NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4237 Software Security and Data Privacy

Exercise 1, 2022: Information Gathering

Group 65

Author(s):
Sander Strand Engen
Adria Torija Ruiz

Contents

1 Introduction

In this report we will analyse the endpoints, links, parameters and the authentication barrier of a website by the use of a page map. Moreover we will look into what services are used as the Web-server, Front-End framework, Back-End framework and Database and also brief description of how we found this information.

2 Page map

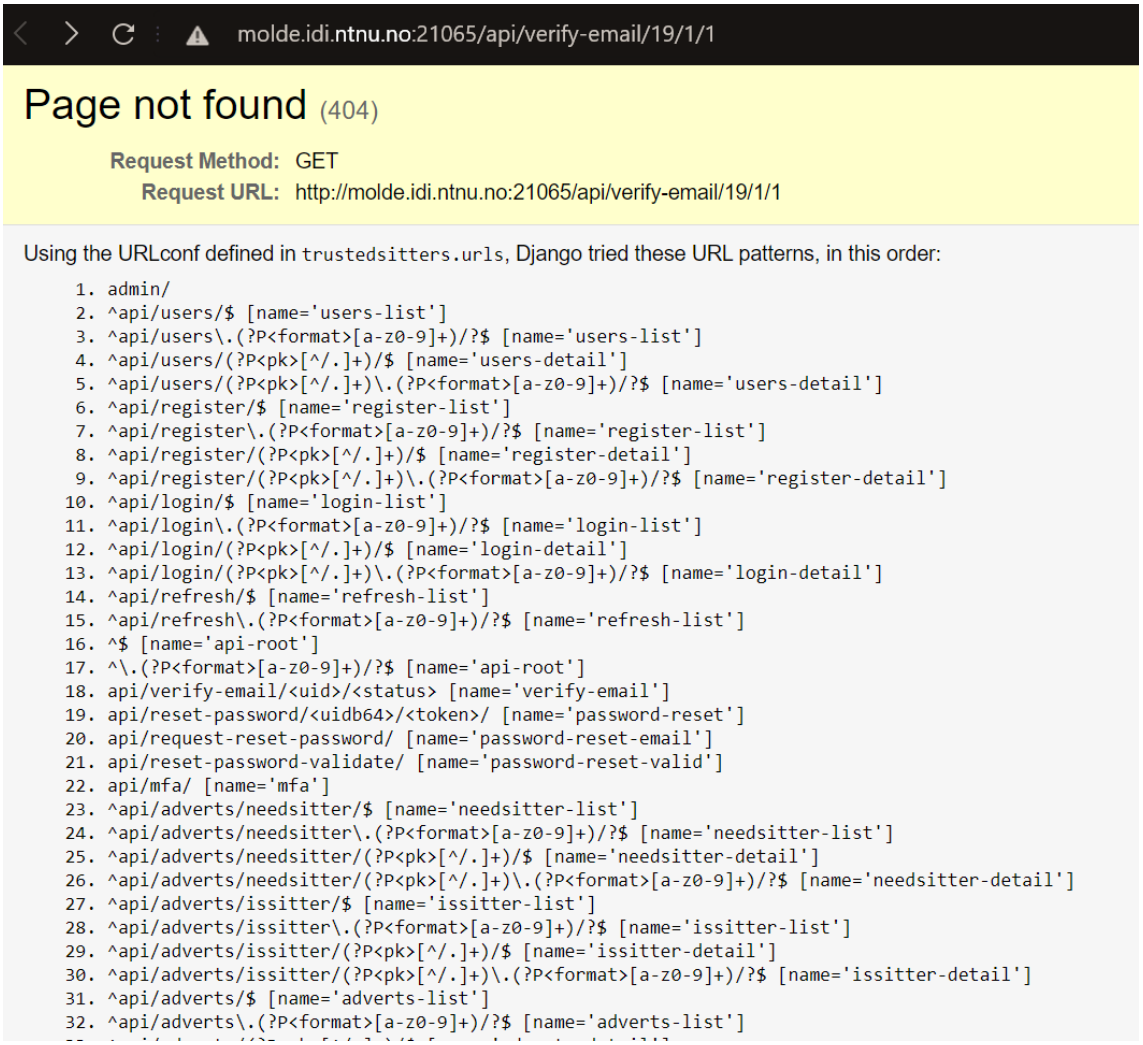

Figure 1: 404 Error generated when traversing the /api/verify-email page.

We mainly used the browser and Burp Suite [1] in order to find the pages. Mainly looking at the Firefox developer tools, looking at the HTML code and also the interactions and behaviour when navigating the site. Using path traversal with different IDs on the /api/verify-email URL, we were able to generate a 404 error providing us with the file "trustedsitters.urls" defined at "URLConf", shown in figure 1, giving a lot of information about valid paths on the website.

To discern what pages were accessible with and without authentication, we tried accessing the different pages we were able to find while both authenticated through the signin page and without, and observing the subsequent behaviour.

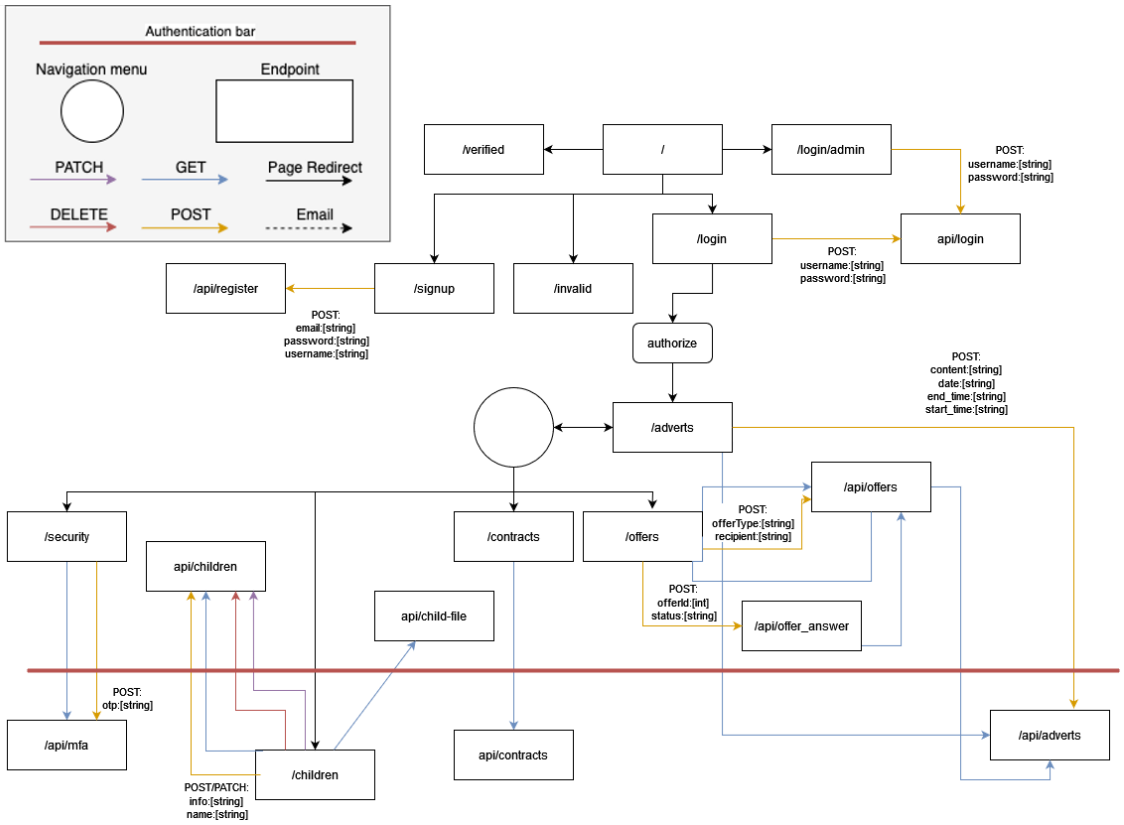

Figure 2: Pagemap

In figure 2 the resulting pagemap from our analysis is presented. Description of the different HTML methods are shown in the legend in the upper left. The pages above the red line are the pages we were able to access without being authenticated, and the pages below are the ones requiring authentication.

3 Services

To audit what underlying services were running the application, we used Burp Suite [1] to map the website and analyse responses to different HTTP requests. Burp Suite quickly identified a SQL injection vulnerability at the page directory /api/offers/ which was used to generate an error that supplied us with a lot of information about the services.

3.1 Webserver

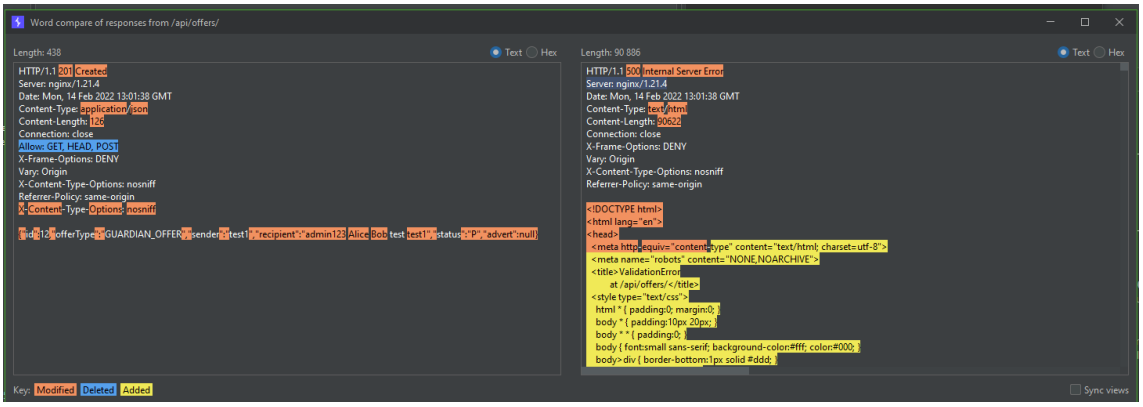

Figure 3: Server field indicating Nginx webserver.

The webserver was the easiest service to figure out. In most of the HTTP responses from the website, the *server* field was populated with the text *nginx/1.21.4*, signifying that the webserver used was Nginx.

3.2 Frontend

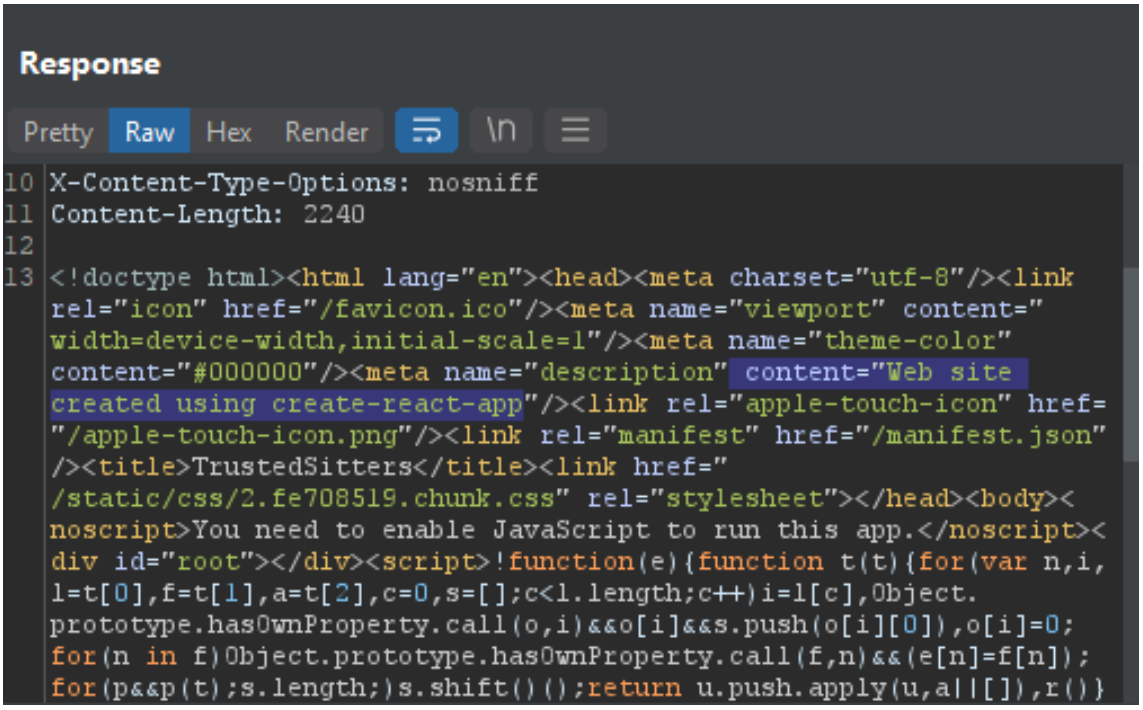

Figure 4: Doctype indicating React framework.

The front-end framework is indicated in the doctype of the response from the main page with the description tag containing the text *Web site was created using create-react-app*. It's safe to assume that the front-end framework is React, as nothing we can find indicates otherwise.

3.3 Backend

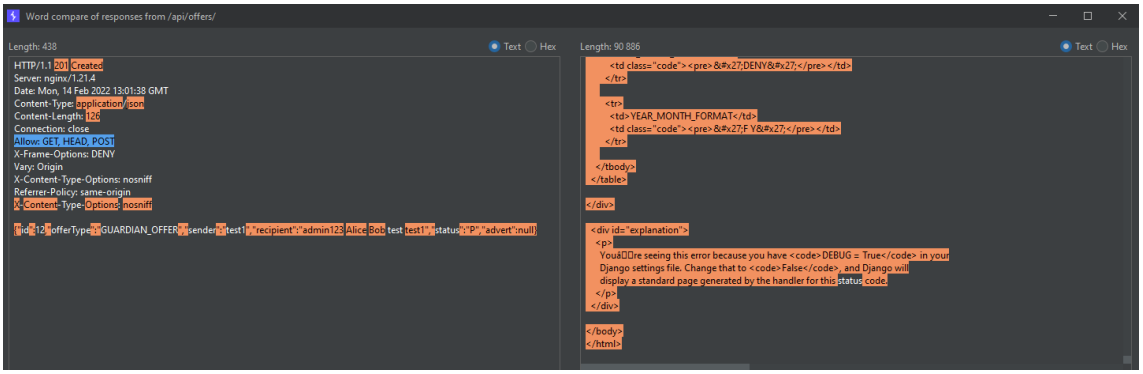

Figure 5: Error message indicating Django backend.

To find the backend framework was a bit more complicated. As described, a SQL injection vulnerability was found at /api/offers/. Using this vulnerability, we were able to generate a *500 server error* response, containing a lot of information about the backend system. Looking through the response, there were several references to the Django framework, making it safe to assume that this is the underlying backend framework.

3.4 Database

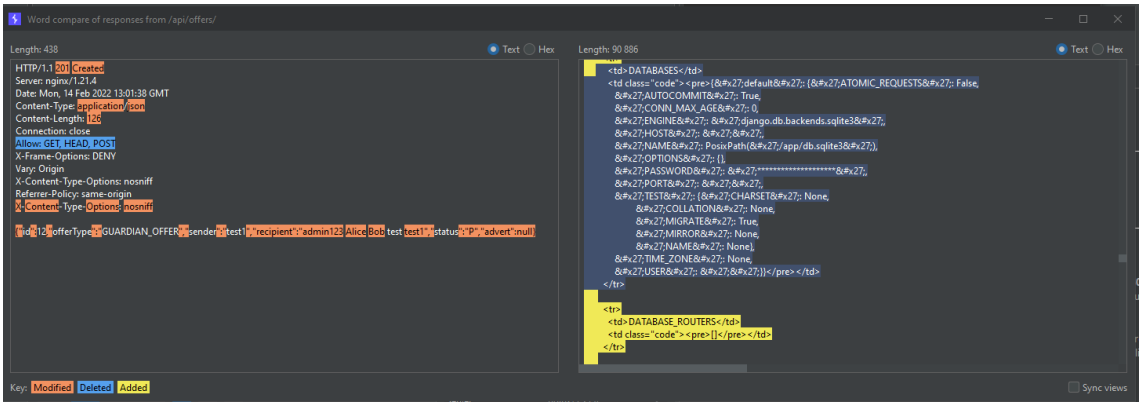

Figure 6: Error message indicating SQLite database.

With the same error message, information about the database was provided as well. In the *DATABASES* field, there is a referral to app/db.sqlite, indicating that the database service is SQLite.

4 Conclusion

By analysing the website several exploitable weaknesses and buggy behaviour were found across the different pages. The most significant one is the SQL injection vulnerability [2] exploited in section 3, but there were also vulnerabilities such as cross-site scripting [3] and XPATH injections [4] present.

The website's buggy behaviour allowed registration without proper verification, posting of offers without authentication and other unintended functionalities.

Concluding, a website as the one provided is a shining example of how not to configure a service, and serves as an example of common security pitfalls when configuring a website.

References

[1] PortSwigger. *PortSwigger - Burp Suite*. 2012. URL: https://portswigger.net/burp.

[2] OWASP. *SQL Injection*. URL: https://owasp.org/www-community/attacks/SQL_Injection.

[3] OWASP. *Cross-site Scripting*. URL: https://owasp.org/www-community/attacks/xss/.

[4] OWASP. *XPATH injection*. URL: https://owasp.org/www-community/attacks/XPATH_Injection.