Ferhat Elmas
Przemyslaw Pietrzkiewicz
Javier Martín De Valmaseda Castro

Lab-6 Simply SQL Injection

Location of the Sources

When we start the web browser and navigate to http://bravo/, we have seen a register or login page. Then, we look at its source by right click view source. We have seen that there is a html index page that contains forms for registration and login and moreover, there are two php scripts that makes actual work for registration and login, namely, register.php and login.php.

To find their location, first hard way, inside we enter into bravo container:
$> vzctl enter 1001
Then, we searched this files:
$> find / -name register.php
And result is the place where these documents are saved.
$> /var/www/localhost/htdocs/register.php
Therefore, correct directiory is /var/www/localhost/htdocs/ , we look at inside of this directory, as expected we see index.html, register.php and login.php and a small connection opener script, db.inc.php.

Or from browser, since we know there is two php scripts from html source, we directly try to access them to generate some meaningful error messages
$> firefox http://bravo/register.php
And it returns index errors line 5 and 6 of the file /var/www/localhost/htdocs/register.php

SQL Injection

Generally, login logic firstly gets the user info according to submitted username. Then, submitted password is compared to that is got from database, if they are identical, access is granted, otherwise, it fails and no permission.
Therefore, to test this logic, firstly we create a valid account by the register function.

- username: hacker
- password: whiteHat

Then, we tested it with correct credentials. Since it should get the user info according to submitted user name, we tried some of forms of the correct user name to estimate / determine the structure of the query. For example, we tried

- username: h%
- password: whiteHat

We could log in so query must be a like query. Since its first letter is correct, it will definitely get our username and check the password which is correct. This seems reasonable but there can be multiple usernames that start with letter h so we created another account to see this.

- username: hackerDevil
- password: blackHat

To test, this time we entered:

- username: h%

- password: whiteHat

Then,

- username: h%
- password: blackHat

Both of them were successful, so when we supply a valid password and get its username by the provided user name, we are in. This could be implemented by get all users with the supplied user name and visit all of them by comparing the actual password to supplied password and if one of them is identical, grant the access because in our test, although we changed the password and in both cases user name filtered the same subset of the users, we could successfully log in.

Now, we can use what we have learnt to trick the system to give us administrative privilege. We don't know the user name or password of the admin but we will supply a special character to get all users in the database and enter our valid password for hacker which is whiteHat. According to above scheme, we will get all users and visit all of them for password and admin priviledge. Since we have all users, both fields will be correct since we supplied the correct password for the actual user hacker and admin is also a user in the database. We tried :

- username: %
- password: whiteHat

Finally, we are in with admin privilege.

Mitigation

When we check the source, we see a while loop that iterates over users whether password and/or admin privilege is correct or not. This is totally nonsense because user names must be unique identifiers for users. Therefore, we should get just one row. If there are multiple rows, something is not going well (maybe we add usernames without checking their existence, this is not correct because register.php checks it or some other sql injection is being tried with union or so.) If there is zero row, this user doesn't exist. In short, there mustn't be a while loop, there must be only an if check. Moreover, there is no sanitization of user input. User input is the start of the devil so it must be checked whether it contains special characters. One final advice is to choose better / more secure (cannot be estimated easily) username and password for the admin account. For general advice, error should be analyzed and error messages shouldn't disclose valuable information such as actual location of files in the server that we utilize in the first part.

What if?

If somebody gets admin privilege, then he will be able to what he wants. Actually this application is just a demonstration application so there is nothing to do (after you logged in) so there is no scripts to automate regular jobs but if the application could provide some admin scripts to modify the website / database, then since we got into with admin privilege, we would be able to use them. However, without scripts, we can modify nothing.

Yummy Passwords

Passwords are stored after they are hashed, this is a good practice because even if password database is disclosed, passwords won't be disclosed since reverse map is very difficult in the theory of hash alogrithm but here we see md5 is used which is hacked and insecure. In md5 case, calculating hash is very fast so what we have said above doesn't count, attacker will be able to calculate hashes and prepare lookup tables easily or he can use ready tables / databases on the web so using md5 is the bad practice. More secure hash algorithm must be

chosen such as bcrypt. Moreover, salt is used, it is also a good practice but it is same for all passwords, if we generate a random salt for each password, it would be more secure.