

NETWORK SECURITY

PASSWORD AUTHENTICATION

Project Report

Group 37:

- IIT2019008 - Chetan Garg
- IIT2019034 - Rajat Mehra
- IIT2019037 - Abhishek Bhaware
- IIT2019038 - Ritik Mehra
- IIT2019040 - Pratham S. Punj
- IIT2019059 - Mandar Wanjare

ABSTRACT

Passwords are the most common method of authentication. The use of passwords is still the most extended mechanism for online authentication. This is understandable, given that the only requirement for everyone is to remember their username and password, instead of the inconvenience of having to carry a digital certificate, specialized hardware or software, etc. We see endless cases of password theft daily and these are an indication that much care needs to be taken when implementing one mechanism or another. Especially keeping in mind that users can't be expected to follow adequate policies for managing their passwords.

In this project, we aim to demonstrate password based authentication using three approaches: Simple Approach, With Hashing, With Salt and Hashing.

1. INTRODUCTION

Almost all server software permits client authentication by means of a name and password. For example, a server might require a user to type a name and password before granting access to the server. The server maintains a list of names and passwords; if a particular name is on the list, and if the user types the correct password, the server grants access. There are these two types of Authentication methods

1. Certificate-based authentication
2. Password-based authentication

In this report, we will be talking about password-based authentication only.

Now, most users are already familiar with passwords. In fact, passwords have been the tried-and-true method for user authentication since the beginning of the internet. You probably have quite a few passwords yourself!

A password-based user authentication process generally looks like this:

- When you land on the page, you'll be asked to enter your username and password.
- Your credentials are sent to the website's server and compared with the information they have on file.
- When a match is found, you'll be able to enter your account.

Passwords are often used to secure personal accounts like social media profiles, online banking and eCommerce sites, and other online resources. However, passwords are

not as secure an option as many users think they are. And a lot of damage can be done if a hacker is able to gain access to one of these accounts!

There are various ways to implement this authentication process.

The naive solution is to just store a user's username/email and password directly in the database, and then check the submitted password against the stored one. While this is essentially what should happen during authentication, this is actually more prone to attack such as keylogger.

The better solution is to store the password with hash, i.e. replace the actual password with its hash value which is more complicated and complex than the original password. In this way, keylogging won't help. But, a hacker could have access to your password with brute force attacks such as dictionary attacks where the attacker can attempt to break the encryption or gain access by spraying a library of terms or other values. Poor password hygiene such as superficially updating passwords with successive numbers, symbols, or letters makes dictionary attacks easier to execute.

A more optimal and secure approach is to use salt hashing in which a string, which we call salt, is attached to the password and is hashed, making it more secure.

Throughout this article, we'll be explaining how you should be doing authentication with all three methods mentioned above and the reasons behind it.

2. LITERATURE SURVEY

Password authentication is a method that keeps unauthorized users from accessing sensitive information. For example, User A only has access to relevant information and cannot see the sensitive information of User B. Cybercriminals can gain access to a system and steal information when user authentication is not secure. The data breaches companies like Adobe, Equifax, and Yahoo faced are examples of what happens when organizations fail to secure their user authentication. Hackers gained access to Yahoo user accounts to steal contacts, calendars, and private emails between 2012 and 2016. The Equifax Data Breach in 2017 exposed the credit card data of more than 147 million consumers. Without a secure authentication process, any organization could be at risk.

3. SETUP AND REQUIREMENTS

REQUIREMENTS:

To test all 3 approaches of password authentication one require the following:

1. You should have node.js installed on your PC. [\(LINK\)](#)

2. You should have SQL workbench installed on your PC. [\(LINK\)](#)
3. You also need to install MYSQL

SETUP in workbench:

1. Open workbench
2. Open a new script
3. Now you have to create 3 databases named passwordauthsimple, passwordauthhash and passwordauthsalted by using the command `Create database <Database name>`
4. In this you have to create table having same name for that you need to write command

```
create table user(  
  Username varchar(255),  
  Password varchar(255)  
);
```

5. Now you are good to go to the extraction part of the zip folder.

SETUP:

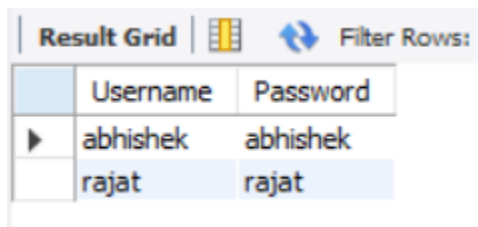
1. Extract one of the zip file of password authentication
2. Open VScode inside the extracted folder.
3. Open terminal in VScode by clicking the terminal button on top and make sure that you are in the current directory in the terminal.
4. Then you need to install the dependencies by typing the command `npm install`
5. This will install all your dependencies that are needed.
6. Then you need to open mysql folder which is just inside the folder that you extracted just now
7. You can see the mysql.js file open that in any editor
8. Change the password of yours workbench password **(Important)**
9. After this you need to start the server by typing command `nodemon server.js`
10. Now the server is running you can see on the terminal showing the verdict that the server is running on port 3001.
11. Now Open the browser of your choice.
12. In the search bar type the address `localhost:3001/`
13. Now you can see the page saying Login or create account.

4. IMPLEMENTATION

4.1. Approach 1:

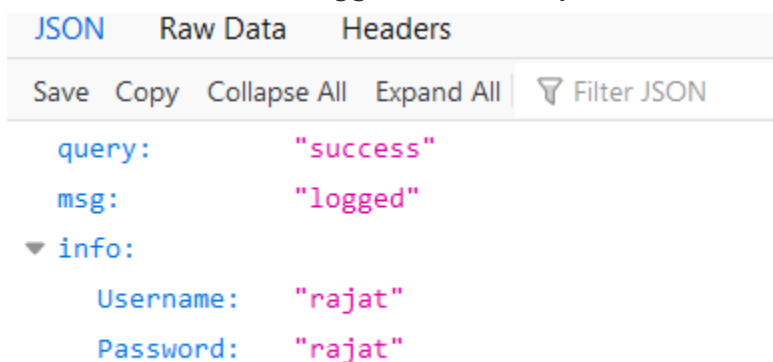
This approach stores the password in the plain format. This is kind of key value pair format here username is treated as key and password is treated as value now if a user came in and sends a username and password then in password file or database in our case checks that if that key exist in your database or not if not exists then verdict comes as there is no user and you need to create account or in layman language you basically need to create a key value pair. The database used is MYSQL if everything goes right then it shows the verdict logged in.

- Image showing the key value pair.



	Username	Password
▶	abhishek	abhishek
	rajat	rajat

- Verdict when logged in correctly



JSON	Raw Data	Headers
Save	Copy	Collapse All
Expand All	Filter JSON	
query:	"success"	
msg:	"logged"	
▼ info:		
Username:	"rajat"	
Password:	"rajat"	

- Verdict when Password is incorrect

JSON	Raw Data	Headers
Save	Copy	Collapse All
	Expand All	Filter JSON
<pre>query: "success" msg: "password is incorrect"</pre>		

- Verdict when username is incorrect

JSON	Raw Data	Headers
Save	Copy	Collapse All
	Expand All	Filter JSON
<pre>query: "success" msg: "Please create account"</pre>		

4.2. Approach 2:

This Approach is also similar to the 1st one but instead of storing the password directly we need to store the password by hashing the password first then for checking we also need to hash the given password first then we will check the key value pair and rest of all things are the same as previous.

Hashing **performs a one-way transformation on a password, turning the password into another String**, called the hashed password. ... "One-way" means that it is practically impossible to go the other way - to turn the hashed password back into the original password.

Hashing **turns your password (or any other piece of data) into a short string of letters and/or numbers using an encryption algorithm**. If a website is hacked, the hackers don't get access to your password. Instead, they just get access to the encrypted "hash" created by your password.

Hashing a password is **good because it is quick and it is easy to store**. Instead of storing the user's password as plain text, which is open for anyone to read, it is stored as a hash which is impossible for a human to read.

Hashed passwords cannot be retrieved in general (this depends on the hashing function, secure hashes cannot be retrieved). If they have the same hash on two sites, they could have the same password, this depends on the hash salt used by the sites, what method etc.

For hashing we are using crypto module of javascript.

Crypto is **a module in Node.js** which deals with an algorithm that performs data encryption and decryption. This is used for security purposes like user authentication where storing the password in Database in the encrypted form.

Hashing, or encrypting is slow. That's one reason why it is secure, because you need a lot of computing power to be able to break it.

Hashing is one way only. That means if you want to retrieve the first and last name of your user, you won't be able to. Another example is if you hash balances, you won't be able to see who owns the most money, or who owes what.

- Image of key value pair

Result Grid		Filter Rows:	Export:	Wrap Cell Content:
	Username	Password		
▶	rajat	438b96eda2c35cb34f65dd1f7dca3ff4		
	rajata	6d3c228440b65a8079ea660ada61a745		

- Verdict when logged in correctly

JSON		Raw Data	Headers
Save	Copy	Collapse All	Expand All
query:		"success"	
msg:		"logged"	
▼ info:			
Username:		"rajat"	
Password:		"rajat"	

- Verdict when Password is incorrect

JSON		Raw Data	Headers
Save	Copy	Collapse All	Expand All
query:		"success"	
msg:		"password is incorrect"	

- Verdict when username is incorrect

JSON	Raw Data	Headers
Save	Copy	Collapse All Expand All Filter JSON
query:	"success"	
msg:	"Please create account"	

- Verdict when logged in correctly

JSON	Raw Data	Headers
Save	Copy	Collapse All Expand All Filter JSON
query:	"success"	
msg:	"logged"	
▼ info:		
Username:	"rajat"	
Password:	"rajat"	

- Verdict when Password is incorrect

JSON	Raw Data	Headers
Save	Copy	Collapse All Expand All Filter JSON
query:	"success"	
msg:	"password is incorrect"	

- Verdict when username is incorrect

JSON	Raw Data	Headers
Save	Copy	Collapse All Expand All Filter JSON
query:	"success"	
msg:	"Please create account"	

4.3. Approach 3:

This approach is also similar to the previous one but instead of storing by simple hashing we need to do the salting process first.

Password Salting is **a technique used to help protect passwords stored in a database from being reverse-engineered** by hackers who might breach the environment. Password salting involves adding a string of between 32 or more characters to a password and then hashing it. "Hello", for example, will always equal the same combination of letters and numbers, and therefore can be guessed through brute force. One way of protecting against this is by adding salt or using salted passwords. The first step is to make your Salt as unique as possible. Make it as different as you can, using characters which one would never commonly pick. For example, if you use ten different salts, you are increasing the security of the hashed password by a factor of ten.

Furthermore, when the salted password is stored separately, using rainbow tables, it makes it difficult for the attacker to determine the password. The best method to [ensure privacy protection](#) is to use a unique salt each time the same user generates or changes their password.

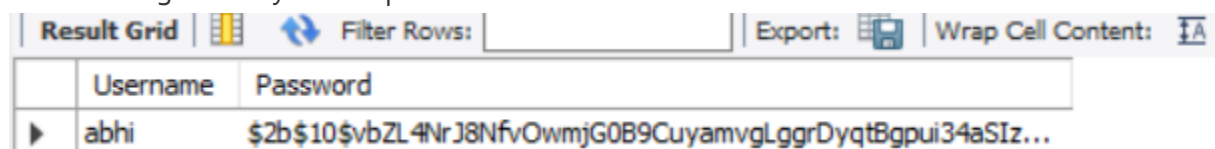
For salting and hashing we are using bcrypt module of javascript.

bcrypt is a password-hashing function designed by Niels Provos and David Mazières, based on the Blowfish cipher and presented at USENIX in 1999.

The largest benefit of bcrypt is that, over time, **the iteration count can be increased to make it slower allowing bcrypt to scale with computing power**. We can diminish any benefits attackers may get from faster hardware by increasing the number of iterations to make bcrypt slower.

bcrypt has a maximum input length of **72 bytes** for most implementations. To protect against this issue, a maximum password length of 72 bytes (or less if the implementation in use has smaller limits) should be enforced when using bcrypt.

- Image of key value pair



Username	Password
abhi	\$2b\$10\$vbZL4NrJ8NfvOwmjG0B9CuyamvgLggrDyqtBgpuI34aSIz...

5. CONCLUSION

We have demonstrated a simple DDos attack using ICMP flooding with a python program. We have also demonstrated the detection (by monitoring network resource usage) and prevention (with help of firewall rule filtering icmp packets).

We have demonstrated a simple password authentication by using javascript and some modules relate to it. We have also discussed the pros and cons of the method and how can you even secure the current method.

6. REFERENCES

- I. <https://www.netscout.com/what-is-ddos/icmp-flood>
- II. <https://console.cloud.google.com/>
- III. <https://www.sciencedirect.com/science/article/abs/pii/S1389128603004250>