Authors : Group 8

Cybersecurity Project Report

Analysis and discussion of security vulnerabilities in an open-source application

Ada Hysa

Damla Gözük

Riccardo Romagnoli

Hugo Teixeira de Castro

Table des matières

.....1
Introduction.....3
I. Analysis of problems.....4
I.1. Spam.....4
I.2. Ransomware, data loss.....5
I.3. Firewall.....6
I.4. DoS with sockets.....7
II. Member contribution.....8
II.1. Ada.....8
II.2. Damla.....9
II.3. Hugo.....10
II.4. Riccardo.....11
III. Summary of the organization.....12

Introduction

The main goal of this project was to apply all the different skills and knowledge we gained during the courses in a real, concrete example. In groups of 4, we were to analyze a software from a cybersecurity perspective to identify potential vulnerabilities or problems, before suggesting our own, documented solutions and countermeasures to prevent any such threats.

The application we had to use is NodeBB, an open-source forum software designed for communication between groups of people, for example friends or professional discussions. It is a complete, well-documented and often updated system that offers a wide range of functionalities, including message posting, user registration and authentication, file upload or even mail communication. It is also highly maintainable with easy-to-implement plugins that we can either build ourselves or download.

We thus had to test and understand the inner mechanisms of NodeBB and think of some ways to improve the system and make it more secure, either by fixing vulnerabilities or by introducing measures of prevention. We were already given a list of different potential issues that we could pick from, but also had to come up with a problem of our own that we thought was relevant. In the end, we chose to discuss the following vulnerabilities: Spam, Ransomware/Data loss, Firewall, and added another issue on how to prevent a particular type of denial-of-service attack.

In this report, we'll go through every single one of these problems, and for each one carefully describe it by defining for example how it can be exploited, the impact and consequences of a successful attack, as well as the reasoning behind our choice of countermeasures. In the Git repository, you will also find more detailed documentation with how to reproduce certain vulnerabilities and the configuration steps to implement the solution.

This project was also a team project, which is why we will also describe our organization inside the group, how we divided the work, how we communicated, the contribution of each individual member as well as the different difficulties we faced in the process.

I. Analysis of problems

I.1. Spam

When registering in the NodeBB forum, a user can post anything from links to uploading files. As an admin, there aren't much restrictions you can impose, besides some basic settings. Therefore, without a proper countermeasure, the forum could be affected by spam.

There are many reasons why and ways how an attacker can target the forum by spamming. The most common is advertising. The attackers' goal in this case is to promote certain goods or services for their own financial benefit. They can do this by creating normal users accounts in the forum and posting announcements continuously, where they introduce the product/service and add links to their webpage. An attacker can also follow other users in the forum and send them messages directly or reply to their posts, thus flooding their feed with unsolicited advertisement. Besides advertising, an attacker can spam users in the forum to perform financial scams. This can be done by sending users direct messages notifying them that they have won some kind of price or lottery or by using the famous "Nigerian letters" scam.

Secondly, another preferred method is phishing. A malicious user can publish fake notifications with links from different providers or organizations and encourage other users to enter their personal information. By redirecting the users to a fake login page, the attacker can steal their passwords, bank account details or any other confidential information. Next, an attacker can publish links to infected websites, designed to lure users to click on them and become victims of viruses, Trojan horses or other malicious programs.

With all this, it is essential to find a countermeasure against spam in the NodeBB forum. Luckily, there was a NodeBB plugin against spam installed with the version of NodeBB that I was using. I enabled the "Spam Be Gone" plugin by executing "nodebb activate nodebb-plugin-spam-be-gone" in Git cmd and restarting NodeBB. After that, Spam be gone was visible in the forum's plugins page and ready to be configured. This anti-spam plugin is quite complex, as it includes many dependencies. It makes use of Project Honeypot, Google reCAPTCHA and hCAPTCHA to check every user registration, Akismet to check every user post for spam and StopForumSpam to report malicious users when needed. For the scope of this project and testing purposes, I have only configured Google reCAPTCHA and Akismet in my local forum.

To check that every user who registers and logs in the forum is legitimate, I used Google reCAPTCHA, as it is reliable and widely used. It uses advanced risk analysis techniques to tell humans and bots apart. To configure it in my forum, I needed to sign up for an API Key pair. The public key is used to invoke reCAPTCHA on my forum, while the secret key authorizes communication between my forum and the reCAPTCHA server to verify the user's response. I chose reCAPTCHA v2 checkbox type and added localhost and 127.0.0.1 as my domains. After generating the keys, I added them to my forum and created new user accounts to check that it was working properly (Read documentation page for more details).

Next, I decided to use Akismet to check every post of every user in the forum for spam content. Akismet is a popular service that filters spam by combining information about spam captured on all participating sites, and then using those spam rules to block future spam. To configure Akismet on my forum, I needed an API key, which enables my forum to communicate with the Akismet servers. I created an account for personal use on akismet and received my API key, which I added to my forum. To test the service, I made several posts with several users and Akismet received all API calls. It was able to prevent me from posting when it flagged the content as spam, as well as detect spam after posting (Read documentation page for more details).

However, it is important to note that there are limitations to this countermeasure. Because I am running the forum locally, all API calls made from my machine will pass as legitimate. Therefore, we must understand that results obtained on my testing environment are not as accurate and reliable as when using real traffic.

I.2. Ransomware, data loss

In a Ransomware attack, hackers may deploy malware to a targeted system and may encrypt data. This data is only decrypted by the decryption key the hackers have. And may be given to the victim if the ransom payment is done. This type of attack can target an individual, a business, or a business network.

Permanent data loss may occur if precautions are not taken beforehand. The crucial way to recover from such an attack without paying the ransom required by the hackers is to have a proper backup system that frequently takes the backup of the system, files, etc. When such an incident happens, the owner/user of the system can recover the data. Ransomware is an important topic to cover because, especially nowadays, ransomware attacks are one of the most common attacks specifically targeting and affecting data and data compliance. In order to structure an appropriate backup system, some questions must be asked including but not limited to:

- What are the backup objectives?
- What kind of data is being used inside the organization?
- How much size is needed for the backups?
- How often should the system take backup?
- Where system backups will be stored? Is the backup separated from the network and in a different product environment or offline?
- What should be the restoration and recovery plan?
- What are the compromised paths for an attacker to gain access to the backup?
- What are the privilege levels for this backup? (i.e., who can write/access this backup) Can attackers access those accounts in some way? How can one protect those accounts from compromise?

NodeBB currently has no built-in or plug-in features for backups. The system runs on either Mongo or Redis databases. Hence, it is vital to take a backup of the database. The most important aspect of the NodeBB forum Software is to store the changes made in a safe and remote environment. This will prevent data losses from unexpected system crashes and mitigate ransomware data compromises. Since NodeBB is an open-source software project under development, it is highly vulnerable to external factors.

For this part of the project, the main objective was to create a straightforward automatic backup system to minimize data loss. In general, the script runs every minute. Connects via ssh to dump the mongo database to the AWS server. The script checks how many backups exist in the folder. If the number of backups is more than five, it deletes the oldest backup file in the folder. For this task, a cloud system is used because, in the case of local file encryption, the backup may be encrypted as well. To prevent this situation, it is safe to utilize cloud or remote server services.

I.3. Firewall

Eventually, the forum will be running on a dedicated web server. It can be integrated in a small network for a company for example, or directly open to the internet for large scale deployment. However, making the application available to any user comes with some risks. Anyone can send request to the server, some of which may differ from what is usually expected. For instance, an attacker could try to read, steal or even delete information from the server. With carefully crafted requests, he could analyze the application, exploit software vulnerabilities, find out where the traffic is most important, etc. Even inside a sub-network, we may not want all users to access the same files or perform the same actions, so we must define rules. To prevent or at least mitigate these kinds of risks, we decided to build a firewall for our application.

It is common practice to block unwanted requests and set filters to protect our server. By blocking unnecessary ports and forbidding some request, we can greatly limit the actions an attacker can perform, while making sure that legitimate users can still use the forum safely. For our firewall, we went with a “Drop by default” policy, which means that we start by forbidding all communication between our server and the outside. Then, on case-by-case basis, we define all the rules that are required for the application to run properly, and only these rules. That way, we restrict the traffic to a bare minimum and prevent most malicious behaviour.

For this project, we chose a very simple configuration in which we only have the NodeBB server and the internet, the firewall being directly implemented on the NodeBB server (see Figure in firewall documentation). We did this because it was more convenient for testing purposes, as we couldn’t define as many machines as we would have liked. In practice, it would be better to use two different computers, one for running the forum, and another one for the firewall. The commands we need to enter for configuring the firewall will not be the same in the two cases, but the logic for defining the rules remains fundamentally the same.

For this implementation, we hosted NodeBB on a linux server and used ‘iptables’ to define the firewall. First, we set the default policy DROP for all chains. We chose DROP instead of reject to give as little information as possible to a potential attacker, and to reduce the load on the system since we don’t need to craft a response for a dropped packet. After that, we must allow http connection from internet users. This uses the TCP protocol, and we know that we can block all ports except 4567 on which the application is accessible. We must define this rule for both the INPUT (so that users can send request) and for the OUTPUT chain (so that the server can reply). In this case, we can also take an extra precaution by only allowing replies to already established connections. Finally, since NodeBB needs to communicate with MongoDB, which is also hosted on the same machine, we also authorize all traffic to and from the loopback interface. Our instance on mongod runs on port 27017, so we open only this port. For more details on configuration, see the corresponding script in the firewall documentation folder.

The main difficulty was to effectively determine which rules were necessary for the application to run properly. Once we defined the default DROP policy for all chain, then of course nothing would work, and we had to rely on trial and error to find the rights command to enter. Besides, we were new to firewalls and client-server communication, and thus didn’t have the experience or intuition to find the appropriate rules. To solve this, we looked up tutorials on the internet. We can find many that explain the usual firewall configuration for web applications (this HYPERLINK "https://bencane.com/2012/09/17/iptables-linux-firewall-rules-for-a-basic-web-server/" was particularly useful). All we had to do was to tailor this to our own needs. We started with quite broad and lenient rules, that would allow the forum to run and connect, and then by looking at which packet were accepted or not, we could determine which rules matched and were thus necessary. The other could simply be removed without causing any issue. This is how we defined the minimum set of rules for NodeBB. was particularly useful).

I.4. DoS with sockets

Threats

When there is some type of service provided and available on the internet, somebody, a malicious actor, could put in place a Denial of Service attach to make that service not usable anymore. By that, in this scenario both the provider of the service (who is profiting by it) and the user of the service (who is getting a problem solved by it) are the end target by the attacker. The impact easily escalates when providers are big corporations or governments that serve users millions of people or entire nations. DoS with socket attack consists in generating an amount of socket events such that the receiver is overwhelmed and caught unprepared causing a server or software to brake and inevitably stop.

Chosen countermeasure & reasoning

The chose countermeasure for the aforementioned problem consists in controlling and limiting the usage of the socket functionality for the clients that wants to communicate with the server. This limitation can be done at numerous levels and combination of them:

- CORS: origins, methods, etc
- IP
- Bandwidth

The first limits the actual functionality of the socket communication, for instance, by imposing that only POST and GET can be used, PUT can then not be used. This countermeasure can especially be put in place by the owner of the service since only them know what is strictly necessary for the service to function.

The second consist of limiting the connections/events associated to a specific IP.

The last consist of limiting a specific socket connection by its activity.

The specific chosen countermeasure that we decided to apply is a combination of the last two, which provides both close control (IP) on the potential attacker and manoeuvrability by its activity.

Implementation

For the implementation, we made use of a specific NPM package that has already implemented the logic for keeping count of a specific identifier. The trick here is the right positioning the function calls and the right configuration. The name of this library is called node-rate-limiter-flexible and is properly supported by node, the environment where NodeBB resides. We import the *RateLimiterMemory* class from the library and instantiate an object by setting two properties: points, duration. The firsts are the number of events that can occur in a set timeframe (duration in seconds). We decided to go for 5 points and 1 second. Then, we chose the right spot on the code where every message needs to go through (*onMessage* function) and placed a call to the function of the previously instantiated object: *consume(param).* The param here is the ip of the client associated with the socket event. When this function gets called, it registers one point used by a specific ip. If the number of points exceeds the configured threshold, an exception is thrown such that the event is not handled.

II. Member contribution

II.1. Ada

For this project, I worked on implementing a countermeasure against spam. Before deciding which problem to focus on, I had several issues when trying to configure MongoDB on my machine. For some reason, it did not work properly so I had to remove it and instead use a MongoDB cloud instance that another member of our group had set up. After some time, I was finally able to run NodeBB forum on my localhost.

I inspected the forum and I noticed that when switching "Require new users to specify an email address" toggle on, users need to enter an email address to register. However, it does not check at all if the email is real or that it belongs to the user, so a user can enter any email and still be allowed to become part of the forum. My idea was to have nodebb send an email to the user's email address with a code. Only after inputing this code in the next registration step, can the user create his/her new account. After spending some days on trying to figure out how to implement this solution and after consulting with our TA supervisor, I decided to implement a countermeasure against spam instead.

There was already an anti-spam plugin installed with NodeBB so I enabled it and read about its dependencies. Spam Be Gone plugin included several anti-spam services, some of which had the same functionality. Project Honeypot, Google reCAPTCHA and hCAPTCHA could all be used to check every user registration. Akismet could be used to check every user post for spam and StopForumSpam, to report a spammer. After researching all of them, I decided to use Google reCPATCHA and Akismet for this project work. I chose Google reCAPTCHA out of all the others, because not only does it check every user registration, it also checks every user login. On the other hand, I decided not to configure StopForumSpam for reporting spammers, since I was creating all the users and posting through them, so the IP would be the same for all of them.

After reading the documentation page of Google reCAPTCHA, I learned that reCAPTCHA offers a test key pair when using localhost for testing purposes. However, when using those keys all verification requests will pass. Instead, I wanted to use real keys so that each request is properly validated. After consulting StackOverflow (our best friend) I learned that I had to register both localhost and 127.0.0.1 for it to work. First, I tried using reCAPTCHA v3, but an error “Invalid type” would appear each time, so I used reCAPTCHA v2 Checkbox, which was the appropriate type. Finally, reCAPTCHA seemed to be working fine.

Next, I went through similar steps to configure Akismet, to check every user post for spam. I went through the FAQ page on Akismet website and created a personal account. Luckily, I did not face any particular issues when configuring Akismet.

Finally, I wanted to trigger both anti-spam services by trying to spam the forum. I registered new users and went to the admin page on Google reCAPTCHA to look at the received requests. All requests were received and verified and there were no suspicious ones (which is expected). Then, I tried to publish spam content by using key words and broken links. Some of them were detected by Akismet and it did not allow me to publish them by displaying that the “content was flagged as spam”. I went to the admin page on Akismet and these requests were correctly included in the chart. However, spam detection was not always 100% correct, as there were times I could post irrelevant broken links, but that is also expected because anti- spam databases are constantly expanding.

II.2. Damla

My contribution to the project was on ransomware, data loss topic. First, I instantiated an EC2 server on AWS with the contribution of my friend, who is the account owner. Then I connected to the server via SSH, "54.82.198.169". However, I encountered some problems such as user privilege. Therefore, I needed to create a new key pair for myself and used the root user, *root@dasak*, by using "*sudo su*" instead of my local user, *student@dasak*". Then by using our ubuntu DASAK VM, I set up MongoDB and NodeBB. Since NodeBB has no built-in or plug-in features for backups, I wrote a *.sh* script that took a backup of the server and config files automatically. For this purpose, I benefitted from **mongodump** tool which creates a binary export of a database's contents. These files are stored in */usr/bin/mongodump*. Backup directory is */home/ubuntu/dump*. The file names are generated as DateTime which is Year-Month-Day-Hour-Minute. The number of available backup files is checked via "*wc -l*". If the data is greater than 5, the least recent one is deleted, *rm –rf "$ (ls -t | tail -1)"*.

The basic algorithm structure works as shown below:

cron: with period T

run script

script:

ssh <remote-server>

mongodump

scp <remote-server>

fetch dump

ssh <remote-server>

remove dump

if #dumps > 5 then

delete oldest dump

end

For setting the automatic backup, I utilized the **cron** scheduler. Commonly, in organizations, it is a good practice to set a backup daily after midnight. For the sake of the demonstration, I set a backup schedule every minute. To edit the cron, I used ***crontab - e***. Once I gained access to the document, I added the following line, ********/home/student/Desktop/script/script.sh***, which sets the backup schedule to every minute.

One of the problems I encountered while configuring MongoDB was Mongo Authorization Problems / Can't Access or Backup Database. I tried to connect to MongoDB with admin and the password I was getting an error:

~/nodebb$ mongo -u "admin" -p "123456" --authenticationDatabase=admin.

Error: couldn't connect to server 127.0.0.1:27017, connection attempt failed: SocketException: Error connecting to 127.0.0.1:27017 :: caused by :: Connection refused :

The solution I found on the Stackoverflow website was to Disable Authorization - comment out the security section at /etc/mongod.conf.

II.3. Hugo

I worked on the firewall configuration for protecting the NodeBB application. I installed the system on the Dasak VM because I was more familiar with the Ubuntu commands and iptables. Windows firewalls are more complex and do not give as much freedom. Defining the appropriate rules wasn't so long, about 6-8 hours in total.

Most of that time was spent reading the iptables documentation (labF greatly helped me) and looking up tutorials for web server firewalls. At first, I didn't know which traffic to filter or not, shat I first did was define common rules for web applications. For example, on most of the tutorials I've read, it is recommended to allow SSH, HTTP, FTP and DNS traffic. Accepting loopback traffic to make the communication with the database possible was also something that was encouraged.

Some of these rules were not necessary, but at least with these the NodeBB server could run and accept connections without any issues. I then tested the forum by logging in, registering new users, writing posts or uploading files. In a nutshell, I did what a normal user would usually do. After that, I looked at all the rules in my firewall and noticed that only a few of them had matched any packet. It meant that the configuration was too lenient and that I could remove some useless filters.

I kept proceeding like that until I reached the smallest set of rules that would still allow the application to work properly. I would first make sure that I could execute NodeBB in a given configuration, and then I would change the rules one at a time to see if it was still working. That's how I pinpointed which rules were essential and how precise I could make them.

A good example would be the rule to enable loopback traffic. Before I wrote this rule, NodeBB was able to start, but would crash as soon as it received a request. I understood after a while that it was because it was unable to communicate with MongoDB, which was also inside the VM. I added the corresponding rule, and it worked fine. After that, after reading on the documentation that MongoDB was accessible through port 27017, I restricted the firewall rule a little to only allow that port. With slight adjustments at every step, I was still able to launch the application, which meant that I had successfully pinpointed which traffic was required.

Once I had found all the rules, I just wrote a small sh script with all the necessary iptables commands as well as the guide in this report for configuring VirtualBox and testing the configuration on the Dasak VM.

I also spent some time on other problems. At first, I wanted to implement a solution for user authentication, and I found a Two-Factor Authentication plugin for NodeBB. It worked great, but it was easy to install, and they were few configuration steps as the plugin was already quite complete without having to modify it in any way. That's why I didn't deem that it was worth mentioning, as it wouldn't have represented the expected amount of work for this project.

II.4. Riccardo

My main contribution has been related to the implementation of the socket DoS countermeasure. Here is the list of activities that has been carried out:

1. Project setup/exploration: first look into the repository/codebase and configuration of the local environment. For MongoDB I have opted to use the free cloud instance they provide so I just entered those details at configuration time. I shared this with the rest of the team.
2. Identified socket implementation: Since our countermeasure was going to be implemented on top of the existing implementation, I took some time to explore how the socket are implemented on NodeBB.
3. Download and execution of GoldenEye (https://github.com/jseidl/GoldenEye): Even though is was not required, I decided to explore the actual presence of a vulnerability in the current socket implementation. I performed two initial DoS attack tests with the following params :
   - Test 1: base params (10 workers, 1000 sockets, get method), img 1
   - Test 2: tweaked params (10 workers, 5000 sockets, random method), img 2

   After some time, we get error 503 from server img3
4. Analise current socket implementation (file src/socket.io/index.js) : There is a event flooding control that consists in a middleware that is called with method "isFlooding" configured to allow maximum 100 events in a timeframe of 10000 milliseconds (10 sec).
5. Search for npm packages that are flexible (allow customization) -> https://www.npmjs.com/package/rate-limiter-flexible
6. As described before, we want to apply a more radical countermeasure, limiting by rate by IP using "rate-limiter-flexible" package. It works by means of maximum points per second, for every event the client consumes one point. We set 5 points per seconds, which is also the standard configuration that can be adapted based on the overhead of events out application has implemented (we don't want to block clients for a legit number of events consumed).
7. Performing multiple tests of the implemented countermeasure: I don't receive the expected log of connection blocked but I don't see the 503 error anymore. I also tried changing the implementation to different parts of the code.

III. Summary of the organization

For labor division, we took the approach we thought was simplest. We knew we had to describe and fix 4 problems, so each member of the group had to select one problem to work on. At the very beginning of the project, we made some online calls to talk about which problems everyone would like to choose. This way, we were sure that two people wouldn't work inadvertently on the same issue. Of course, our choices back then were not definitive, and after we had the opportunity to test the application and understand it in more depths, some of use changed their minds and went for other problems, while notifying the others of course.

For communication, we essentially used WhatsApp for group discussion, GitHub for sharing code, and OneDrive to work together on documents such as the report. In the end, we didn't spend a lot of time together. Everyone managed to come up with a solution on their own, and we only exchanged when we had questions or some problems with the system. Of course, we would also discuss important topics, such as the meeting with the TA or how to write the report.

At first, we were planning to follow a time allocation similar to that of Canvas, but we faced some issues and things didn't go exactly as planned. First, it was hard for some of us to actually find time to work on the project because the other courses were very demanding, so some people only managed to find a solution near the deadline.

The configuration of NodeBB was also somewhat complicated, which made it harder to just focus on solving vulnerabilities. For example, one of us had real troubles making MongoDB work on their personal machine and had to use the cloud instance of another member until they finally fixed the issue.

Moreover, we had a hard time understanding what was expected of us and what sort of problem we should work on. At some point, some of us thought that it was necessary to find real, exploitable weaknesses in the code, which was not guaranteed, so they wasted a lot of time on this before switching to a more realistic countermeasure.

Finally, we were not very experienced with some cybersecurity concepts or even the programming language. Some members of the group were even completely new to it, and we thus took an extra time to understand every technical aspect and how they worked. Writing scripts for backup, configuring firewalls or configuring plugins were not things we had done before, and though it was a great learning experience, it was also very challenging and required a bit more time than initially expected. Of course, the lectures and labs we had during the period helped us a lot.

If we were to compare to the time allocation on Canvas, we'd say that for installing the system and getting familiar with it, we took a bit longer than expected (maybe 2 days instead of one), but we somehow made up for it by finding countermeasures more easily (2 days instead of 3). Indeed, the configuration required for some countermeasures wasn't too overwhelming and we were able to understand it pretty fast. For writing the report, everyone talked about the section they had worked on and it went smoothly, letting us finish everything up before the deadline, as planned.

In conclusion, even though everything didn't go as we expected, every single one of us managed to describe and fix a vulnerability in time, and we are overall quite satisfied with our results. As already mentioned, some parts were very hard, frustrating and time-consuming, such as going through the documentation or understanding the source code, but we still managed to overcome this and we believe that we have learned a lot during this project and that it will greatly help us in the coming cybersecurity courses.