Finding Security Vulnerabilities in a NodeJS Application   
Assignment 1  
Group 10   
Victor Kubrak  
Francesco Losi  
Harika Naidu  
Yuliya Yatsenko  
BTN710

Table of Contents

[**Introduction** 3](#_Toc25184327)

[The Exploit 4](#_Toc25184328)

[The Attack 5](#_Toc25184329)

[Brute Forcing the Login Page 5](#_Toc25184330)

[Anyone Can Create an Admin Account 6](#_Toc25184331)

[System does not require strong passwords 7](#_Toc25184332)

[Database Denial of Service 8](#_Toc25184333)

[No encryption of data 8](#_Toc25184334)

[Security Policy 10](#_Toc25184335)

[Conclusion 11](#_Toc25184336)

[References 12](#_Toc25184337)

**Introduction**

This report provides information about the security deficiencies found in a student developed NodeJS application. The main goal is to determine various exploits in code that an attacker could use in the application. The application in question is a capstone project from BTI-325. It is a simple web app that holds a database of users and their information. Information such as address, phone number, the department they belong to, social security number, name and employee number. There is some security that is implemented such as hashing the password before being saved to the database, cookies with login information and the need to be logged in to see or modify any information. But apart from that there are numerous other security vulnerabilities we have identified in the app, and this report will elaborate more on those vulnerabilities and how an attacker could attack the web app. Finally, using the various security fundamentals we’ve learned in the BTN710, we will improve the security of the application by applying them to remedy said vulnerabilities.

# The Exploit

While searching for security vulnerabilities within the web application there were many things that could be exploited, one of the first things that became apparent was there is nothing that would protect the login page being brute forced. The system does well in that you need to create an account in order to see any information that is stored in the site but there is nothing that would stop a user from brute forcing the login screen.Also, Users entered into the database are case sensitive so if a hacker wants to be logged in they must get the whole username right even the case sensitive portions. When a user gets their login wrong the system tells them whether it was their username or password that was incorrectly entered and based off this information an attacker can brute force a login until they are allowed access. Upon logging in the database saves the date and time of the log in and saves it to the database.

The next apparent vulnerability was presented at the registration page of the site, any user who does not have credentials can still get into the system by simply creating an account. This poses an obvious risk to the whole system since upon logging in you can remove a user’s information from the database. Since this issue exists solely because of the registration page, this page must be changed to ensure that any new accounts created do not have the ability to delete any information.

The database is also susceptible to a Distributed Denial of Service (DDoS) attack. This type of attack would render the web app useless to both legitimate users and administrators since this attack would affect the responsiveness of the app. This is an important attack to tackle because it is one that is constantly evolving due to Botnets that allow for this attack to be implemented on a large scale (Hulme). There is currently no system in place on the web app that could stop or lower the effectiveness of this attack.

Although the system does hash all passwords before storing them, the passwords that are hashed have no requirements. This makes passwords that are hashed easier to decrypt and leaves any existing users, including admins, susceptible to an attack as simple as just guessing using the most common passwords. Most users will create simple passwords out of laziness, so the system must enforce users to create more complex passwords. A survey conducted in 2015 found that weak authentication is the leading cause of data breaches making up for 76% of compromised records (Brecht).

The last vulnerability that was detected was data that is sent between the app and the database where tables of employees and departments are stored is not encrypted. This means that if an attacker could find out which ports the information is being sent out of, they would be able to read it in plain text.   

# The Attack

In this section of the report we will explain the steps of each exploit that we found in the program along with how they work and how to protect against these attacks.

### Brute Forcing the Login Page

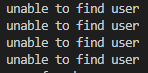
How the code currently implements this page is when the user hits enter the app checks to make sure that that user exists in the database, when the user is checked in the database there is absolutely no encryption.  
  
A screenshot of a cell phone

Description automatically generated

As you can see in this snippet of code, if the username doesn’t exist in the database the function will return a string back to the HTML page containing “unable to find user:” along with the incorrect user that they entered. If the username is found in the database, then the following code is executed.   
  
                 A screen shot of a smart phone

Description automatically generated

Now that the username is verified the app must verify the password that the user entered.  Every password saved into the database is hashed using Bcrypt which is a built-in library in NodeJS that uses a salt and a hash function to encrypt passwords saved into the database. Bcrypt does allow the system to slow down brute forcing by increasing the number of times the input is salted, but it does not completely nullify the attack. When someone attempts to log in, the password entered is salted and hashed and is then compared to the stored salted and hashed password, if they match the user is authenticated if not string is returned to the page saying “Error verifying user” along with the user name that they got the password wrong for.  If the app is run locally then there is a signature left behind in the console in the form of error messages as shown below.



The solution that has been decided for this vulnerability is to limit the amount of login attempts that the hacker has to get the right login. This will be done by timing out the attacker after a certain amount of login attempts have occurred.  The following update doesn’t allow users to make more than 5 login attempts from the same IP in 1 hour and more than 2 registration attempts in 30 days.

A screenshot of a cell phone

Description automatically generated



A screenshot of a cell phone

Description automatically generated

### Anyone Can Create an Admin Account

This vulnerability does not involve a weakness or exploit in the code itself, but in how the registration page handles a new users access level. When a new user is created the system does not check for or ask if the user is an administrator it simply creates a new user, adds them to the database and displays all the employee/department records currently in the database. Next to all of the employees and departments in the database the user is given the option to remove the database entry. To fix this issue changes need to be made to the database and the registration form to ensure only authorized users will receive a higher level of access. The best course of action to ensure only authorized users can have higher level access would be to make it so that only admin users can create admin accounts internally. This would ensure any admins created are verified by the first admin account, since that first admin accounts needs to create and subsequent admins. Upon logging in, any user who is not an admin would simply be shown the information in the database but without the ability to remove employees and departments, but they can still create them. The database would also have to be changed by adding a column that would confirm whether or not a user has administrator access since the pages displayed would be different upon logging in depending on this column. When admins log in they can see, for example, a remove button next to every entry. Here is some of the code before and after update:



A close up of a logo

Description automatically generated

### System does not require strong passwords

Everyone has heard that creating strong passwords is crucial for security. It makes brute forcing impossible with current technologies. The bigger a password is and the more different kinds of symbols it uses, the harder it is to be brute forced. So, the solution that fixes this vulnerability is to set a few rules for passwords. For example:

* Passwords must be at least 12 characters long
* Passwords must contain at least 1 lowercase character
* Passwords must contain at least 1 uppercase character
* Passwords must contain at least 1 number
* Passwords must contain at least 1 special symbol, such as “!@#$%^&\*()\_+”

Also, it makes sense to advise users that using common words and personal information that can be discovered by someone is a bad idea. It would ease the brute forcing: dictionary attack in combination with social engineering could hack your password. Here is the code that we added:

A screenshot of a cell phone

Description automatically generated

### Database Denial of Service

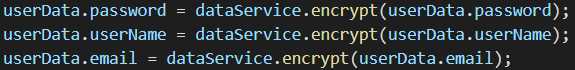
If someone hacks client code, he/she will be able to send malicious requests directly to the database. Actually, a huge number of requests can be sent to a database even without hacking the client code. So, the code should be separated into front-end and back-end. Separating code itself is not a protection against Database DoS attack and also requires implementing some logic, such as dropping packets or setting low timeouts. One of the solutions involves creating a firewall that will hide the database.

### No encryption of data

The project is vulnerable to network scanning, so we need to take some actions to protect application against network scanning.

The first option is to use Diffie-Hellman protocol to create a cryptographic key that will be then used for symmetric encryption of client-server communication. This solution would be good if we split application into front-end and back-end.

Another option (that we implemented) is to encrypt data before saving it to a database and to decrypt it after fetching from database. Thus, data received from network scanning, can’t be interpreted without a key. Here some of the changes made to implement encryption:





A screen shot of a social media post

Description automatically generated

# Security Policy

Since our exploits cover a wide variety of security policies there will be quite a few that can be implemented to ensure these attacks will no longer affect the system.

**All users will be authorized and given access level accordingly**. This security policy will ensure the integrity of the data in the system and will ensure that only authorized users can remove entries from within the dataset.

**Authorized users will be determined by an admin or someone with administrative access.** Implementing this policy will ensure all users who can remove data from the database have been authorized by the admin.

**User cannot create a user account unless all password requirements are met.** Password length and complexity rules ensure that the brute force attack cannot be performed. User login credentials will be complex enough in order to not be guessed or dictionary attacked.

**Limit the number of attempts that could be performed by user to login.** Locking out the attacker after a certain amount of login attempts will prevent brute-force attack at the login page.

**Data is encrypted before saving to database and decrypted after fetching.** This policy protects application against network scanning. The data received from network scanning now cannot be interpreted without key.

# Conclusion

Using the techniques and security policy knowledge gained from the BTN710 course we were able to identify most of the vulnerabilities and address them. Before this course it was difficult to tell if what we are working on is secure or we didn’t even think about security. But now that we know the numerous ways an attacker can exploit a web page, network or security we are more aware of how to implement the necessary security policies to minimize the consequences of an attack.

**References**

Hulme V.George. “DDoS explained: How distributed denial of service attacks are evolving.” *CSO Online*. <https://www.csoonline.com/article/3222095/ddos-explained-how-denial-of-service-attacks-are-evolving.html>. Accessed 6 November 2019.

Breacht, Daniel.”Password Security:” Complexity vs. Length.” *INFOSEC Institute.* <https://resources.infosecinstitute.com/password-security-complexity-vs-length/>. Accessed November 10 2019.

Porter, Kim. “What is encryption and how does it protect your data.” *Norton Security Center.* <https://us.norton.com/internetsecurity-privacy-what-is-encryption.html>. Accessed November 12 2019.

Petters, Jeff. “What is a Brute Force Attack?” *Varonis Security Blog.* <https://www.varonis.com/blog/brute-force-attack/>. Accessed November 20 2019.