DevOps - Group I - Security

Lasse Agersten, Niclas Valentiner, Mikkel Østergaard, Philip Johansen

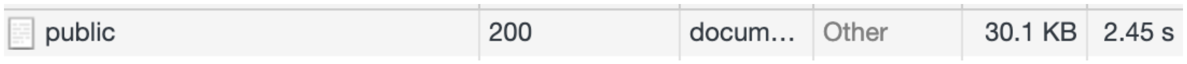
April 15th, 2020

1 Monitor SLA

According to the SLA as provided by our monitored group, they promise a responsiveness with a meantime of 11 ms and responding to 98% of all requests within 500ms.

As there has been complications in relation to setting up monitoring of our target group during the allotted time we have not had the chance to monitor the team and thereby conclude whether or not they have held up on these claims.

However, we have performed limited manual monitoring on their front-end. When accessing their public front-end on `http://46.101.215.40:5000/public` it becomes clear that response times always are over 500ms. As an example it took 2.45s from request to receipt of the page when accessing it on the 15. April 20:18. However, since we do not have active monitoring on their endpoints we are not able to prove that it is below 98% of all requests that actually takes more than 500ms to respond. We do however highly suspect this is the case, and we believe this is where they have introduced their performance regression.



public	200	docum…	Other	30.1 KB	2.45 s

Figure 1.1: Time spent for `http://46.101.215.40:5000/public` to respond when acessing via. Chrome

2 Pen Test Your System

2.1 Risk assets

After an analysis of our application, then we've determined that we have several assets that we need to be aware of, and secure properly. These are as follows:

- One of the main assets of our Minitwit application is the information of our users, primarily their passwords as all other information is publicly available on our site.
- The second main asset is the access to user profiles. Only the user who owns a profile should be able to access it.
- Another asset that is at risk is our password hashing configuration which is publicly available in our source code at the moment. Letting this information get into the hands of malicious people could enable them to crack user passwords and enable malicious access to user profiles.
- Our simulation API is also an asset that is worth protecting, since this lets callers do actions for any user. If this is not protected, then it would enable malicious users to perform actions of another user, i.e. manipulating with our primary assets.
- Our server is an asset as well that we need to be aware of, since malicious access to it would allow extraction of production specific configurations such as users and passwords for Kibana, Grafana and our database.
  Access to the server would furthermore enable hackers to delete entries in our logging, enabling them to hide their own traces and attacks.

2.2 Risk matrix

A risk matrix (Which can be seen in figure 2.1) will be made based on the following analysis of each of our assets:

- The risk of exposing user passwords is expected to be quite low since access to the server is only possible through ssh. However, since our site is running on http it would be easy to simply sniff internet traffic thereby getting specific users' raw passwords.
  Since letting a password get into the hands of the hacker compromises our two main assets, this is classified as **medium risk with high impact**.
- The risk of a hacker getting access to our password hashing configuration (including our static salt) is very likely since the configuration is publicly available on GitHub.
  In case a hacker somehow gets access to our database, then it would make the process of decoding the original passwords a lot easier. Therefore this will be classified as **high risk with medium impact**.
- If we don't protect our simulation API then it would allow malicious users to add messages as any user they want to be. In other words, this poses a risk to one of our primary assets.
  The probability of this happening isn't the highest since our API is hidden behind authentication. The problem with this authentication is that it uses a static string which is available in the source code on GitHub. Hence we classify this as **medium risk with medium impact**.

- Access to our server would give hackers full access to the production server, which would allow them to simply change passwords of profiles giving them access to arbitrary users, while blocking the user's access. This would allow the hackers to alter logs to their liking and more.
  However, the likelihood of a hacker getting full access to our server is fairly low, since it requires ssh keys. The only way would be for the hacker to target the developers of Minitwit directly and attempt to extract their keys.
  Therefore this risk is classified as **low risk with high impact**.

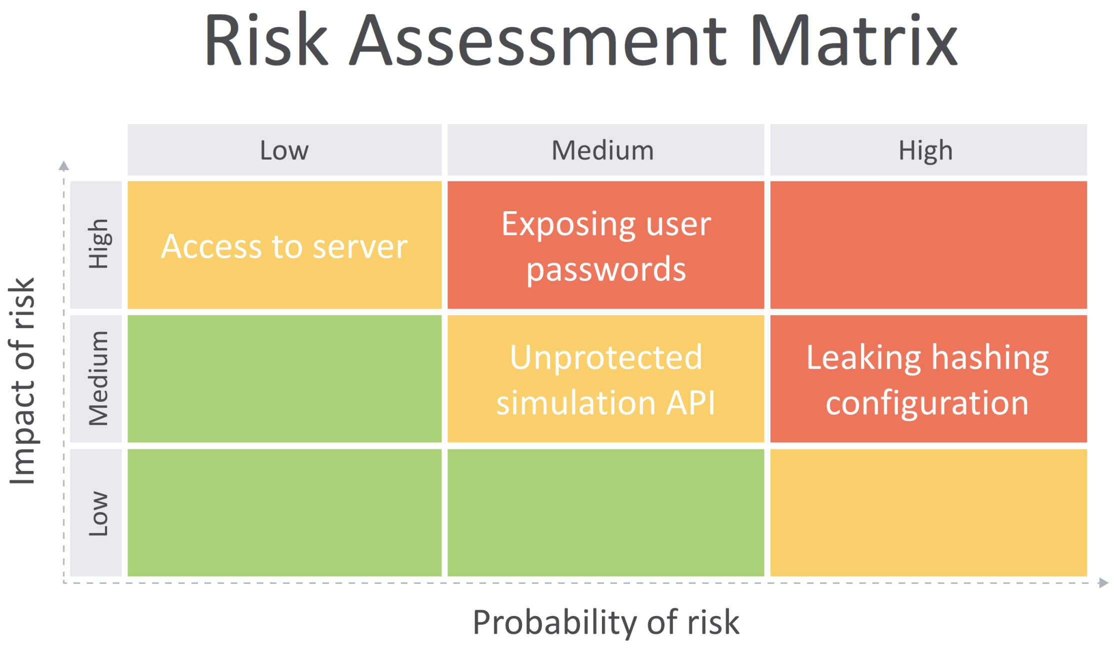


Figure 2.1: Risk Assessment Matrix

2.3 Pen test your system

We tried several different techniques while attempting to do penetration tests on our own system. Firstly we tried running the OWASP ZAP system to attack our own application, and then we considered security implications based on items from the OWASP top 10 security list.

The findings will be described in the sections below.

2.3.1 OWASP ZAP and Injection

After running OWASP ZAP it became apparent that there were some misconfiguration in the results sent from the server to the client. These were primarily leaking of "X-Powered-By", which allows users to know information about the stack that the server runs on, which potentially can lead to targeted attacks. Furthermore we need to enable Web Browser XSS Protection.

The alerts from ZAP can be seen in image 2.2.

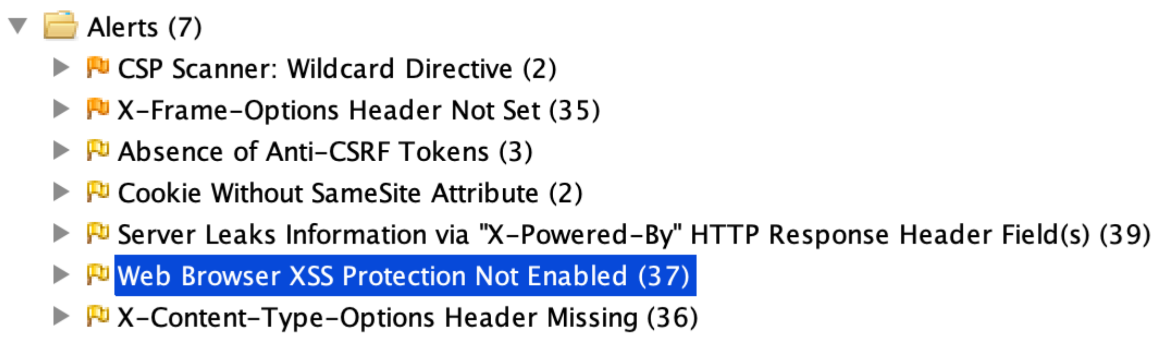

Alerts (7)
CSP Scanner: Wildcard Directive (2)
X-Frame-Options Header Not Set (35)
Absence of Anti-CSRF Tokens (3)
Cookie Without SameSite Attribute (2)
Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) (39)
Web Browser XSS Protection Not Enabled (37)
X-Content-Type-Options Header Missing (36)

Figure 2.2: Alerts found while attacking our own site with OWASP ZAP

We also made another interesting find while inspecting our logs after running the ZAP-attack. Specifically we noticed that the program attempted to inject statements, through tweet messages, into our application such as `ZAP';cat /etc/passwd;'`. Image 2.3 is a screenshot from our logging showing ZAP attempting to perform injection attacks.

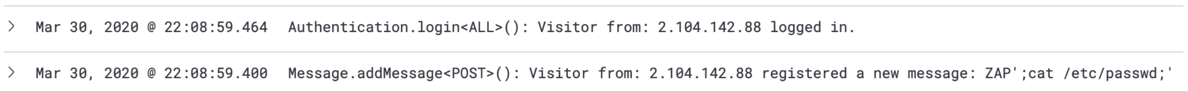

```
> Mar 30, 2020 @ 22:08:59.464  Authentication.login<ALL>(): Visitor from: 2.104.142.88 logged in.
> Mar 30, 2020 @ 22:08:59.400  Message.addMessage<POST>(): Visitor from: 2.104.142.88 registered a new message: ZAP';cat /etc/passwd;'
```

Figure 2.3: Logs recorded during ZAP attack

Through this, and inspection of our own code, we concluded that our application is safe from injection attacks since we do not handle any form of user input ourselves, and the fact that our ORM handles sanitation before inserting anything into the database.

2.3.2 Broken authentication

Since we did not find any issues related to our security with injection, we then decided to look at the second most popular security issue as described by the OWASP top 10 list: Broken authentication.

This led us to look at the login form, which we determined posed major security risks. These were as follows:

- Feedback from server notified user whether the password or username was wrong. When logging in the server should never notify whether it is a password or a username that is wrong since this could enable hackers to crack passwords more easily.
- There is no throttling on the input form, which can enable hackers to brute force passwords and thereby putting our primary assets at risk.

Based on these risks, we were able to determine that if we do not fix this, then we are at a very high risk of exposing our user passwords and user accounts which are our primary assets. Hence we had to fix this problem.

In the end we mitigated the risk by implementing limiting from IP-addresses, meaning that if a user types incorrect login information more than 5 times in 10 minutes, then they will be locked out from logging in the 10 minutes after the first logged attempt in order to prevent brute force attacks.

Furthermore we also removed the descriptive notifications when a user attempts to login, limiting it to either ”Invalid username or password” and ”You've entered incorrect login information too many times in a row, please come back later.”

3 Test the Security of Your Monitored Team

3.1 Pen test

During our penetration testing of Group J (`http://46.101.215.40:5000/`) we used the tool OWASP ZAP and ran a bit of manual testing on their site to determine possible vulnerabilities on their site.

Furthermore we noted that their site is served with 'http' and not 'https' which may mean that their users data is at risk when they access the site.

Configuration flaw A few pokes and prods at their websites revealed an admin user that was still active in a live production environment. (Username and password were both admin on their Grafana dashboard). While their admin user didn't have any specific credentials seemingly nor any privileges, an intruder wouldn't be able to do much harm. However, imagining a scenario in which an admin user would be able to moderate the forum. You could potentially have a major case on your hands with the havoc a single intruder could cause.

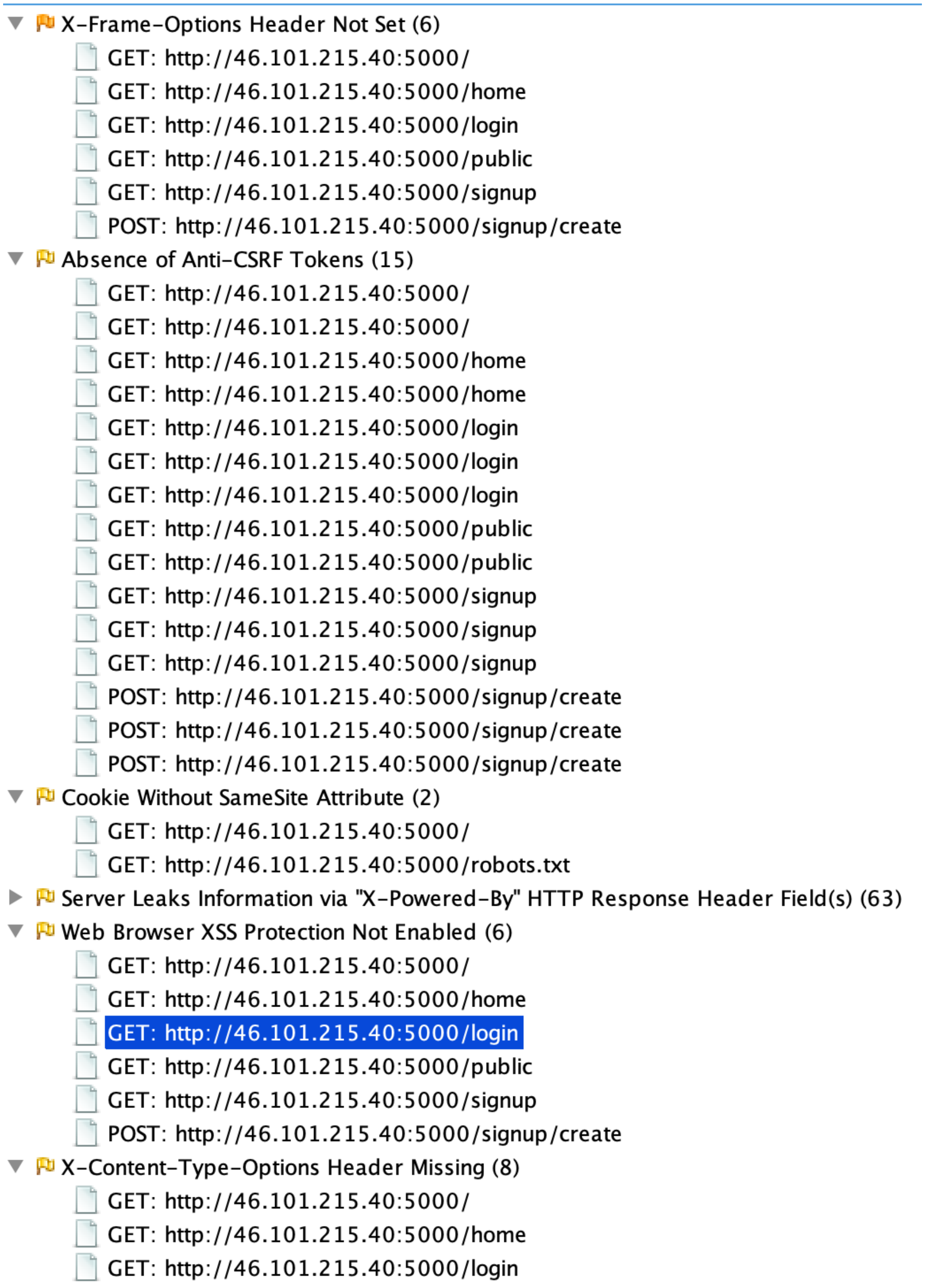
Login form Another finding on their site was that they keep anonymous which part is wrong in a login attempt. Saying it is either the username or the password, but not specifying which. This is keeping up to code regarding the user credential issues as expected, however there is one flaw in the form.

That is that there does not seem to be any sort of throttling, which means that a bot or an automated script could assault the site with repeated spam attacks, potentially setting up a Denial of Service Attack using their login page.

ZAP The results of our pen testing using ZAP can be seen in Figure 3.1.

As can be seen then multiple pages are not protected Cross-site scripting (XSS) which potentially could allow malicious people to inject client-side scripts into other users clients when browser the site. If their users and their data is seen as an asset to them, then action should be taken to resolve this security issue.

Furthermore, the remaining warnings generated by ZAP should be considered by Group J in order to determine if they pose a security threat severe enough for them to handle.



```
▼ X-Frame-Options Header Not Set (6)
      GET: http://46.101.215.40:5000/
      GET: http://46.101.215.40:5000/home
      GET: http://46.101.215.40:5000/login
      GET: http://46.101.215.40:5000/public
      GET: http://46.101.215.40:5000/signup
      POST: http://46.101.215.40:5000/signup/create
▼ Absence of Anti-CSRF Tokens (15)
      GET: http://46.101.215.40:5000/
      GET: http://46.101.215.40:5000/
      GET: http://46.101.215.40:5000/home
      GET: http://46.101.215.40:5000/home
      GET: http://46.101.215.40:5000/login
      GET: http://46.101.215.40:5000/login
      GET: http://46.101.215.40:5000/login
      GET: http://46.101.215.40:5000/public
      GET: http://46.101.215.40:5000/public
      GET: http://46.101.215.40:5000/signup
      GET: http://46.101.215.40:5000/signup
      GET: http://46.101.215.40:5000/signup
      POST: http://46.101.215.40:5000/signup/create
      POST: http://46.101.215.40:5000/signup/create
      POST: http://46.101.215.40:5000/signup/create
▼ Cookie Without SameSite Attribute (2)
      GET: http://46.101.215.40:5000/
      GET: http://46.101.215.40:5000/robots.txt
▶ Server Leaks Information via "X-Powered-By" HTTP Response Header Field(s) (63)
▼ Web Browser XSS Protection Not Enabled (6)
      GET: http://46.101.215.40:5000/
      GET: http://46.101.215.40:5000/home
      GET: http://46.101.215.40:5000/login
      GET: http://46.101.215.40:5000/public
      GET: http://46.101.215.40:5000/signup
      POST: http://46.101.215.40:5000/signup/create
▼ X-Content-Type-Options Header Missing (8)
      GET: http://46.101.215.40:5000/
      GET: http://46.101.215.40:5000/home
      GET: http://46.101.215.40:5000/login
```

Figure 3.1: Results from running ZAP on Group J's site

3.2 Discovering attacks on our own system

When searching through our own logs we were able to determine when the group monitoring us 'attacked' our system by inspecting our logging in Kibana.

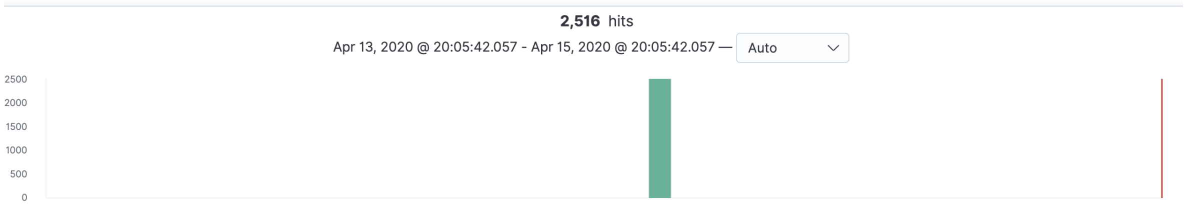
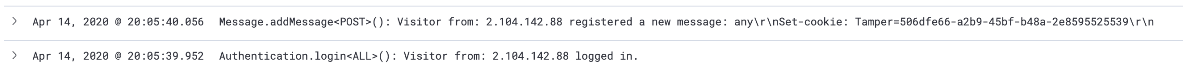


Figure 3.2: Graph of requests logged in Kibana, filtered to only display relevant results (i.e. hiding logs related to the Simulator)

They attacked us around **14. April 2020 22:00**, which can be seen in Figure 3.2 and Figure 3.3

It seems that our site was not harmed in the process, and hence we didn't respond in any way to this attack.



```
> Apr 14, 2020 @ 20:05:40.056  Message.addMessage<POST>(): Visitor from: 2.104.142.88 registered a new message: any\r\nSet-cookie: Tamper=506dfe66-a2b9-45bf-b48a-2e8595525539\r\n
> Apr 14, 2020 @ 20:05:39.952  Authentication.login<ALL>(): Visitor from: 2.104.142.88 logged in.
```

Figure 3.3: Example of logs recorded during the 'attack'

3.3 Introduction of vulnerability

We introduced a performance regression the **2. of April 2020** on our `/public` endpoint which caused the response time to be $> 1s$.

The regression can be found on `http://157.245.35.115/public`.

This regression was not reported to us from the group that has monitored us. The regression will be reverted no later than the **15. of April 2020 23:59**.