**Team members:**

Jovan Vunić
Dejan Radulović
Dušan Petković

# SECURITY ANALYSIS AND THREAT MODEL FOR CYBERSEC AND XWS PROJECTS

**Faculty of Technical Sciences, June 2020.**

# Short introduction to projects

**CYBERSEC** project represents PKI tool implemented with strong reliance on gathered information about X.509 certificates and digital certificates in general. In order to make tool properly functional, we have combined information gathered from independent research with knowledge acquired from lectures on *E-Business Systems Security* course.
We have also done research on OCSP protocol and implemented API for purpose of its usage.

**XWS** project represents microservice-architecture based system, including one standalone Spring Boot application, in order to demonstrate different types of communication on relations application-to-service and service-to-service. This includes SOAP protocol, while we also occasionally use feign client and message queue publish-subscribe patterns.
Most of implemented security aspects (everything but PKI tool) is binded with XWS project.

# Description of every implemented security aspect and its security controls in XWS project

Generally speaking, we could divide security aspects specified on afore-mentioned *E-Business Systems Security* course to following wholes:

- Data validation and attack prevention
- Authentication, authorization and access control
- Logging and monitoring.

## Data validation and attack prevention

In order for system to work accordingly to its use, data in use should be protected, properly validated and sanitized before having any touch with database or other sorts of data storages. Malicious attacks can be divided to:

- Injection attacks *(SQL Injection, short SQLI)*
- Cross-site scripting *(XSS attacks).*

To make it short – SQLI is most common attack in terms of standard web applications, which our project is. It consists of malicious code injection through client or some of widely used API tools (e.g. Postman, Insomnia) and sending it to interpreter.

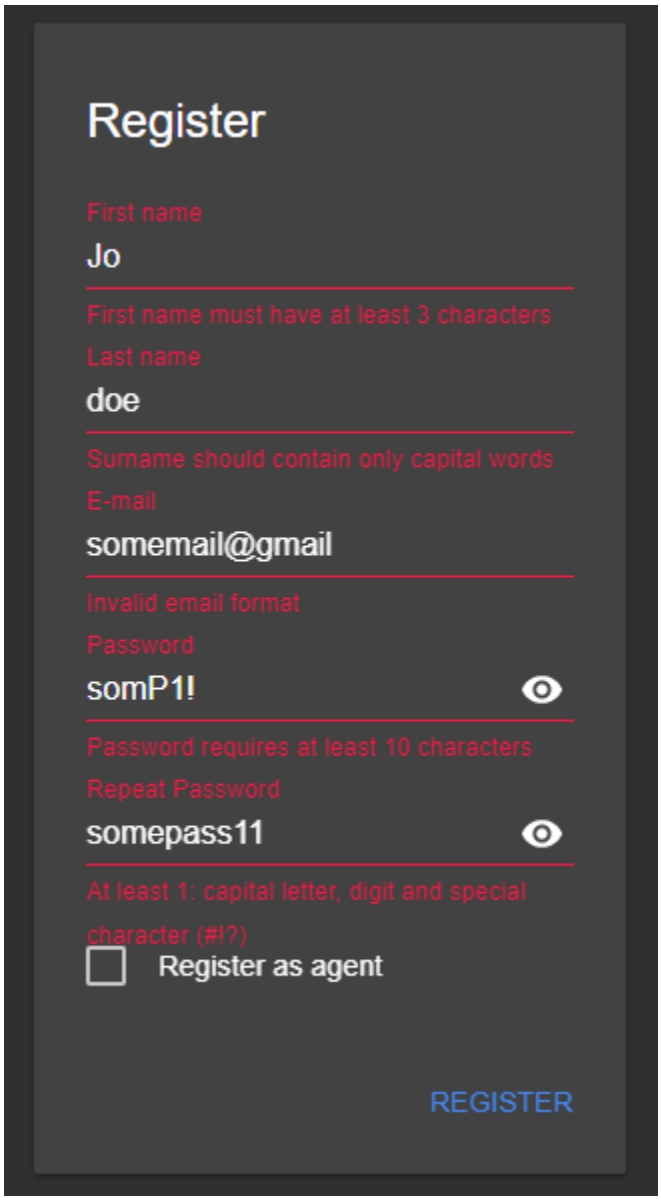
In order to prevent such sort of attacks, we have used few tools in combination with standard regular expression (further regex) validations.
On client side, every input is strongly validated with help of Vuelidate module – validation library made especially for Vue.js framework, which is mainly used for client-side code on XWS project. Most of input fields, specially text inputs, are validated with multiple criteria and if field fails at only one of them, action cannot be completed nor continued in any way possible.
On server side, data of every received request (if it has any) firstly makes its way through rigorous validation, only to be sanitized by OWASP Java Encoder module and then forwarded to further business logic. If at any moment validation fails, appropriate exception handler is activated, so it sends feedback about failure to client.

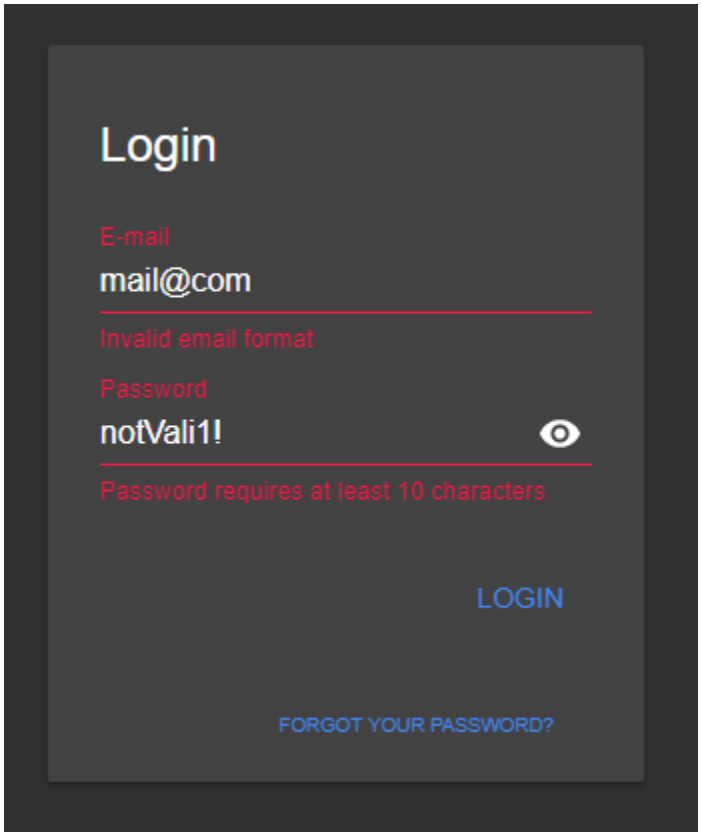
```java
/*
 * Checks if there is mismatch against given regex in any of String attributes of User, which are received from
 * client through DTO. It also checks if attributes are existing.
 * Returns TRUE if given DTO is valid, else returns FALSE.
 */
private Boolean validateUser(UserDTO userDTO, Pattern patternNames, Pattern patternPass) {
    if (userDTO.getFirstName() == null || userDTO.getLastName() == null || userDTO.getEmail() == null ||
            userDTO.getPassword() == null || userDTO.getRepeatPassword() == null ||
            userDTO.getFirstName().trim().equals("") || userDTO.getLastName().trim().equals("") ||
            userDTO.getEmail().trim().equals("") || userDTO.getPassword().trim().equals("") ||
            userDTO.getRepeatPassword().trim().equals("") || userDTO.getFirstName().length() < 3 ||
            userDTO.getLastName().length() < 3 || userDTO.getPassword().length() < 10 ||
            userDTO.getRepeatPassword().length() < 10 || !userDTO.getRepeatPassword().equals(userDTO.getPassword()) ||
            (userDTO.getEmail().trim().split( regex: "@").length <= 1) ||
            !patternNames.matcher(userDTO.getFirstName().trim()).matches() ||
            !patternNames.matcher(userDTO.getLastName().trim()).matches() ||
            !patternPass.matcher(userDTO.getPassword().trim()).matches() ||
            !patternPass.matcher(userDTO.getRepeatPassword().trim()).matches()) {
        return false;
    }

    return true;
}
```

**Image 1:** data validation on server side

```java
/*
 * Helper method for forbidden character sanitization in terms of user data.
 */
private void sanitizeUserData(UserDTO userDTO) {
    userDTO.setFirstName(Encode.forHtml(userDTO.getFirstName()));
    userDTO.setLastName(Encode.forHtml(userDTO.getLastName()));
    userDTO.setEmail(Encode.forHtml(userDTO.getEmail()));
    userDTO.setPassword(Encode.forHtml(userDTO.getPassword()));
    userDTO.setRepeatPassword(Encode.forHtml(userDTO.getRepeatPassword()));
}
```

**Image 2:** data sanitization on server side




**Images 3 and 4:** data validation on client side

It is worth mentioning that Hibernate (ORM for Spring/Spring Boot framework) takes care about injection attacks as long as we do not type any native queries in our methods, including use of @Query annotation, where developer should enter native query as parameter.
At some point, complex database queries (here executed through use of JPA Repository) can lead to vulnerabilities, but we did not have need for such thing on this project. Sorting is not yet completed, but whole algorithm will operate only on client side with returned values form server.

Cross-site scripting, commonly known as XSS attack, is considered to be a special class of injection attacks. Main idea of these attacks is to make web browser of victim execute malicious Javascript code, which can result in devastating consequences.

Luckily enough, there are many tools and libraries that can help prevent this from happening. Vue.js framework does character escaping by default in component templates, which in great measure prevents script injection and wraps HTML code with additional layer of security.
This means that also all dynamical binds (which is characteristic for Vue.js) are escaped by default, therefore even if any of those binds is malicious, it will not be supported and executed.
With help of Vuelidate, persistent and reflected XSS attacks are also subdued, because of regex combining whitelist and blacklist validation patterns. Some keywords, such as *script* or *from*, are excluded and cannot take place in input. Even if script somehow passes by, with use of API tools, it will be subdued on server side, after which it will be sanitized if it still makes through – all that before connecting to database in any way.
DOM-based XSS attacks are nowadays escaped by default in every modern browser, because of their common feature – escaping of Javascript code in address bar.
All that being said, there were some constraints we needed to abide in order to take risk of XSS attacks to minimum. We used Node package manager (npm) for installation of third-party packages and open-source solutions, so there was no fear of deprecated package versions, npm handles those situations automatically – while CDN does not, therefore danger can come in out of nowhere. Also, avoiding v-html directive can save your application in some critical situations. We did not have need for use of it, on this project or any other in past.

**References:** X-XSS Protection, Should you care about XSS in Vue.js?, The definitive guide to XSS, Cross-Site Scripting Prevention Cheat Sheet, OWASP Java Encoder Project

# Authentication, authorization and access control

In systems such is *RentaSoul* (working name of XWS project), authenticated and authorized access to resources is of essential value for platform to work in a proper way.
At this point RBAC protocol (through Spring Security) and ACL protocol (through Encrypting File System - EFS) made things a lot easier to control.

Some cases include only file locking with help of EFS, embedded in Windows 10, and others include whole folder (along with its content) locking. When file or folder is locked with EFS, only user that locked given resource can again access it. This is due to use of encryption certificate, which is associated with user account, not with machine itself.
For example: user A locks file F and then signs out; user B signs in and tries to access file F, but gets rejected because of authorization issues – file is not locked with his encryption certificate. This kind of approach makes files and folders physically safe and protected, but not without human discipline: if user A leaves his machine without logging out, user B can use it to access file F. That is just one situation. Most common security breaches come as results of human negligence: people tend to leave their credentials on papir sticked to their working desks, which almost every year somewhere has huge violations of data as consequence.
EFS is not magic, but is a nice tool to consider if you want to protect your data locally and if you are well self-disciplined. This way we have implemented ACL protocol for XWS project.

As for RBAC protocol, there are three main roles in our system: ADMIN, AGENT and SIMPLE_USER. Each of these roles has multiple privileges, while in some rare cases there comes to overlapping, it is mostly not the case. Even when it is, business logic is almost entirely different according to logged user's role. In some cases, user with role of SIMPLE_USER can be blocked or banned. While the first one does not include privilege striping, the second one will. It is yet to be implemented, but it will surely be based on striping the privileges off of user. Endpoints in controllers, where service methods are being invoked, are controlled by @PreAuthorize annotations included in Spring Boot framework. When client sends request, it goes firstly to API gateway, which then does filtration and forwards user request with all headers and information needed for proper authorization. Request comes to controller endpoint, where it gets checked not by role inclusion, but by privilege inclusion. For example, both AGENT and SIMPLE_USER can post advertisements to main platform, so this type of check has much more logic in it when considering cases like this. Processes of registration and password reseting are complex and include use of mail, which is made as individual microservice, listening to its message queue publisher – account microservice. Both include one-time-use tokens.
While user is logged in, his token slowly expires and gets refreshed after some time, after proper checks. At last, it is worth to mention that every sight of communication is supported by HTTPS.

Non-trusted sources are restricted and without access to application. Certificates are generated by OpenSSL solution.

**References:** Back-end Spring Boot Security, How to use EFS encryption, Spring Security – Roles and Privileges

# Logging and monitoring

Logging events can be helpful from more than one point of view: debugging, monitoring, preventing unwanted behavior, etc.
Logged events include errors and exception throwing, database transactions, successful operations, endpoint accessing, system responses to client requests, access control information.

Logs include subject (who has executed/triggered action), information about executed/triggered action, execution/trigger moment – this way system guarantees undeniability, i.e. subject cannot deny his connection to executed/triggered action. This can be very important in cases where something malevolent was prevented or has happened to system. It can be unequivocal proof that subject has connection to given action.

Each log in XWS project is *complete* – appeals to demands of undeniability and monitoring, *reliable* – appeals to demands of memory configuration and file rotation, *useable* - appeals to demands of effective extraction of events significant to undeniability, *concise* – appeals to demands of optimization and memory saving.

**References:** Logging in Spring Boot, Overview of logging best practices

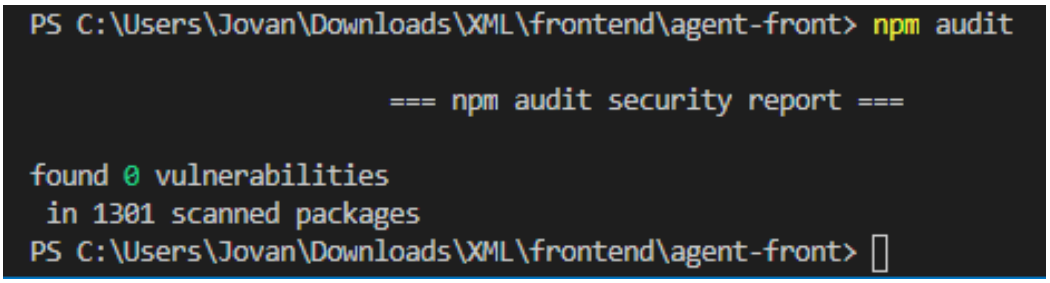
# Security analysis of third-party dependencies and packages

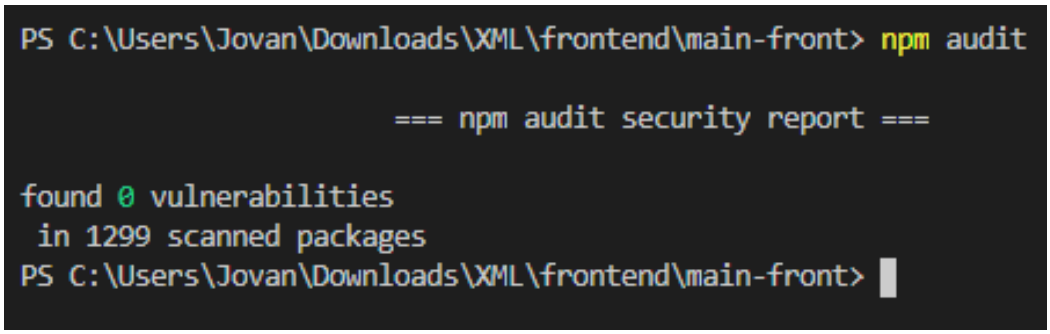
For purpose of testing our front-end third-party packages for vulnerabilities, we have used same package manager we used for installing them – npm. For some time now, npm has pretty handsome command – *npm audit*, which is followed by its variation – *npm audit fix*.
First one shows all vulnerabilities in your project – from low to critical risk level, but the second one has crucial role in successful development – it fixes security issues and vulnerabilities while also giving feedback to user. For example, some package may be update or it may be rolled back, whatever in that moment cleans it from vulnerabilities.
When running front-end with *npm run serve command*, *npm audit command* triggers automatically – with every new run. Basically, it constantly searches for vulnerabilities in libraries and modules installed in project structure.

Another automatic service for potential vulnerability resolving is Dependabot. It is configured to work with Github repositories of XWS and CYBERSEC projects, in terms of maintenance and code security. In few occasions, Dependabot requested merge for its pull requests, most notably for jQuery and Websocket Extensions vulnerability issues. All we had to do was to merge given pull requests and everything worked properly, but now even safer than before.

On server side we used OWASP Dependency-Check project to test dependencies in each of our microservice modules. At time, there were some issues with HIGH and CRITICAL risk level, mainly grouped around deprecated PostgreSQL driver and unsafe parts of Spring Security starter dependency.

**References:** Auditing package dependencies for security vulnerabilities, National Vulnerability Database, OWASP Dependency-Check: How Does It Work?, Ready to use Java dependencies vulnerability checker, Dependency-Check Maven: Usage, Dependency Vulnerabilities Check



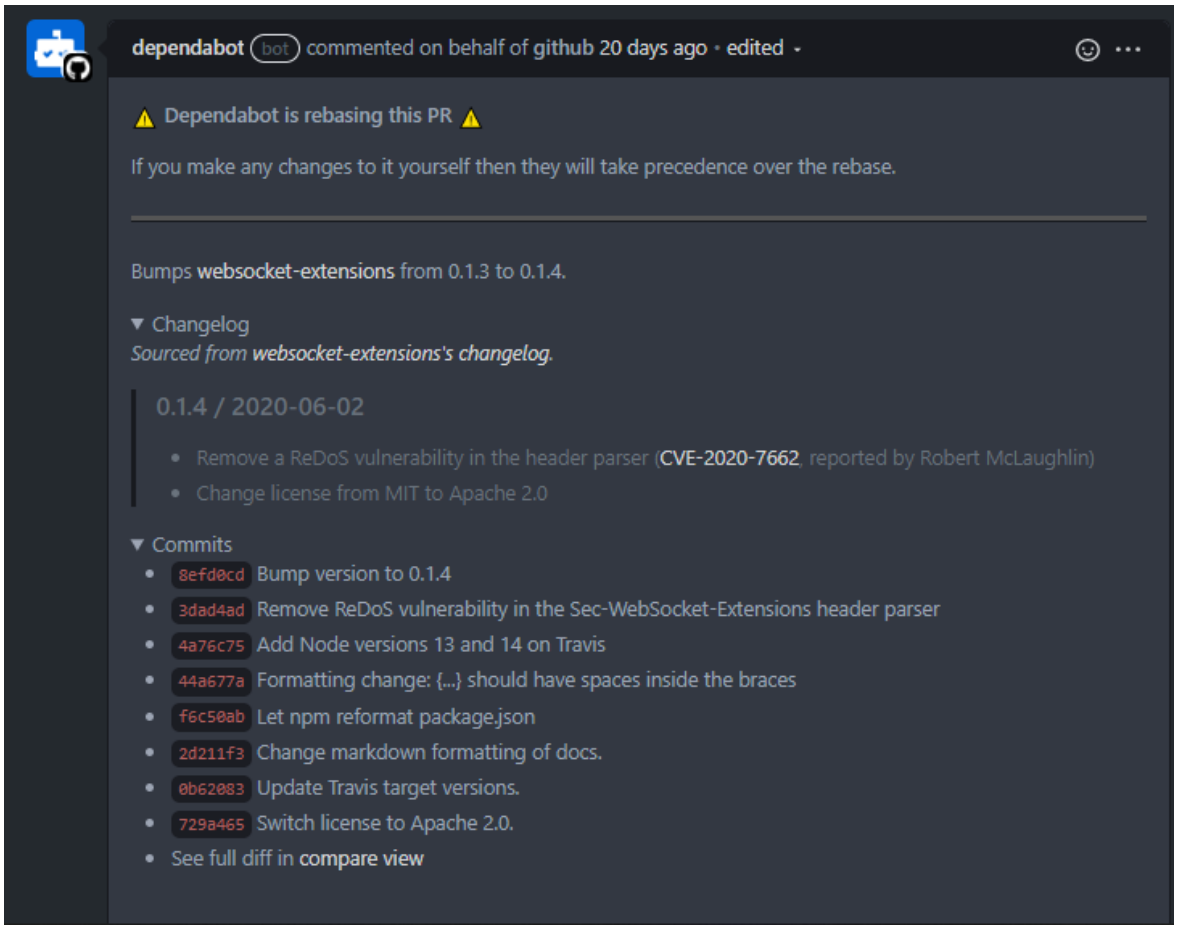

**Images 5 and 6:** successful npm audit vulnerability check for agent application front (top) and main front (bottom)

```
PS C:\Users\Jovan\Downloads\CYBERSEC\cybersec_client> npm audit

                    === npm audit security report ===

# Run  npm update http-proxy --depth 4  to resolve 1 vulnerability

  High            Denial of Service

  Package         http-proxy

  Dependency of   @vue/cli-service [dev]

  Path            @vue/cli-service > webpack-dev-server >
                  http-proxy-middleware > http-proxy

  More info       https://npmjs.com/advisories/1486


# Run  npm update webpack-dev-server --depth 2  to resolve 1 vulnerability

  Low             Prototype Pollution

  Package         yargs-parser

  Dependency of   @vue/cli-service [dev]

  Path            @vue/cli-service > webpack-dev-server > yargs > yargs-parser

  More info       https://npmjs.com/advisories/1500


found 2 vulnerabilities (1 low, 1 high) in 1342 scanned packages
  run `npm audit fix` to fix 2 of them.
PS C:\Users\Jovan\Downloads\CYBERSEC\cybersec_client>
```

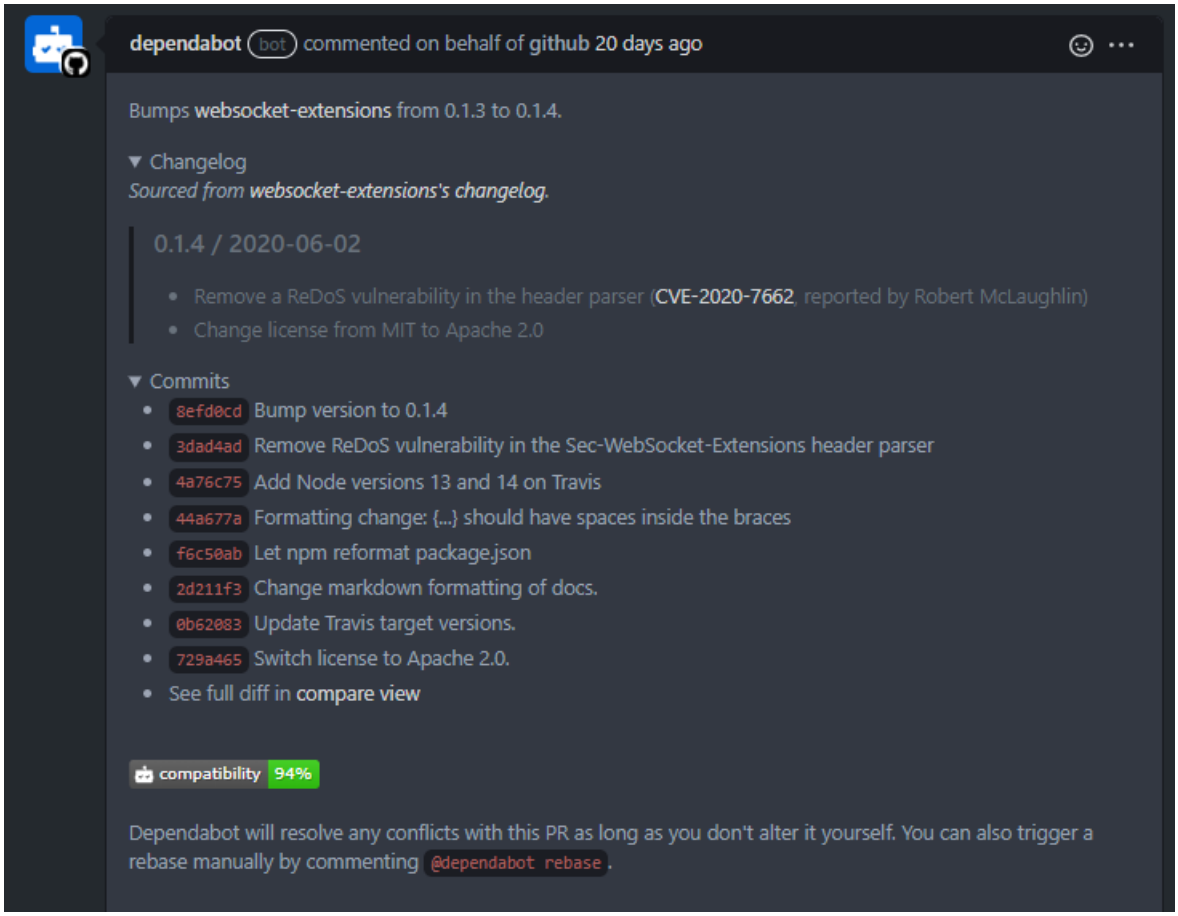
**Image 7:** unsuccessful npm audit vulnerability check for cybersec front

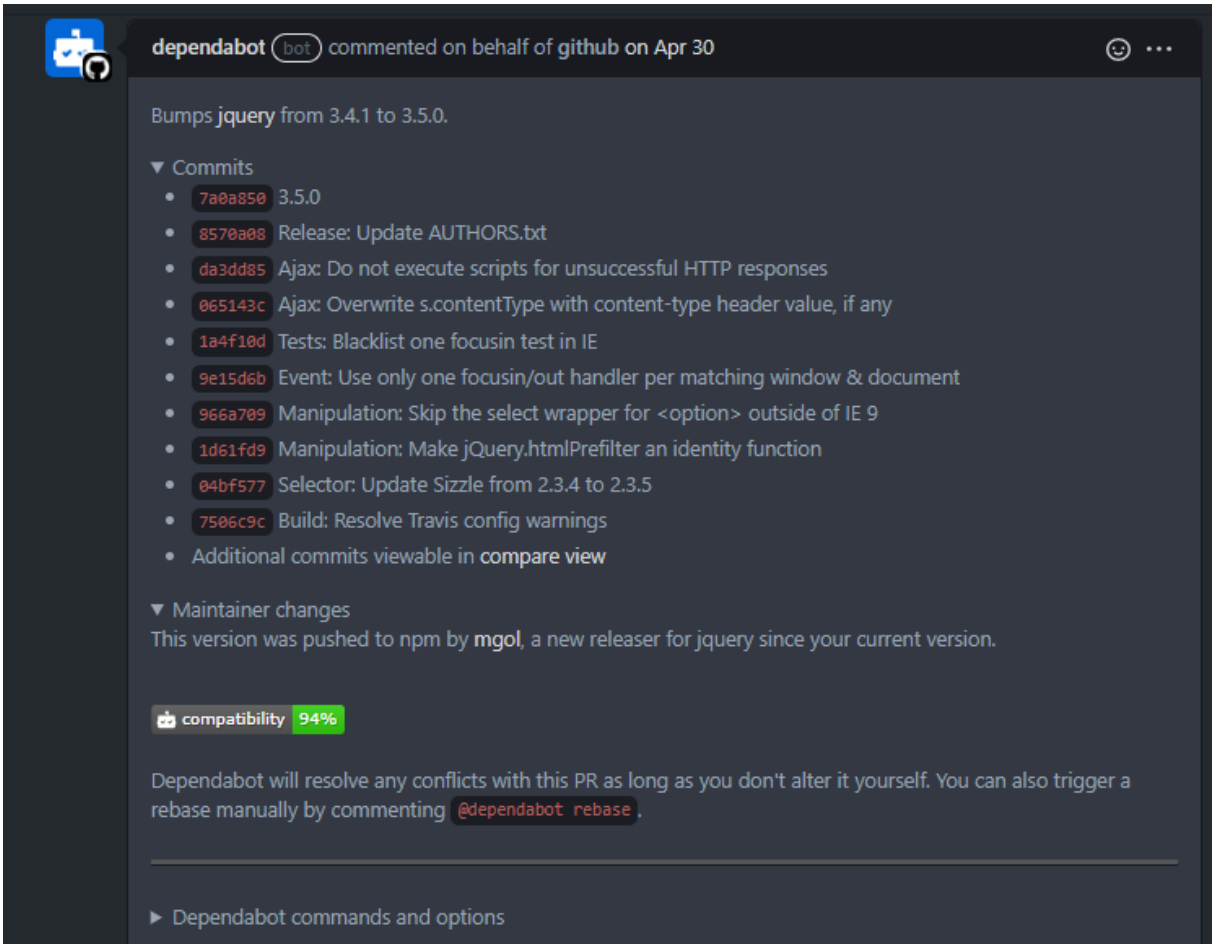
```
PS C:\Users\Jovan\Downloads\CYBERSEC\cybersec_client> npm audit fix
npm WARN rollback Rolling back are-we-there-yet@1.1.5 failed (this is probably harmless): EPERM: operation not permi
tted, lstat 'C:\Users\Jovan\Downloads\CYBERSEC\cybersec_client\node_modules\fsevents\node_modules'
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.12 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.12: wanted {"os":"darwin","arch"
:"any"} (current: {"os":"win32","arch":"x64"})

added 7 packages from 3 contributors, removed 15 packages, updated 17 packages and moved 1 package in 31.844s

42 packages are looking for funding
  run `npm fund` for details

fixed 2 of 2 vulnerabilities in 1342 scanned packages
PS C:\Users\Jovan\Downloads\CYBERSEC\cybersec_client>
```

**Image 8:** successful npm audit vulnerability fix for cybersec front

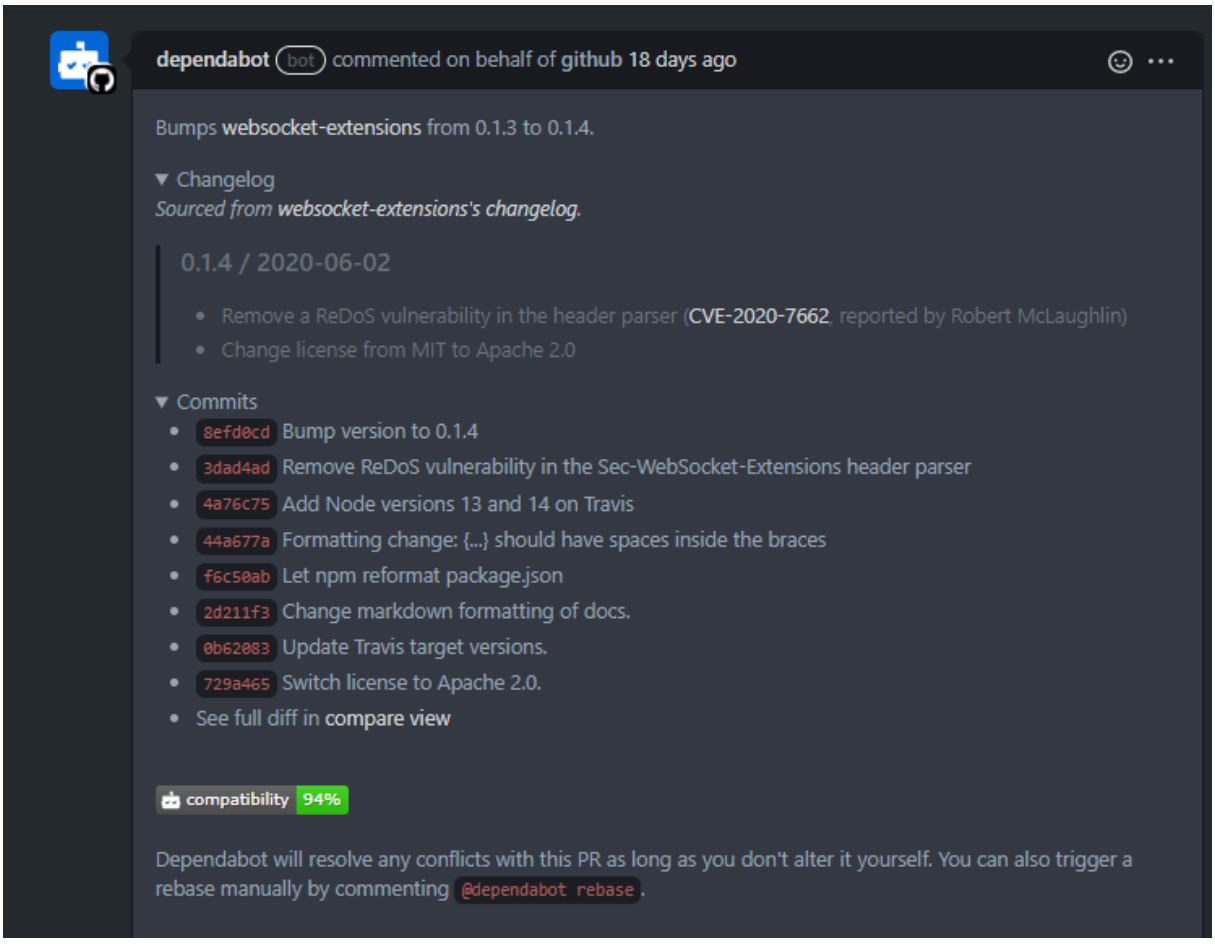



**Images 9 and 10:** Dependabot fixes for Websocket Extensions on main and agent front






**Images 11 and 12:** Dependabot fixes for jQuery and Websocket Extensions on cybersec front



# Issue solving

Some of common vulnerabilities for XWS project were resolved simply by modifying pom.xml file under the structure of given microservice module which had that sort of vulnerability.
In that way we have solved the following vulnerabilities, according to OWASP:

- CVE-2018-10237 by moving to 25.1-jre version (stable)
  - **Description:** Unbounded memory allocation in Google Guava 11.0 through 24.x before 24.1.1 allows remote attackers to conduct denial of service attacks against servers that depend on this library and deserialize attacker-provided data, because the AtomicDoubleArray class (when serialized with Java serialization) and the CompoundOrdering class (when serialized with GWT serialization) perform eager allocation without appropriate checks on what a client has sent and whether the data size is reasonable.
  - **Severity:** MEDIUM, with Base Score of 5.9
  - **Use:** Guava is used in XWS project for Orchestrated Saga pattern through Axon server
- CVE-2020-13692 by moving to 42.2.13 version (stable)
  - **Description:** PostgreSQL JDBC Driver (aka PgJDBC) before 42.2.13 allows XXE.
  - **Severity:** CRITICAL, with Base Score of 9.8
  - **Use:** PostgreSQL is used as data storage solution for every database-demanding module of XWS project
- CVE-2017-13098 by moving to 1.60 version (stable)
  - **Description:** BouncyCastle TLS prior to version 1.0.3, when configured to use the JCE (Java Cryptography Extension) for cryptographic functions, provides a weak Bleichenbacher oracle when any TLS cipher suite using RSA key exchange is negotiated. An attacker can recover the private key from a vulnerable application. This vulnerability is referred to as "ROBOT."
  - **Severity:** HIGH, with Base Score of 7.5
  - **Use:** BouncyCastle project is used in CYBERSEC project as helper with implementation of PKI tool with its predefined functionalities
- CVE-2018-1000180 by moving to 1.60 version (stable)
  - **Description:** Bouncy Castle BC 1.54 - 1.59, BC-FJA 1.0.0, BC-FJA 1.0.1 and earlier have a flaw in the Low-level interface to RSA key pair generator, specifically RSA Key Pairs generated in low-level API with added certainty may have less M-R tests than expected. This appears to be fixed in versions BC 1.60 beta 4 and later, BC-FJA 1.0.2 and later.
  - **Severity:** HIGH, with Base Score of 7.5

    - **Use:** BouncyCastle project is used in CYBERSEC project as helper with implementation of PKI tool with its predefined functionalities
- CVE-2018-1000613 by moving to 1.60 version (stable)
    - **Description:** Legion of the Bouncy Castle Legion of the Bouncy Castle Java Cryptography APIs 1.58 up to but not including 1.60 contains a CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') vulnerability in XMSS/XMSS^MT private key deserialization that can result in Deserializing an XMSS/XMSS^MT private key can result in the execution of unexpected code. This attack appear to be exploitable via A handcrafted private key can include references to unexpected classes which will be picked up from the class path for the executing application. This vulnerability appears to have been fixed in 1.60 and later.
    - **Severity:** CRITICAL, with Base Score of 9.8
    - **Use:** BouncyCastle project is used in CYBERSEC project as helper with implementation of PKI tool with its predefined functionalities

Some vulnerabilities reported by OWASP Dependency Check were considered, thought through, analyzed in terms of possible severity to system and kept without any sort of resolving.
We have made short explanation followed by arguments for every reported possible vulnerability that was kept intact:

- CVE-2014-9494
    - **Description:** RabbitMQ before 3.4.0 allows remote attackers to bypass the loopback_users restriction via a crafted X-Forwarded-For header.
    - **Severity:** MEDIUM, with Base Score of 5.0
    - **Exploitation possibility:** 0%
    - **Possible consequences:** Compromise of default user data and data integrity breach. Possible data exploitation and various types of attacks, for example man-in-the-middle-attack.
    - **Explanation:** In XWS project we use RabbitMQ for few things – data pumping and email communication included – based on Publish-Subscribe pattern. As our application is stateless, not just that we do not have a need for X-Forwarded-For header, but it is also disabled by configuration settings in every of our microservice modules. Therefore, opinion is that this vulnerability does represent danger, but only in non-protected and stateful applications.
- CVE-2018-11087
    - **Description:** Pivotal Spring AMQP, 1.x versions prior to 1.7.10 and 2.x versions prior to 2.0.6, expose a man-in-the-middle vulnerability due to lack of hostname

validation. A malicious user that has the ability to intercept traffic would be able to view data in transit.
    - **Severity:** MEDIUM, with Base Score of 5.9
    - **Exploitation possibility:** 50%
    - **Possible consequences:** Attacker can get access to the sent message, but it will be encrypted. In case of successful message decryption, man-in-the-middle-man attack can be executed.
    - **Explanation:** In XWS project, RabbitMQ has its own certificate for message signing, so man-in-the-middle attack should not be so easily executed.
- CVE-2018-1279
    - **Description:** Pivotal RabbitMQ for PCF, all versions, uses a deterministically generated cookie that is shared between all machines when configured in a multi-tenant cluster. A remote attacker who can gain information about the network topology can guess this cookie and, if they have access to the right ports on any server in the MQ cluster can use this cookie to gain full control over the entire cluster.
    - **Severity:** MEDIUM, with Base Score of 6.5
    - **Exploitation possibility:** 0%
    - **Possible consequences:** Attacker can gain full control over the entire cluster and possibly compromise whole data flow of cluster.
    - **Explanation:** This requires higher level of security than one demanded from us, because XWS represents just a simple student project with basic security controls implemented and demonstrated, therefore opinion is that this security feature is irrelevant at this course. In addition, we do not configure multi-tenant cluster and do not use more than one machine at current point. Not a single module is deployed, instead they are locally available for demonstration of implementation.
- CVE-2019-11281
    - **Description:** Pivotal RabbitMQ, versions prior to v3.7.18, and RabbitMQ for PCF, versions 1.15.x prior to 1.15.13, versions 1.16.x prior to 1.16.6, and versions 1.17.x prior to 1.17.3, contain two components, the virtual host limits page, and the federation management UI, which do not properly sanitize user input. A remote authenticated malicious user with administrative access could craft a cross site scripting attack that would gain access to virtual hosts and policy management information.
    - **Severity:** MEDIUM, with Base Score of 4.8
    - **Exploitation possibility:** 0%
    - **Possible consequences:** Unknown
    - **Explanation:** As it is stated in above mentioned vulnerability-keeping explanation, we consider this also an advanced security feature, for which our RentaSoul solution does not have need or use.

- CVE-2018-1258
  - **Description:** Spring Framework version 5.0.5 when used in combination with any versions of Spring Security contains an authorization bypass when using method security. An unauthorized malicious user can gain unauthorized access to methods that should be restricted.
  - **Severity:** HIGH, with Base Score of 8.8
  - **Exploitation possibility:** 0%
  - **Possible consequences:** Malicious user, which does not have privilege/privileges, can gain access to previously authorized methods of system.
  - **Explanation:** Although this vulnerability does represent very dangerous hole in system which uses Spring Security's method security, XWS project does not use method security nor its annotation @Secure. Authorization was described before and it has strong connection with Spring Security, but none of it is based on method security. Therefore, danger could not be present if method security is not explicitly invoked inside of microservice structure.

- CVE-2015-9251
  - **Description:** jQuery before 3.0.0 is vulnerable to Cross-site Scripting (XSS) attacks when a cross-domain Ajax request is performed without the dataType option, causing text/javascript responses to be executed.
  - **Severity:** MEDIUM, with Base Score of 6.1
  - **Exploitation possibility:** 30%
  - **Possible consequences:** Possible forwarding of malicious scripts to server side, after which it will be sanitized and not executed.
  - **Explanation:** In RentaSoul solution jQuery is not used explicitly in code, some of third-party packages do have connection to it, but every security issue is tracked and fixed by Node Package Manager. This was explained earlier in the report.