DD2394 Cybersecurity Project Report

Group 10

Gustaf Johansson, Kristin Mickols, Alexander Söderhäll, Daniel Workinn

HT22

Contents

1 Spam

Preventing spam is not only necessary for improving user experience, it is also a very important part of ensuring users aren't at risk of phishing attempts and other methods used by adversaries via spam.

1.1 Countermeasures

Three countermeasures to reduce spam were implemented on our platform.

Google reCaptcha

Google reCaptcha was implemented via the "Spam be gone" plugin, enabling this plugin prompts users with a checkbox during registration to prove they are no a robot. This is to ensure that no spam bots are able to sign up as a user and begin spamming. This countermeasure was chosen as to stop spammers before they can even begin.

Link prevention

After ensuring all users are real people, two countermeasures against real human spammers were implemented. The plugin "beep" was installed and configured to censor common curse words. Apart from curse words, the domains of common url shorteners as well as your typical "http://" and "www." prefixes were added to the filter to prevent users from posting links.
Preventing links stops adversaries from attempting to redirect an unsuspecting user to a malicious site. Such sites could be for example phishing websites or malicious sites executing unsafe scripts, both of which may harm a users Integrity and Confidentiality. While this implementation does not completely remove links from the platform (for example a user could post "google.com" and it would go through), this solution is supposed to make it harder for an adversary to post a link, and since URL shorteners are often used to mask malicious sites, filtering these will better help users to detect malicious links. Due to time constraints, as of right now these filters are for all users. A better implementation would be to implement more restrictive posting rules for new users while allowing regular users to post more freely.

Mass post prevention

Lastly, default site settings were configured to prevent users from mass posting. New users now has to wait 3 minutes from submitting a post until they can make another. Normal users has to wait one minute in between posts. Preventing mass posting and censoring common slurs is essential to ensure an inclusive and pleasant user experience.

1.2 Countermeasure difficulties

The "spam-be-gone" plugin offers more anti-spam services than just the Google reCaptcha, however some services were locked behind a paywall subscription. One such service is Akismet, which scans all user posts and comments for spam against its spam filter, and removes these posts.
Another problem arose as our project was done on the localhost and not a "real" forum, some services did not recognise the URL of our NodeBB and therefore the API key provided did not work properly. This was also the case with Akismet and also Project Honeypot which is a service that tracks malicious IP addresses and denies them access to your website.

Due to these limitations other, simpler, methods were explored and implemented. Google reCaptcha offers API keys for testing purposes and the beep plugin is installed and configured locally.

2 User authentication

User authentication is essential for Confidentiality, Integrity and Availability (CIA). Without it, unauthorized people may gain access to restricted systems and data. Bootstrapping is a common problem among web servers and makes it difficult with user authentication [1]. User authentication over web services commonly applies 'Something you know' authentication. This type of authentication needs a prover and a verifier. The prover, or user when talking about web services, states who they are to the verifier. This is usually through a username. The verifier, or server, asks the user to authenticate this username by sharing a secret that only the prover and verifier know of [2]. This is usually done through a password. However, this authentication method has vulnerabilities. An adversary could guess the secret and convince the verifier. Additionally, users are not good at choosing difficult passwords since they are hard to remember. This creates a problem where user passwords might be stolen easily by an adversary and as such poses a threat to CIA.

Furthermore, creating a web service with great user authentication security takes a lot of time. There are a lot of services, such as Google, that have invested great resources into making their user authentication secure. This could be taken advantage of when creating secure user authentication for other web services [1].

2.1 Countermeasure

Two countermeasures have been chosen. Firstly, two-factor authentication (2FA) has been chosen. 2FA strengthens the authentication process by adding an additional step. The second authentication usually consists of 'Something you have' authentication [3]. This could be a limited-time token on a phone or a physical hardware key for example. This makes it very difficult for an adversary

to threaten CIA since they would need to guess the password of a user and have access to their limited-time numerical token on the user's phone or the users physical hardware key [4].

Secondly, Open Authorization (OAuth) has been chosen. OAuth enables users to register/login by authenticating on that service instead. If the authentication succeeds it will relay this information back to the server of our service which then authenticates the user. This puts user authentication on another service and commonly big organizations like Google are chosen since they have invested a lot of time into making their user authentication secure.

2.2 Countermeasure implementation & configuration

The countermeasures were implemented on NodeBB running on Dasak VM operatingsystem along with MongoDB database. See `https://docs.nodebb.org/installing/os/ubuntu/` for installation instructions. See Appendix 6.1.4 for instructions on the installation of 2FA and OAuth.

2FA has been implemented with the 2FA plugin (see section 6.1.4 for links to both plugins), enabling the Google Authentication app to be used. The plugin has been configured such that admins are enforced to use 2FA, but not regular users. The greatest threat to CIA of the web service is that an adversary gets credentials for the admin user since they have greater authority over the web service. However, user CIA should still be considered and this discussion will continue in the next section.

OAuth has been implemented with the SSO-Google plugin. This enables users to authenticate with their Google account. Since Google can be argued to have a much more secure user authentication than we would build ourselves we argue that this authentication process provides users with more secure authentication. The plugin did not provide many configurations. However, we did configure it so that users who chose this way of authenticating do not need to verify their e-mail address when registering with their Google account.

We argue that these two implementations of user authentication strengthens the security of the services. Firstly, 2FA is much stronger and harder to overcome than simply "Something-you-know" (see section 2.1). Secondly, OAuth provides added security becuase it relies on Googles authentication which arguably is much safer than many other services (see section 2.1).

2.3 Countermeasure difficulties

There were no issues getting 2FA implementation to work. However, OAuth had it's share of difficulties in getting to work. We experienced two issues. Both issues occurred when a user tried to log in with OAuth. Firstly, there was an "uri_redirect_mismatch". Secondly, there was another error "too many

redirects". The solutions to those problems are described in `https://github.com/kmickols/DD2394Project/tree/userauth/setup`.

3 Data loss

Data loss occurs when data unintentionally or without consent is deleted or lost. This can be caused by either human or software. For example, an employee within a business can accidentally delete files without intending to. Another cause of data loss is ransomware attacks, which will be discussed below.

3.1 Ransomware

A ransomware attack is a type of cyberattack used to gain money from victims by forcing them to pay a ransom. Ransomware is a type of malware that usually encrypts data, and the victim will be requested to pay ransom to decrypt the data.[5] As paying ransom can be costly and a way of supporting criminals, it is generally advised against. Additionally, paying ransom does not guarantee a retrieval of all (or any) data. [5] Therefore, it is wise to regularly back up data to be prepared in the event of data loss.

3.2 Countermeasure

The chosen countermeasure for this problem was backing up data. If data is being lost or tampered with, a backup can be used to restore the data. When doing backups, it is good to do them regularly, so the data that's being restored is as up to date as possible. In this implementation of a backup solution, backups are being made every day at 23:59. The idea is to store this data on a backup server. Data is stored for 30 days before it is removed, but this can easily be prolonged or shortened. What to keep in mind is that the longer the data is stored, the more memory is required. In this project, the memory used for each backup is not significantly large, but for a larger project it could be costly to keep very old data.

3.2.1 Configuration & implementation

The daily backup includes the database and the NodeBB uploads-folder, where uploaded images and files are stored. The database and the uploads-folder is what NodeBB recommends doing backups of, explaining the choice of files to back up. This solution requires using MongoDB as database for NodeBB, as it uses mongodump – a utility that creates a binary export of the database's contents. To do a daily backup, a bash script was written, which was scheduled using cron. Crontab was set up to schedule the bash script to run at 23:59 daily. The bash script creates a folder named backup_<yyyy-mm-dd> with the current date, which mongodump and the upload-folder is exported to. It also automatically removes folders that are more than 30 days old. In the script, it is specified where the data should be exported. The initial idea was to use

Dropbox, but this was changed to simply a (fictional) backup server. This decision was made as using Dropbox seemed redundant and having a backup server is more fitting for larger projects. This could easily be changed to a Dropbox-folder if desired and if Dropbox Desktop is set up on the computer, as it only requires the files to be exported to the Dropbox-folder. Please see Appendix 6.1.1 for full implementation details.

3.2.2 Difficulties

Initially, NodeBB was setup with a Redis database. Redis however does not provide a similar feature to mongodump, requiring the project to be reconfigured using MongoDB instead.

4 Brute-force Attack

A brute-force attack consists of an attacker trying bypass restrictions of a system by entering many passwords for an account with the hope of eventually guessing correctly. This is often done in a systematic manner, where all passwords are checked until the login attempt is successful.

The impact of a successful brute-force attack can be a loss of the whole CIA-triad (Confidentiality, Integrity and Availability) depending on what system and user was compromised. More concretely, this could lead to identity theft, loss of data, having data being altered or downtime for the system or services relying on it.

4.1 Chosen countermeasures

In this project, two countermeasures have been chosen to minimize the risk of brute-force attacks being successful in targeting our system. These countermeasures are requiring the use of strong passwords and throttling login attempts.

Password Strength

There are four different categories of characters available for passwords; numbers, lowercase letters, uppercase letters and symbols. Passwords becomes stronger by using all four categories as well as many instances of them, i.e. longer password (see Appendix 6.2). This is due to the fact that a password becomes exponentially harder to guess with the number of possible characters and length: $(\#\text{ pos char})^{(\text{length})}$.

We therefore force our users to adopt a complex password, using all four categories of characters, and with minimum password length of eight characters.

Throttling Login Attempts

Throttling login attempts, or login throttling, is when you limit the number of login attempts possible for a specific user ID or IP address. This is most commonly implemented by setting a fix number of possible attempts for a user ID or from an IP address. Other possible implementations are introduction of time delay between login attempts, grows with the number of failed attempts, or adding complexity, such as CAPTCHA, after a number of failed login attempts. For this countermeasure, it's important to define rules which minimizes the limiting of genuine users, but drastically slows down malicious ones.

In our implementation of this countermeasure, we use three counters and a 24 hour ban:

- User ID and IP pair: ban after 10 consecutive failed attempts
- User ID: ban after 50 consecutive failed attempts
- IP: ban after 100 consecutive failed attempts

4.1.1 Reasoning

The combination of the two chosen countermeasures completely nullifies any attempts of maliciously getting access to our platform by means of a traditional brute-force attack. The first countermeasure, enforcing strong passwords, ensures that an attacker needs to make hundreds of billions of guesses before gaining access (95^8). The second countermeasure, banning user ID's and IP addresses after too many failed attempts, in combination with the first, means that it would take millions of years for an attacker to gain access. We also believe that the rules defined for banning user ID's and passwords are non intrusive for genuine users, since a real user often tries to reset his/hers password after very few failed attempts.

4.2 Implementation of Countermeasures

The implementation of the first countermeasure was very straight forward: while logged in to NodeBB as an administrator, navigate to the settings tab and then users. On this page you can enable a strong password setting which forces users to adopt strong passwords. You can also set minimum password length, which in our case was set to eight character. For a full description of the implementation of this counter measure, please see Appendix 6.1.2.

We could not find a suitable plugin for the second countermeasure and chose therefore to implement it by changing the source code of NodeBB. The file that was changed can be found under path /nodebb/src/user/auth.js. Please see Appendix 6.1.2 to see how the code in the file was changed.

Having to change the source code meant we had to overcome difficulties such

as understanding the code structure of the platform, where and how authentication of login attempts are done, as well as adding our own code and coming up with a suitable policy for banning user ID's and IP addresses. We overcame these challenges by looking participating in discussion forums of NodeBB and throttling login attempts and spending a lot of time with the code.

5 Group members contributions

5.1 Alexander Söderhäll

I contributed to the project foremost by being first responsible for the user authentication problem. I first started with simple research and past knowledge to understand the problem of user authentication in the context of the NodeBB service. Then I began researching possible plugins that would countermeasure this. I found the 2FA plugin already installed on NodeBB and reasoned it to be an ample countermeasure for the problem. I did not have much trouble installing and configuring it (implementation) and have tested its usage on the admin account multiple times. It works as expected.

After the supervisor meeting it was recommended to look for further countermeasures for user authentication and again I started the research process. I found the OAuth plugin ample since it utilizes Google which is a large and well-known company. The installation was without problems but configuration took quite some time. I had to configure nginx and the "config.json" within the NodeBB directory to properly set up the rerouting of localhost to another address. This was because Google API would not allow localhost as a redirect address. Once that was done there was another problem with getting 2FA and OAuth working together which I was never able to solve. However, after reconfiguring 2FA to only enforce it on admins the OAuth authentication process worked without errors or problems. In this report, I've contributed by writing sections 2, 6.1 and 6.2 which seemed justified given that I was mainly responsible for implementing those countermeasures.

Furthermore, I've contributed with some things on brute-force attacking and data restoration. Firstly, regarding brute-force attacks I helped a bit by inspecting the NodeBB source code and discussing a bit of the issue we we're experiencing when trying to add code. Secondly, for data restoration, I've discussed the countermeasure and how it might be implemented. Other than that I've contributed to all problems by discussions during meetings.

Lastly, I've contributed by engaging in discussions, meetings and general documentation with my other group members. Such as writing notes during our supervisory meeting, setting up some other documents on our shared Google Drive folder, and discussing NodeBB in general and the problems during meetings.

5.2 Daniel Workinn

My main contribution of this project was being responsible for the brute-force attack problem. The problem was solved by first researching the topic. When I had a good understanding of the problem of brute-force and what countermeasure I want to deploy (password strength and login throttling) I investigating if

there was a simple countermeasure to deploy using NodeBB's plugin store. Since no plugin for this problem was found, I decided to spend time understanding the platforms code base and see if I could add my own code for throttling login attempts. After quite some time I understood the code base and found where to add the code (/nodebb/src/user/auth.js). However, after adding my own code I got stuck with the application crashing on startup. Luckily Alexander volunteered to spend time with me discussing the issue and we quickly found the problem.

I also contributed to the group work by joining all joint meetings and discussed all the selected problems throughout all the stages of the project. From a group-work perspective, I also contributed by assisting in organizing and planning meetings. My last contribution was assisting with the data restoration problem by discussing the issue, what counter measures are usually deployed in real life, and what counter measures makes sense for our project as well as how to implement them.

5.3 Kristin Mickols

I contributed to the project mainly by being responsible for the data loss problem. I started out by doing some research on different approaches on how to do backups for NodeBB, and found out that there was no real solution in terms of plugins or similar. I investigated using Dropbox as the solution, but after the supervisor meeting, I was told to further expand the solution. Therefore, I decided to write a script that would export what I through NodeBB:s documentation found to be crucial to backup – the database and the upload folder, which contains all uploaded images and files. I ran tests to ensure that the solution worked as intended.

Prior to dealing with the data loss problem, I had the ambition to solve the problem with user registration emails from the NodeBB forum going to the spam folder, by using the third-party emailing service SendGrid, which I found a NodeBB plugin for. However, I had too many obstacles that I realized it would be wise to focus on a different problem. This was a conclusion that was made after spending hours without any success, but rather the opposite, as NodeBB was refusing SendGrid's API key and it all resulted in not being able to send any emails at all. Eventually, we discussed within the group that the initial idea should be discarded and that we should focus on another problem instead. From here we decided to add the problem of data loss.

As the rest of the group, I participated in and planned all of the group meetings and attended the supervisor meeting with a TA. I have also been very active in our Discord server to discuss different project related topics with the rest of the group. I was also responsible for setting up all tools necessary for the project – cloning the NodeBB GitHub repo, setting up a Google Drive, creating and inviting everyone to a Discord server for the project, as well as setting up and

inviting everyone to an Overleaf document.

5.4 Gustaf Johansson

I contributed to the project mainly by focusing on the spam problem. I began by searching for plugins related to my topic and quickly found the "spam-be-gone" plugin. After installing the plugin and researching the services it was offering I began the process of getting API keys for each service to try them out and get an overview of what services would be suitable for our platform. This is when I encountered the problems discussed in section 1.2, so I settled with using the Google reCaptcha solution.

After meeting with the supervisor it was recommended to also find a solution for actual users posting spam. My initial plan for this was to implement the more sophisticated solutions offered by spam-be-gone, like Akismet, but it did not seem to work with our local version. After trying to figure out how to overcome the problems I was facing and researching some more solutions, I settled with the "beep" plugin. After configuring beep for a bit I felt like something else was needed to make the solution a bit more presentable, so that is when I explored the admin settings and made appropriate configurations to the platform.

Other than this I have participated in our group meetings as well as our meeting with a supervisor. I have also been active in discussions on our discord channel, often providing my input on how I interpreted the project instructions as I think everyone felt a bit confused about what to do in the beginning.

5.5 Summary of process

5.5.1 Labor division

We discussed different problems and together we decided on four problems. Of these four problems, each group member was assigned a problem they had main responsibility. We had multiple meetings where we discussed the progress, showed what we had done to ensure that everybody was on track and discussed different solutions. Our initial idea was to set up work meetings where we would sit together and intensely work on the problems as a group, which would be helpful if anyone got stuck. However, it was hard to find a time when all four group members would be able to participate in person for a longer period of time. Due to this, most of the work hours were set up individually by each group member. We used a Discord server to discuss and share ideas or problems that occurred during the individual work sessions, which worked out fine as all group members actively participated and responded in the chat.

5.5.2 Time plan

We aimed to follow the suggested time plan:

- 1 day installing & familiarize with the system
- 3 days countermeasures (select problems, countermeasures, apply, configure, write into report, document)
- 0.5 day wrap up report, documentation
- 0.5 day seminar prep and seminar

Overall, the actual time spent did not differ too much from the initial time plan.

5.5.3 Design decisions

We decided to have a customized NodeBB setup for each member of the group. This is because some members experienced problems with the setup of NodeBB for different databases and operating systems. As such, each team member has a customized OS and database for NodeBB. Some used the Dasak VM and some macOS along with either Redis or MongoDB.

5.5.4 Problems

We all spent some time exploring and familiarizing with NodeBB prior to deciding what problems to tackle. We then held a meeting where we decided on the problems spam, data loss and user authentication. We chose these three problems among the ones listed because the group felt most comfortable implementing countermeasures against these after we had investigated the NodeBB system. We did start with user registration instead of data loss. However, as the intended solution by using SendGrid as a third-party emailer service did not work out as we had expected, this idea was discarded due to too much time spent on an error caused by NodeBB that seemingly could not be resolved. The issue was that NodeBB would not accept the API key. As such, we decided to switch that problem to data loss after a discussion, instead of spending more time without any progress. Our individual problem came from a discussion of the system and realizing that brute-force attacks could be possible on NodeBB. Furthermore, since NodeBB denies the service of a user if too many attempts has been attempted independently of the IP address this enables attackers to employ a DOS attack. Having this in mind, we decided our individual problem to be brute-forcing of user credentials.

6 Appendix

6.1 GitHub

https://github.com/kmickols/DD2394Project
Each problem has a seperate branch, which can be found below.

6.1.1 Data loss

https://github.com/kmickols/DD2394Project/tree/data-loss

Please see folder HowToGuide for configuration details.
https://github.com/kmickols/DD2394Project/tree/data-loss/HowToGuide

6.1.2 Brute-force

https://github.com/kmickols/DD2394Project/tree/brute-force

Please see folder HowToGuide for configuration details.
https://github.com/kmickols/DD2394Project/blob/brute-force/HowToGuide/README.pdf

6.1.3 Spam

https://github.com/kmickols/DD2394Project/tree/spam

6.1.4 User authentication

https://github.com/kmickols/DD2394Project/tree/userauth

Plugin installation instructions:

https://github.com/kmickols/DD2394Project/tree/userauth/setup.

6.2 Exponential nature of Password Strength

Number of Characters	Numbers Only	Lowercase Letters	Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters	Numbers, Upper and Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	Instantly	Instantly
7	Instantly	Instantly	2 secs	7 secs	31 secs
8	Instantly	Instantly	2 mins	7 mins	39 mins
9	Instantly	10 secs	1 hour	7 hours	2 days
10	Instantly	4 mins	3 days	3 weeks	5 months
11	Instantly	2 hours	5 months	3 years	34 years
12	2 secs	2 days	24 years	200 years	3k years
13	19 secs	2 months	1k years	12k years	202k years
14	3 mins	4 years	64k years	750k years	16m years
15	32 mins	100 years	3m years	46m years	1bn years
16	5 hours	3k years	173m years	3bn years	92bn years
17	2 days	69k years	9bn years	179bn years	7tn years
18	3 weeks	2m years	467bn years	11tn years	438tn years

References

[1] D. Bosk, *Bootstrapping*, Nov. 2020. [Online]. Available: `https://github.com/OpenSecEd/auth/releases/tag/v2020`.

[2] D. Bosk, *Authentication: Something you know*, Nov. 2020. [Online]. Available: `https://github.com/OpenSecEd/auth/releases/tag/v2020`.

[3] D. Bosk, *Authentication: Something you have*, Nov. 2020. [Online]. Available: `https://github.com/OpenSecEd/auth/releases/tag/v2020`.

[4] D. Bosk, *User-to-machine authentication*, Nov. 2020. [Online]. Available: `https://github.com/OpenSecEd/auth/releases/tag/v2020`.

[5] R. Guanciale, "Malware (Lecture in course DD2395, KTH)," 2020.