

Figure 13: POM file

Dependency	CPE	CVE	Type	Base score	Probador reported by NIST
BluesBreakerBank-0.1.5-SNAPSHOT.jar: jackson-databind-2.12.6.1.jar	cpe:2.3:a:fasterxml:jackson-databind:2.12.6.1	CVE-2022-42003	vulnerable	7.5 HIGH	cpe:2.3:a:fasterxml:jackson-databind:2.12.6.1
BluesBreakerBank-0.1.5-SNAPSHOT.jar: jackson-databind-2.12.6.1.jar	cpe:2.3:a:fasterxml:jackson-databind:2.12.6.1	CVE-2022-42004	vulnerable	7.5 HIGH	cpe:2.3:a:fasterxml:jackson-databind:2.12.6.1
BluesBreakerBank-0.1.5-SNAPSHOT.jar: jolokia-core-1.4.0.jar	cpe:2.3:a:jolokia:jolokia:1.4.0	CVE-2018-10899	vulnerable	8.8 HIGH	cpe:2.3:a:jolokia:jolokia:1.4.0
BluesBreakerBank-0.1.5-SNAPSHOT.jar: jolokia-core-1.4.0.jar	cpe:2.3:a:jolokia:jolokia:1.4.0	CVE-2018-1000129 (OSSINDEX) DISPUTED	false positive		cpe:2.3:a:jolokia:jolokia:1.3.7
BluesBreakerBank-0.1.5-SNAPSHOT.jar: postgresql-42.2.25.jar	cpe:2.3:a:postgresql:postgresql_jdbc_driver:42.2.25	CVE-2022-26520 (OSSINDEX) DISPUTED	false positive		42.1.0 — 42.1.4 and 42.3.0 — 42.3.3
BluesBreakerBank-0.1.5-SNAPSHOT.jar: postgresql-42.2.25.jar	cpe:2.3:a:postgresql:postgresql_jdbc_driver:42.2.25	CVE-2022-31197	vulnerable	8.0 HIGH	cpe:2.3:a:postgresql:postgresql_jdbc_driver:42.2.25
BluesBreakerBank-0.1.5-SNAPSHOT.jar: postgresql-42.2.25.jar	cpe:2.3:a:postgresql:postgresql_jdbc_driver:42.2.25	CVE-2022-41946	vulnerable	5.5 MODERATE	cpe:2.3:a:postgresql:postgresql_jdbc_driver:42.2.25
BluesBreakerBank-0.1.5-SNAPSHOT.jar: snakeyaml-1.28.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28	CVE-2022-1471	vulnerable	9.8 HIGH	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28
BluesBreakerBank-0.1.5-SNAPSHOT.jar: snakeyaml-1.28.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28	CVE-2022-25857	vulnerable	7.5 HIGH	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28
BluesBreakerBank-0.1.5-SNAPSHOT.jar: snakeyaml-1.28.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28	CVE-2022-38749	vulnerable	6.5 MODERATE	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28
BluesBreakerBank-0.1.5-SNAPSHOT.jar: snakeyaml-1.28.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28	CVE-2022-38751	vulnerable	6.5 MODERATE	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28
BluesBreakerBank-0.1.5-SNAPSHOT.jar: snakeyaml-1.28.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28	CVE-2022-38752	vulnerable	6.5 MODERATE	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28
BluesBreakerBank-0.1.5-SNAPSHOT.jar: snakeyaml-1.28.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28	CVE-2022-41854	vulnerable	6.5 MODERATE	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28
BluesBreakerBank-0.1.5-SNAPSHOT.jar: snakeyaml-1.28.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28	CVE-2022-38750	vulnerable	5.5 MODERATE	cpe:2.3:a:snakeyaml_project:snakeyaml:1.28
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-core-5.3.18.jar	cpe:2.3:a:springframework:spring-core:jar:5.3.18	CVE-2023-20860	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-core-5.3.18.jar	cpe:2.3:a:springframework:spring-core:jar:5.3.18	CVE-2022-22971	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-core-5.3.18.jar	cpe:2.3:a:springframework:spring-core:jar:5.3.18	CVE-2023-20861	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-core-5.3.18.jar	cpe:2.3:a:springframework:spring-core:jar:5.3.18	CVE-2023-20863	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-core-5.3.18.jar	cpe:2.3:a:springframework:spring-core:jar:5.3.18	CVE-2022-22968	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-core-5.3.18.jar	cpe:2.3:a:springframework:spring-core:jar:5.3.18	CVE-2022-22970	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-security-config-5.5.5.jar	cpe:2.3:a:springframework.security:spring-security-config:jar:5.5.5	CVE-2022-22978	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-security-config-5.5.5.jar	cpe:2.3:a:springframework.security:spring-security-config:jar:5.5.5	CVE-2023-20862 (OSSINDEX) DISPUTED	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-security-config-5.5.5.jar	cpe:2.3:a:springframework.security:spring-security-config:jar:5.5.5	CVE-2022-22976	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-security-core-5.5.5.jar	cpe:2.3:a:springframework.security:spring-security-core:jar:5.5.5	CVE-2022-22978	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-security-core-5.5.5.jar	cpe:2.3:a:springframework.security:spring-security-core:jar:5.5.5	CVE-2022-22976	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-security-crypto-5.5.5.jar	cpe:2.3:a:springframework.security:spring-security-crypto:jar:5.5.5	CVE-2022-22978	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-security-crypto-5.5.5.jar	cpe:2.3:a:springframework.security:spring-security-crypto:jar:5.5.5	CVE-2020-5408 (OSSINDEX) DISPUTED	false positive		pivotal or vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-security-crypto-5.5.5.jar	cpe:2.3:a:springframework.security:spring-security-crypto:jar:5.5.5	CVE-2022-22976	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-security-web-5.5.5.jar	cpe:2.3:a:springframework.security:spring-security-web:jar:5.5.5	CVE-2022-22978	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-security-web-5.5.5.jar	cpe:2.3:a:springframework.security:spring-security-web:jar:5.5.5	CVE-2022-31692 (OSSINDEX) DISPUTED	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-security-web-5.5.5.jar	cpe:2.3:a:springframework.security:spring-security-web:jar:5.5.5	CVE-2023-20862 (OSSINDEX) DISPUTED	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-security-web-5.5.5.jar	cpe:2.3:a:springframework.security:spring-security-web:jar:5.5.5	CVE-2022-22976	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-web-5.3.18.jar	cpe:2.3:a:springframework:spring-web:jar:5.3.18	CVE-2016-1000027	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-web-5.3.18.jar	cpe:2.3:a:springframework:spring-web:jar:5.3.18	CVE-2023-20860	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-web-5.3.18.jar	cpe:2.3:a:springframework:spring-web:jar:5.3.18	CVE-2022-22971	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-web-5.3.18.jar	cpe:2.3:a:springframework:spring-web:jar:5.3.18	CVE-2023-20861	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-web-5.3.18.jar	cpe:2.3:a:springframework:spring-web:jar:5.3.18	CVE-2023-20863	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-web-5.3.18.jar	cpe:2.3:a:springframework:spring-web:jar:5.3.18	CVE-2022-22968	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: spring-web-5.3.18.jar	cpe:2.3:a:springframework:spring-web:jar:5.3.18	CVE-2022-22970	false positive		vmware
BluesBreakerBank-0.1.5-SNAPSHOT.jar: tomcat-embed-core-9.0.60.jar	cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.60	CVE-2022-29885	vulnerable	7.5 HIGH	cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.60
BluesBreakerBank-0.1.5-SNAPSHOT.jar: tomcat-embed-core-9.0.60.jar	cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.60	CVE-2022-42252	vulnerable	7.5 HIGH	cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.60
BluesBreakerBank-0.1.5-SNAPSHOT.jar: tomcat-embed-core-9.0.60.jar	cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.60	CVE-2022-45143	vulnerable	7.5 HIGH	cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.60
BluesBreakerBank-0.1.5-SNAPSHOT.jar: tomcat-embed-core-9.0.60.jar	cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.60	CVE-2022-34305	vulnerable	6.1 MODERATE	cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.60
BluesBreakerBank-0.1.5-SNAPSHOT.jar: tomcat-embed-core-9.0.60.jar	cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.60	CVE-2023-28708	vulnerable	4.3 MODERATE	cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.60
BluesBreakerBank-0.1.5-SNAPSHOT.jar: tomcat-embed-core-9.0.60.jar	cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.60	CVE-2021-43980	vulnerable	3.7 LOW	cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.60

Figure 14: Dependency check vulnerability analysis