

Term Project Results

COKE

by BlackShrimpBurger
(09) 김기훈 김병우 서인원 임세현
(12) 김현주 박봉우 정지현 조혜경

Abstract	
	<p>It has come to our attention that the current login systems of many websites do not take necessary measures needed to secure personal information of users. Thus, we have attempted to devise a secure login system to be used on various websites for the purpose of protecting users from malicious hackers accessing personal accounts. It is designed to secure even the accounts whose users are reluctant to the threats of security breach their confidential information is under. It is also more proving to be in need, as many users tend to habitually refuse to change their passwords.</p>

Table of Contents

1. Introduction	p.
1.1. Background	p.
1.2. Objectives	p.
	.	
2. Implemented Features	p.
2.1. Password Assessment	p.
2.2. Salt, Hashes, and Hash Chains	p.
2.3. Captcha	p.
3. How to Use	p.
3.1. Registration	p.
3.2. Login	p.
4. Conclusion	p.
5. References	p.

1. Introduction

1.1. Background

Early this year, there was a massive leak of personal information from three major credit card companies. This incident left people in fear that their personal information might be used for unwanted or illegal purposes. Some even suffered financial loss from cybercrimes such as Internet fraud. Following a series of similar breach of security, companies have taken on various measures to upgrade their online security systems.

In addition, the users of such websites were urged to change the passwords for their accounts to safe ones. Various media and organizations heavily promoted this precautionary method. If users were to choose a safe password, further damage would be prevented even when the company's web server is hacked. Despite these efforts, users still favor their old passwords because it is easier for them to remember. What is worse, they tend to use the same passwords for several of their accounts. In an article from 2008, the Guardian reported a study, which showed that 61% percent of the internet users had only one password for every one of their accounts. This means that if even one of the websites has a faulty login system, hackers can use the same acquired password to obtain information from other websites.

Although users should take more responsibility to their hands and implement measures to protect their information, that is not a fundament solution against security attacks. There will always be negligent or misinformed users such who do not fully realize the danger they are under. The more rational approach to prevent the damage from possible attacks are for the web providers to develop a system that secures the passwords of the users.

1.2. Objectives

Our team aims to design a log-in system that is able to secure everyone's passwords. This system will first recommend users to have a strong password by assessing the strength of their passwords upon registration. More importantly, our system will be capable of guarding

the passwords in unfortunate occasions such as when there is a leakage of internal database files. The system is also immune to machines' brute force mechanisms to obtain the password and access the account by implementing captcha.

2. Implemented Features

2.1. Password Assessment

According to Dashlane, an online password management company, 86% of the most popular websites have a weak password policy. For instance, it revealed that 66% of the sites allowed alphanumeric passwords and 55% still accepted the worst passwords on the web. Widely used websites like Amazon and LinkedIn were included in this list of websites with weak policies.

These websites are allowing the users to be easy targets for attacks. Thus, the first feature that was implemented in our system was password strength checking.

The system does its password strength analysis based on the following criteria:

- One or more lowercase alphabets
- One or more uppercase alphabets
- One or more numbers
- One or more special characters (\$%^#&)
- Longer than 12 characters

For each of the criteria, the password is given a point.

After the points are gathered, the password is considered one of the following for each range of the total points:

- 1~2: Weak
- 3: Medium
- 4~: Good

2.2. Salts, Hashes, and Hash Chain

An article from readwrite mentioned that 30% of the websites store users' passwords in plain text. This way, companies are putting people's passwords and their data at risk. Once a hacker gets a hold of the database files, it immediately gives the hacker the password. This can be used to obtain information from user's other accounts that use the same passwords.

This is why our system stores password data using salts, hashes, and hash chain. It enhances security by making it difficult to crack passwords even if the hacker has server's database files.

Salts: Generating random string of characters to be salts, and adding different ones to a password before hashing will assure the password hash to be different and random each time. Attacks using lookup tables, reverse lookup tables, rainbow tables become ineffective. Therefore, we decided to generate unique and random salts not just per-user per-password, but also per-login.

Hashes(Key Stretching): By implementing a slower hash function, in other words a CPU-intensive hash function, attacks take much, much longer. The idea is to have a hash function that is slow enough to impede attacks, but still fast enough to avoid a noticeable delay for the user. Thus, after adding salt we will be hashing with this function so that a password will be well-guarded against brute force attacks.

Hash Chain: A hash chain successively applies the cryptographic hash function to a piece of data and is able to generate many one-time keys from a single key or password. The length of a hash chain will indicate how many times a hash function will hash a string, producing a different value each time. Using the idea suggested by Lamport, implementing hash chains will make the system impenetrable by eavesdropping.

We combined all three of the features together and implemented it to our log-in system to store users' passwords. This will secure our system's confidentiality (eavesdropping/interception will not work) and ensure the passwords' confidentiality when there is an attack on the server (leaked internal database files will less likely lead to leaked passwords).

The following illustrates the login process.

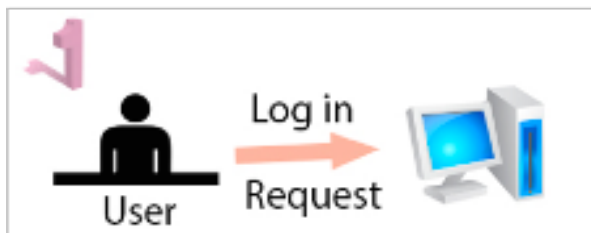
<initial db state>

Username	Hashed PW	Salt	Number
Shrimp	Hash1	Salt1	n

Initialize

Id_user	name	email	phone_number	username	password	salt	number
18	blackshrimp	blackshrimp@korea.ac.kr		blackshrimp	\$2yc6VRGjdn36	\$2a\$05\$1GxIR1TrCAsmckm2qCgCj\$	100

<log-in>



First, the login request is made by the user.

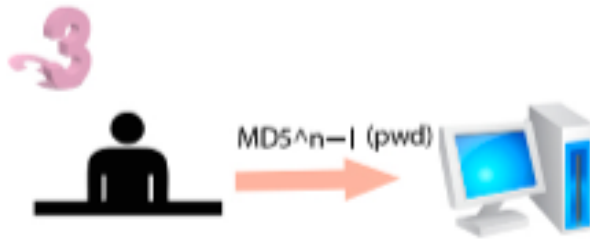
He or she types in the username and the password and clicks on the login button. This event sends a request to the database. However, sending passwords in plaintext leaves vulnerabilities that can be exploited by sniffing, only the username is sent when the first request is made.

```
$qry3 = mysql_query("Select number from $this->tablename where username='$username'", $this->connection);
```



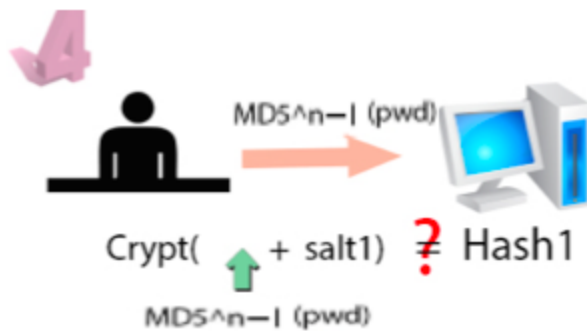
Then the database sends the user a value(=n) of a variable called "number." This allows the users to securely send their passwords after they have hashed it n-1 times.

```
$qry3 = mysql_query("Select number from $this->tablename where username='$username'", $this->connection);
$row_num = mysql_fetch_assoc($qry3);
$num=(int)$row_num['number'];
```



The user password is now hashed n-1 times with md5 hash algorithm and then sent. (\$tmp is a temporary variable that is later defined)

```
for($i=0;$i<($num-1) ;$i++){
    $pwdcrypt = md5($tmp);
    $tmp=$pwdcrypt;
}
```



The database takes this hashed password and encrypts it with a salt specific to the username that was originally saved in its database files. If this encryption value matches the initial Hash1 stored in the database, the user is logged in. Otherwise, it shows that the login failed.

```
$pwdcrypt = crypt(md5($pwdcrypt), $user_rec['salt']);
```

```
$qry = "Select name, email from $this->tablename where username='$username' and password='$pwdcrypt'";  
$result = mysql_query($qry,$this->connection);
```

```
if(!$result || mysql_num_rows($result) <= 0)
```

```
{
    $this->HandleError("$pwdcrypt"."Error logging in. The username, password or captcha does not match");
    return false;
}
```

```
$row = mysql_fetch_assoc($result);
```



Immediately after the user is logged in, the database updates the salt2 value using a random generator. The "number" variable is now changed to $n-1$, and the Hash2 becomes the encryption value of the $n-2$ -times-hashed-password with salt2.

```
$next_salt=$this->Mycrypt(); //Mycrypt() is RandomGenerator

$qry_new_pw = "Update $this->tablename Set password='".crypt($mp,$next_salt).'" Where username='$username'";
mysql_query( $qry_new_pw , $this->connection);
$qry_nextsalt = "Update $this->tablename Set salt='".$next_salt.'" Where username='$username'";
mysql_query( $qry_nextsalt , $this->connection);

$row_num['number']=(int)$row_num['number']-1;
$qry_nextnum = "Update $this->tablename Set number='".(int)$row_num['number'].'" Where username='$username'";
mysql_query( $qry_nextnum , $this->connection);
```

The hash chain is not susceptible to hackers' eavesdropping. Even when the hacker sees the data being sent (i.e. $\text{md}^n(\text{pw})$), he is unable to re-transmit the information to server since the database will then expect another irreversible piece of login data (i.e. $\text{md}^{n-1}(\text{pw})$). If the internal database files are acquired by the hacker the password is still not leaked because the database only contains the hash(- $\text{crypt}(\text{md5}^n(\text{pw}), \text{salt})$, salt, n , not any passwords in cleartext.

2.3. Captcha

Captcha was implemented to prevent brute force attacks. We tried to make captcha with floating point vulnerability at first. Our algorithm was so simple. First, get users' input as

string. Second, split strings character by character and store in to the array. Third, if there is '.' character or input string is more than 2 characters, then make login process fail.

But our captcha's vulnerability is if attackers know our captcha's algorithm, attackers can bypass our captcha easily through simple addition of code. So, we modified our captcha to prevent malicious programs by implementing image generator function.

```
<?php
    session_start();

    $im = imagecreate(200, 15);

    $first=10;
    $second=mt_rand(1, 9);
    $third=mt_rand(1, 9);
    $r_answer=(int)$first*($second/10+$third/10);

    $_SESSION['r_answer'] = $r_answer;

    $msg = "$first X (0.$second + 0.$third) = ?";

    $bg = imagecolorallocate($im, 255, 255, 255);
    $textcolor = imagecolorallocate($im, 128, 128, 128);

    imagestring($im, 15, 0, 0, $msg, $textcolor);

    header('Content-type: image/png');

    imagepng($im);
    imagedestroy($im);
?>
```

This is an image generating code in 'captchaimg.php'. Our captcha is math captcha with image. First, make problem for captcha randomly and calculate answer for the math problem. Second, send calculated answer through '\$_SESSION' to 'fg_membersite.php'. In 'fg_membersite.php', compare users' input and calculated answer and determine the user is malicious robot or not. Third, make generated math problem expression to image. Thanks to this string-to-image function, attackers cannot read math problem in the page. For now, Our captcha image is just not-modified string image but we can improve this image like real world captcha by adding noise in the image later.

```
<div class='container'>
    
    <label for='captcha'>Captcha*:</label><br/>
    <input type='text' name='ANSWER' id='ANSWER' maxlength="50" /><br/>
    <span id='login_captcha_errorloc' class='error'></span>
</div>
```

Generated image by 'captchaimg.php' is used in 'login.php' to provide math problem image to users and get users' answers. Entered answers are treated as text(even though they are numeric inputs) and send to 'fg_membersite.php' through '\$_POST'.

```
$r_answer = $_SESSION['r_answer'];

$answer = trim($_POST['ANSWER']);
```

Captcha's answer and user input are computed in 'fg_membersite.php'.

```
// strArr 배열에 answer를 저장한다
$strArr = str_split($answer);
```

First, split users' input and store in array by using str_split function. In case that php version does not providing str_split function, we made our own str_split function (see the detail code in the attached files).

```
$bot = false;

for ($i = 0; $i < count($strArr); $i++)
{
    if ($strArr[$i] == '.')
        $bot = true;
}

if (count($strArr) > 2)
{
    $bot = true;
}
```

Our malicious program detecting algorithm. If users' input has '.' character or input length is more than 2 characters. We regard that inputs as attackers' inputs (set \$bot variable true).

```
if (($bot == true) || ($cnt != $r_answer))
{
    $this->HandleError("Error logging in. The username, password or captcha does not match");
    return false;
}
```

If \$bot variable is true or captcha's answer is not matched with users' input, send error message and return login process false.

3. How to use

3.1. Registration

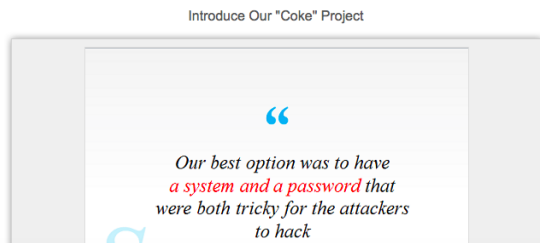
This is what our website (<http://blackshrimpb.mireene.com/exercise/>) looks like.

Black Shrimp Burger

Project COKE

By clicking [REGISTER] then the "register" tab, the user is directed to a new page where they can type in the registration information.

[\[HOME\]](#) ~ [\[REGISTER\]](#)



Membership website

- [Register](#)

When you type in the password of your preference, it automatically assesses the strength of the password based on the aforementioned algorithm.

Examples below show the assessment of the passwords when they are weak, medium, and good, respectively.

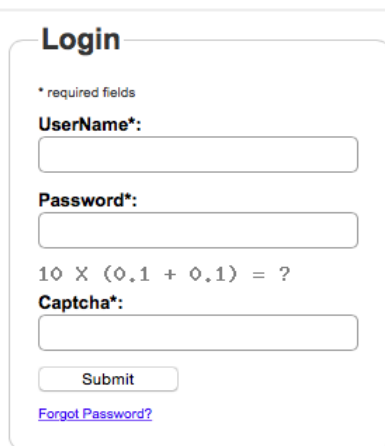
Register	Register	Register
* required fields	* required fields	* required fields
Your Full Name*: <input type="text" value="HeejoLee"/>	Your Full Name*: <input type="text" value="HeejoLee"/>	Your Full Name*: <input type="text" value="HeejoLee"/>
Email Address*: <input type="text" value="infosecurity@korea.ac.kr"/>	Email Address*: <input type="text" value="infosecurity@korea.ac.kr"/>	Email Address*: <input type="text" value="infosecurity@korea.ac.kr"/>
UserName*: <input type="text" value="Heejo"/>	UserName*: <input type="text" value="Heejo"/>	UserName*: <input type="text" value="Heejo"/>
Password*: <input type="password" value="*****"/>	Password*: <input type="password" value="*****"/>	Password*: <input type="password" value="*****"/>
Show Generate weak	Show Generate medium	Show Generate good
<input type="button" value="Submit"/>	<input type="button" value="Submit"/>	<input type="button" value="Submit"/>

3.2. Login

Clicking on the Login tab, you should arrive at this page below.

Users can log into their accounts by typing in their username, password, and the value for captcha.

*As mentioned above, our captcha has not distorted the numbers in the equations yet. At this state, the image is readable by machines. But it is later recommended that the numbers be distorted and made unreadable by computers.



The image shows a login form with the following elements:

- Login** (Section Header)
- * required fields
- UserName*:** followed by a text input field.
- Password*:** followed by a text input field.
- A captcha image showing the equation: $10 \times (0.1 + 0.1) = ?$
- Captcha*:** followed by a text input field.
- A **Submit** button.
- A [Forgot Password?](#) link.

4. Conclusion

LIMITATIONS

For the time being, we have used md5 hash functions and crypt functions. Our main goal was to give a general idea of a stronger system by using salts, hashes, and hash chains to store passwords. Safer algorithms can be used to enhance security in place of our functions. Also, our captcha image is not sufficient to misguide the machines. Using floating point error alone was not enough, so we made it into a image. However, in order for this image to work as a real Captcha, the numbers must be distorted so that it is only readable by humans.

COMMENTS

All of our teammates thought that working on this project gave us more insight on information security. By receiving feedback at each step of the way, we were able to see the vulnerabilities the system can have. We feel that the process of patching up these vulnerabilities was a great learning experience. With our implementation of the Captcha, it was difficult at times because we were starting from the ground up. When our initial attempt of applying floating point errors to distinguish computers and humans failed, we ended up thinking that we should use the original version of Captcha, distorting numbers, along with the floating arithmetic. It is a pity that we couldn't quite perfect our version of the Captcha, but we had the pleasure of taking up a new challenge and trying something interesting and novel. Also, our teammates really came together to think of a secure way to store the user passwords. Of course there were trial and errors but we feel fairly confident that we have come up with a storing method that is better than what is used in some big and popular websites.

5. Reference

-template: google docs

-registration form, the algorithm of the first feature is referenced from an opensource code (gnu licensed)