Secure Website Authentication using Hashing Algorithms

Shankarlal Sharma
Department of Information Technology
AP Shah Institute of Technology
Thane, Maharashtra, India
shankarjsharma@gmail.com

Dharmraj Yadav
Department of Information Technology
AP Shah Institute of Technology
Thane, Maharashtra, India
yadavdharmraj1998@gmail.com

Gaurav Babar
Department of Information Technology
AP Shah Institute of Technology
Thane, Maharashtra, India
gauravbabar15@gmail.com

Satyajeet Yadav
Department of Information Technology
AP Shah Institute of Technology
Thane, Maharashtra, India
yadavsatyajeet19@gmail.com

Prof. Apeksha Mohite
Department of Information Technology
AP Shah Institute of Technology
Thane, Maharashtra, India
atmohite@apsit.edu.in

Prof. Neha Deshmukh
Department of Information Technology
AP Shah Institute of Technology
Thane, Maharashtra, India
npdeshmukh@apsit.edu.in

Abstract **-** In this growing technology era, where most of the information sharing and processing is being done online, it is of great importance that the end-user’s security be ensured. The number of websites on the Internet is increasing day by day, and so is the need to secure them. When the users interact with the website, they must be ensured that their data is safe. This will increase the reputation of the website and build trust among the potential visitors. Also, securing a website ensures great reputation for the website owner as they align well with the Search Engine Optimizations. In this paper, we have analyzed a technique by which user authentication can be made secure by using bcrypt algorithm. We discuss the implementation and compare it with various other hashing algorithms.

Index Terms- bcrypt, nodejs, website security, authentication, hashing, rainbow table attack

I. Introduction

Website attacks and Cyber-attacks are increasing day by day and it is of utmost importance to provide security to the user as the data that is hold by the website. A secure website is an area which builds trust among the users towards the website as the users are ensured that their data does not go into wrong hands. Another advantage website security provides is that the website get good search engine rankings as more people start visiting the website. Websites that fall under the category of Banking, E-commerce, E-wallets, social media particularly needs to ensure high security of their customers. In following sections, we describe various hashing algorithms which can be used to hash the passwords of user while registration process and store the hashed passwords in database instead of storing the plain passwords directly, that is more vulnerable to attacks.

Also, the implementation of bcrypt algorithm in a nodejs application will be discussed.

II. Need For Hashing Passwords before storing into database

Passwords are an important means by which a website is prevented against cyber-attacks. So, storing the password as a plain text into the database can prove to be vulnerability and cause the user data to get hacked easily. So, it is important to store the password after applying a hash on them which is not easy to crack. The difference between hashing and encryption is that once a password is hashed, we cannot derive its original plaintext back from the hash, whereas it can be done using encryption technique. But, even after storing passwords in hashed form, the issue is that a password "abc123" will always have the same hash and if some hacker knows the password of one user then it is probable that the same user may have used same password for various other sites and it will be easy for the hacker to hack multiple accounts of that user. Because of this, in addition to hashing the password, there is also a need to add salt to the hashed password which is unique for every website. If, the password is added to the database after adding salt to it, then it will be different for different websites, and thus the encrypted form of same password will be stored in a different pattern on different websites, unless the salt provided for both websites are exactly the same. Thus, if the hacker finds out the hashed password of one website then finding the password of same user and different websites with same password will not be simple. Later on, when the user attempts to login into the system with their plaintext password, then it converted into hash and applying salt on it and is compared with the one already stored in database for that user, which was stored during user registration.
Also, while selecting a hashing algorithm for password, speed of hashing is considered to be an important factor. It is recommended that the hashing of passwords must be slower as this will demotivate the attempts of attacker, but at the same time, the hashing speed must be such that the performance of website is not reduced.

III. Possible security attacks on websites

Before moving into the ways by which a website can be secured, it is important to get ourselves acquainted with the several means by which the passwords may get attacked.
Knowing about these attacks can create the awareness among users regarding how to choose a secure password for their accounts.
Following subheadings discuss the methods:

A. Dictionary Attack

It is a type of attack in which the hacker prepares a file containing all the possible passwords like the words in dictionary and then those are applied to crack the password. As the name suggest, it is rather a straightforward attack. It is the first method of attack and more the words contained, quickly the password can be cracked. However, by choosing a random and long password can prevent this attack.

B. Brute force Attack

This attack is similar to dictionary attack, but instead of dealing with just the dictionary words, the hacker tries all the possible letter, numbers and symbol combinations that a user can use as password. But, this is one of the ineffective techniques which can result in making a huge number of unnecessary guesses and thus increasing time. Sometimes choosing longer passwords can also defeat this technique.

C. Rainbow Table Attack

These are attacks in which a hacker prepares a table containing all the possible hashes for a password which is applicable to any given hashing algorithm. It reduces much of the time of the attacker as the hashed are pre-calculated and only computation task remaining is comparing target hash with the stored hash.

But, if a salt is applied on the hashed password then it becomes rather difficult for the hacker to crack the password as compared to just hashed password.

D. Rule based Attack

In this attack, the attacker defines several rules on generating passwords based on only the rules the websites allows for creating a password. So, the attacker need not generate and use passwords that do not match websites password creation rules.

IV. Several techniques of Storing Passwords

Following the methods by which a user's password can be stored in database:

A. Plaintext:

It is a way storing passwords in database where no encryption is performed on the plain text password entered by user and is stored as it is into the database. It is never recommended to use this method if the website commands security against attackers, as it is most vulnerable to security attacks.

B. MD5:

Message Digest 5 (MD5) produces a 128-bit hash value for a given plaintext. The requirement is that no two words must have a same hash value but MD5 catastrophically fails in satisfying this constraint and so, its use is reduced. However, it can be used for the checksums.

C. SHA 1

Secure hash algorithm 1 (SHA1) is an algorithm which takes an input and converts it into a 160 bit hash value which is a hexadecimal number. A collision attack was detected when this algorithm created same hash value for two different pdf files and thus is now not much recommended for hashing passwords.

D. SHA 256

Secure hash algorithm 256 (SHA256) is a successor of SHA1, so it is also referred to as SHA2. It is an algorithm which provides an almost unique 256 bit hash value for a given input.

E. PBKDF2

Password-Based Key Derivation Function 2 is a hashing technique in which we can force the algorithm to perform slowly by increasing the iteration count.

F. Scrypt

Scrypt is an algorithm which demands very high memory hardware requirements in order to execute hashing functions, which helps in making attacker's attempts to crack passwords unfruitful.

G. Argon2

This algorithm has three variants: Argon2i, Argon2d and Argon2id out of which Argon2id is the hybrid of other two and is recommended most of the time. It combines the data-independent memory access (for resistance against side – channel timing attacks) and data-depending memory access(for resistance against GPU cracking attacks).

The parameter passed to Argon2 function are: memory, iterations, parallelism, salt-length, key-length.

H. Bcrypt

Designed by Niels Provos and David Mazières, bcrypt is a key derivation function for hashing passwords that is based on blowfish cipher, and presented at USENIX. In this technique, first the hash is calculated by passing in the plaintext into the function, but as discussed the hash is not enough for security, in addition to hash, salting is performed, where the hashed password is passed along with the salt that is unique for every website. Apart from implementing salt to prevent password from rainbow table attack, it also acts as an adaptive function, where over time the iteration count can be increased to make the password resistant to brute-force search attacks even with increasing computing power. The prefix "$2a$" or "2y" in a hash string in a shadow password file indicates that hash string is a bcrypt hash in modular crypt format. The rest of the hash string includes the cost parameter, a 128-bit salt (base-64 encoded as 22 characters), and the 192-bit hash value (base-64 encoded as 31 characters). [4]

V. Implenting bcrypt using NODEjs

In this section, we are going to see the implementation of bcrypt using node.js, bcrypt can be installed using node package manager as:

```
D:\project>npm install bcrypt
```

Then import the bcrypt module into your file as:

```
const bcrypt = require("bcryptjs");
```

Next, while registering the user we take the password and apply salt and hash functions to it as shown:

```
// Create salt & hash
  bcrypt.genSalt(10, (err, salt) => {
    bcrypt.hash(newUser.password, salt, (err, hash) => {
      if (err) throw err;
      newUser.password = hash;
      newUser.save().then(user => {
        jwt.sign(
          { id: user.id },
          config.get("jwtSecret"),
          { expiresIn: 3600 },
          (err, token) => {
            if (err) throw err;
            res.json({
              token,
              user: {
                id: user.id,
                name: user.name,
                email: user.email
              }
            });
          }
        );
      });
    });
  });
```

Now, the hash has been successfully generated and it can be stored in the database.

Now, when the user tries to log in with username and plain password, the plaintext password is first converted into its hash and compared with the hash already stored in database, if match is found then the user is logged in, else the user is prevented from logging in.

```
// Validate password
bcrypt.compare(password, user.password).then(isMatch => {
   if (!isMatch) return res.status(400).json({ msg: "Invalid credentials" });
    jwt.sign(
        { id: user.id },
        config.get("jwtSecret"),
        { expiresIn: 3600 },
        (err, token) => {
          if (err) throw err;
          res.json({
          token,
          user: {
            id: user.id,
            name: user.name,
            email: user.email
           }
         });
       }
     );
 });
```

VI. Conclusion

It can be concluded that website security is of utmost importance for building the users trust and one way of ensuring this is by storing the password in its hash form instead of plaintext in order to prevent the password against brute-force attack and dictionary attack. We have seen that even hashed password can be attacked using rainbow table attack, but by using bcrypt, we can apply a unique salt to the password, and thus prevent it from any kind of security attack. Other methods of storing has been discussed, and the implementation of bcrypt using nodejs has also been shown.

References

[1] Atishay Aggarwal, Pranav Chapekar, Rohit Mandrekar, “Cryptanalysis of bcrypt and SHA-512 using Distributed Processing over the Cloud,” International Journal of Computer Applications (0975 – 8887) Volume 128 – No.16, October 2015

[2] Ashwak ALabaichi, Faudziah Ahmad, Ramlan Mahmod, “Security Analysis of Blowfish algorithm”, 2013

[3] Praveen Gauravaram, “Security analysis of salt password hashes”, 2012 International Conference on Advanced Computer Science Applications and Technologies

[4] P. Sriramya and R. A. Karthika, “Providing Password Security By Salted Password Hashing Using Bcrypt Algorithm”, ARPN Journal of Engineering and Applied Sciences, VOL. 10, NO. 13, July 2015

[5] Alexander D. Kent, Lorie M. Liebrock, “Secure Communication via Shared Knowledge and a Salted Hash in Ad-hoc Environments”, 2011 35th IEEE Annual Computer Software and Applications Conference Workshops

[6] Narander Kumar and Priyanka Chaudhary, “Password Security Using Bcrypt with AES Encryption Algorithm”, 2017