# Cybersecurity Project
# DD2394/DD2391

Ansh Bhalla, Emil Franzell, Beata Johansson

## Problem 1: Denial of Service (DoS)

### Threats

In this section, we explain our own selected problem for the NodeBB platform. We found that the NodeBB application is vulnerable to a Denial of Service (DoS) attack. In this case, the DoS attack would entail locking all accounts on NodeBB for a long time. To perform the attack, the threat actor can exploit the way the login page works. At the moment, the login page allows five failed login attempts before it blocks new login attempts for 30 seconds. A failed login attempt is defined as a correct username and a wrong password. If we are allowed to speculate, the idea of this functionality can be used to prevent brute-force dictionary attacks.

However, this means that an attacker can intentionally spam the login page with the wrong passwords for all users. This is possible to do since the users of the forum are visible to anyone who can access the domain of the forum. Therefore, the login page seems like an attractive target for an attacker.

The impact of this kind of DoS attack would be that all accounts would be blocked for a long period of time. This includes the admin account. If the forum serves an important purpose to a business, this could possibly have a large impact on its operation unless it is quickly patched.

### Countermeasure

The countermeasure that was chosen to counteract this problem was to extend the authentication process when logging into an account. This can be accomplished by verifying that the user is not a non-human. In practice, this is done by using the Google service Google ReCAPTCHA v2. The Google ReCAPTCHA v2 is used for forcing users to authenticate as humans. It does this by using a check box, which the user presses when logging in (Figure 1). If the user is suspected of being non-human, a CAPTCHA challenge will be prompted for the user to complete. How the Google ReCAPTCHA v2 separates suspicious actors and human users are not known, but it looks at several factors to do this[1].

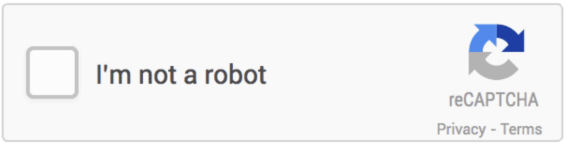


*Figure 1: Google ReCAPTCHA v2 check box.*

[1] https://developers.google.com/recaptcha/docs/versions#recaptcha_v2_im_not_a_robot_checkbox

### Reasoning about the choice of countermeasure

For this problem, the password authentication, and limit for the failed login attempt are kept as they are implemented. The protection against dictionary attacks is reasonable but combined with showing all usernames becomes a threat. One way to eliminate the risk is not letting not logged-in users view all registered users on the page. Doing this would remove the purpose of the site, the choice of countermeasure was instead focused on adding an additional authentication layer for human authentication.

### Steps for Configuration

See file problem1_configuration_steps in github repo.

### Difficulties

The difficulty is that adding human authentication doesn't solve the entire problem of DoS attack. Delegating the problem to another service like Google, and using their services, can be a controversial choice. This is mainly because we cannot be certain of how the service differentiates bots from human users. We can only assume that it includes many different factors. The Google ReCAPTCHA v2 has been used since 2014, and there are ways to bypass it. Therefore, an important aspect of this countermeasure is that it does not entirely secure the website from DoS attacks. However, it will remove some parts of the DoS threat by adding an additional step to the authentication process.

## Problem 2: User Authentication

### Threats

Within the NodeBB web application, each user creates their login and password to gain access to their account. When a user creates this account, they are added to the list of registered users under the web application. One of the pages of the application, "Users", lists all the users registered on the forum website, without a logged-in account. Because of this, attackers can view the usernames of all users on the website. In the case of an attacker attempting to gain access to the account, the only credential left to attack is the password. An attacker or security threat can gain access to an account by guessing or stealing the password, for example, through social engineering. Thus, the attacker can log in to a user's account even though it is not the actual user logging into the account. With this access, the attacker can make posts on the forum, falsely portraying it as the victim's account, demonstrating a threat to user authentication.

### Countermeasure

A possible solution or countermeasure for this is to use two-factor authentication within the application. Two-factor authentication allows for an extra authentication step in the login process. Once a user enters their username and password to log in, the application will request an additional code to gain access to the account. This code is sent to a personal device

that only the actual user has access to. The code may be sent through the user's phone number, email address, a piece of hardware such as a Yubikey, or even an external application that generates this code such as Okta or Google Authenticator. This proves to be a successful countermeasure against attackers gaining access to a user's credentials. Even though the attacker may have access to the user's credentials, they will not be able to log in to the account as they won't have access to the external two-factor authentication application. Two-factor authentication provides an extra step to block attackers from entering the account even if the login information of a user has been leaked or compromised.

**Steps for Configuration**

See file problem2_configuration_steps in github repo.

**Difficulties**

Our original approach to implement two-factor authentication was to alter the application javascript code, to send a verification code to the user's registered email address. This proved to be difficult as described in the summary of our work process. We found a successful workaround to be to utilize the plugins provide by nodebb. One of these plugins was "nodebb-plugin-2factor" which enables two-factor authentication when a user is logging into their account. Another difficulty that was encountered was forcing all accounts in the web application to use two-factor authentication. Without making two-factor autehnticaion apparent to the user, they may not realize it is possible to enable, or they made not choose to use it, which would provide security vulnerabilities in the web application. After some exploring through NodeBB admin page, it was found that the admin can enable two-factor authentication for all users based on the type of account. This ensures that all users who newly register to NodeBB and even for existing users to setup two-factor authentication.

## Problem 3: User registration: notice that often the email is received as spam

### Threats and Countermeasure

There is a risk that emails end up in spam whenever a user submits their email address for registration. This is due to the fact that most email servers guard against sender spoofing in emails. DKIM or "DomainKeys Identified Mail" is a common solution to this. DKIM basically ensures that the sender of an email owns the domain with which the sender name is registered.[2] It accomplishes this by signing the email with a private key. The public key is available as a DNS record, allowing any receiver of an email to access it and verify easily. While DKIM could be manually implemented, it seems like a large hassle. Instead, we chose to implement it using Amazon SES on AWS.

While having emails end up in spam isn't the end of the world, the root cause is a real issue. The reason emails end up in spam is because the receiving server has somehow classified them as spam. This means that our email was deemed as not trustworthy for some reason. If our service, which we know is the real deal, is deemed as untrustworthy, then we have no advantage over random Russian robots sending phishing emails. This is why we need DKIM. With DKIM, we can provide undeniable proof of ownership of the sender address. This makes our emails inherently more trustworthy than someone who is spoofing their sender address, which will make it more difficult to send phishing emails.

### Steps for Configuration

See file problem3_configuration_steps in github repo.

[2] https://www.dmarcanalyzer.com/dkim/

## Team Process

The work started out with the group reviewing the project specification and familiarising with the canvas page. A while was spent discussing the scope of the project and trying to understand what we were supposed to do. After our first meeting, we decided on another date a few days later. Before that meeting, everyone was tasked with downloading and setting up NodeBB on their own machines.

At the next meeting, we got to work trying to clearly define and decide on the problems we were going to solve. We managed to find a few vulnerabilities or shortcomings in the software which we could decide on fixing. From trying a brute-force attack on the login form, we discovered that sign-ins could be disabled for individual accounts by an attacker, effectively enabling a Denial of Service attack that could easily be automated. This we decided as our own problem. Initially, Ansh got tasked with adding 2FA, Beata got spam and Emil got the DoS.

When we first started work on fixing the issues, we began digging in the source code to find what we could change to accomplish our goals. After viewing the sheer number of folders and files, we thought we would have to spend a great amount of time on hacking the Javascript code. We did however write continuously in the report as we went, which let us manage our time better.

After going our separate ways, we all individually did some work and research on NodeBB and its codebase. That is until we found out about plugins. We were unsure whether plugins were fine to use since the task we had initially set out to complete was gargantuan. When we got confirmation of the ability to use plugins, we got to work finding the best ones for the job. We managed to implement most of the solutions using the plugins, but due to some plugins being outdated, we had to reconsider our scope.

Rather than Emil completing the DoS fix using the initially decided on solution, Beata would address the DoS using a reCAPTCHA plugin. Emil would instead work on fixing the email issue. Once we all set our minds to implementing the various plugins, it was a very short process to completion. Since we did not all work with the plugins at exactly the same time, not all group members completed the work simultaneously. This meant that we could share our experiences with each other and make the work more efficient.

The last thing we did was ready a GitHub repository for submission. We had all worked independently of each other, which meant that we had not collaborated in any git repository together.

## Group Member Contribution:  Emil Franzell

I attempted to take a leadership role in the project, since I work with web development at a daily basis. I, along with my peers, did a lot of brainstorming about the possible issues and solutions and with interpreting the list of problems present. I took part in discovering and defining the problems and possible solutions.

When we had defined the problems and came up with some candidates for solving them, I dug deep into the codebase and attempted to find ways to make changes to the code in order to fix the problems. I was baffled at the sheer size of the project and started researching and reading the available resources about NodeBB. I found that there was something called "Plugins" that you could make, so I thought about making our own plugins. However, when reading the list of plugins, I discovered that there were already some plugins that implemented the solutions we had come up with. For example, we had decided to step away from the NodeBB-included username/password verification and use magic links instead (That was my task). I found several SSO plugins and an OIDC plugin with special support for a self-hosted service called FusionAuth. I was unsure if we were allowed to complete the project with this little effort (We were originally convinced that we had to hack the mainframe). For that reason, I wrote a message in the Canvas discussion page (that has yet to get a response).

After talking to my peers, I decided to get to work using the plugins. Unfortunately, they were outdated and did not support the stable release of NodeBB, so I spent some hours pulling my hair out before giving up. I then instead decided to address another problem with the site. Namely that emails bounce. I had recently (a week before) implemented a solution with NodeMailer and Amazon SES on my own website, so it seemed like a surmountable task. Sure enough, it wasn't very bad. I looked through the settings and found that there was an easy, built-in way to provide your own SMTP credentials to NodeBB. Since I couldn't (read: don't want to) use my company domain to send emails from a fake forum site, I had to verify my personal domain with SES.

The reason why I chose SES is that it has built-in support for DKIM. DKIM assures the receiving email server that the sender of an email has permission from the domain owner to use the sending email address. This is necessary because anyone can send email from any address and if some rando in Russia can send an email from my domain that is just as trustworthy as my own emails, that makes it extremely easy for them to execute phishing attacks.

## Group Member Contribution: Beata Johansson

We had group meetings during the beginning of the course. I participated in all group meetings that we had during the project. We started by brainstorming ideas of potential problems together. We also spent a meeting where we analyzed the code base for NodeBB to find security threats. Although I spent time trying to find security problems in the code, I could not find any specific problems there that we used. However, when I looked through the website, I came up with parts of the end problems.

We divided the problems so that we were responsible for one problem each. In the beginning I was responsible for a problem which was not included in the final report. This problem was about spamming user registrations on the register page. The countermeasure that was chosen for this problem was to add human authentication to the user registration. I researched potential solutions to the problems, and searched for plugins that would solve this problem. The team eventually found the NodeBB plugin “nodebb-plugin-spam-be-gone“ which used google ReCAPTCHA v2.

After the supervision meeting with the Teacher Assistant in the course we were encouraged to change the initial problem. Therefore, we changed it to be about a problem which we had discussed in the beginning of the course, namely, the DoS attack. I continued to use the same reasoning for the countermeasures on the DoS attack problem, as it could be used on both of these problems.

When we had a countermeasure, I installed it as the admin user on the NodeBB page. All necessary preconditions for this plugin were also done by me. However, the suggestion of using the plugins was made by another group member. Finally, I also wrote about the problem in the report.

## Group Member Contribution: Ansh Bhalla

As a team, we discussed and looked through the list of problems to brainstorm possible ideas. After looking through the list of problems, we selected a couple of the issues and defined what some possible countermeasures could be. After selecting our problems, we assigned each problem to a group member, where I took the responsibility to implement the two-factor authentication countermeasure to solve the problem of user authentication.

We spent a couple of meetings together trying to look through the codebase and see how we could implement these countermeasures. I looked through the folders of the codebase to see if there was a way to send an email to the user to verify their login to the account. After spending a lot of time on it, we still could not figure out where in the code the countermeasures should be implemented for each of our respective countermeasures.

Fortunately, one of our team members suggested using the NodeBB plugins that were provided in the application itself. After looking into this, I found for the user authentication issue, there existed a plugin called “nodebb-plugin-2factor” which enabled two factor authentication when a user logged into their account. After installing the plugin locally, I setup the process on the admin account at first to see how it works. I decided to use the Google Authenticator mobile application as a way to setup two-factor authentication on the accounts. I was able to setup the two-factor authentication successfully on the admin account and tested it with the login process.

Another issue I encountered was that the two-factor authentication was not automatically enabled for all users, and I felt that it was necessary to have this to prevent any security threats. After looking through the plugin options, I found a way to apply the two-factor authentication to all accounts registered on the NodeBB application. Therefore, I tested this on different accounts to ensure that two-factor authentication was enabled. This testing involved checking that the countermeasure was enabled for preexisting users and new users in the future, and both successfully worked.

This plugin proved to be successful as a countermeasure to threats of user authentication. The admin account provides control of the two-factor authentication for the accounts and can ensure that all accounts have it enabled. It is also a simple process from the user side as they can easily perform the authentication with their mobile devices using an application such as Google Authenticator.