SQL Injection

For year, web pages that used a mixture of PHP and SQL have been the target of many attackers. It is quite easy to attack a web page that many webmaster believe to be protected. Many of which made very simple mistakes that can let attackers to easily manipulate data and more dangerously, delete a whole database. What many webmasters does not know is that from a simple line of code, it could lead to information leak to the hand of unwanted and even lost data.

All forms of attack will have some form of preventions; it might not stop the attacks but can make it more difficult for attackers to go on without breaking a sweat. Here we will talk about the types of attack and how to prevent such attacks. There are two popular ways to attack a website that runs a mixture of PHP and SQL, one is through URL injection in which attackers inject code into the address bar, and the other is through the input forms that attackers inject ambiguous code into an input field to gain access and even manipulate data within. These two popular attacks are known as SQL injection, it is an attack that many people in the field acknowledged but an explanation is still required. SQL injection is an exploit in which attackers inject SQL codes into an input, mostly an address bar or an input field such as a username or a password box to manipulate data and gain access to database.

The first popular type is URL injection; we will explain what they are and show examples of what it looks like in PHP and SQL. We should all know what an address bar is since most of the people that go online need to input a web page address into the address bar. What most do not realize is the information that are given out in the URL field can be alter to a point where the web page thinks that it is the administrator commands and therefore execute that commands. A line less than 5 character string is all that take to take over, for example,

‘OR 1’
A very simple line of code that can gain access into a database, note that the single quote is required because its use to end the string part of SQL queries. To clarify, when the user enter an input, the input is encapsulate into a string, which is enclosed by double quotes; therefore it’s treated as a string and cannot be execute. To bypass this, the single quotes are required so that it end the double quote and turn to code that attackers have inputted into an executable command. What ‘OR 1’ does is that it tells the web page that the input is always true since 1 is true and therefore OR 1 will always be true. This gives the attackers access to a specific table or column in a database. To clarify, let’s look at a hyperlink,

<http://www.homepage.com/login.php?id=3>
Many online users should be familiar with these types of hyperlink, they see it every time they go on a website not knowing that it can be use to manipulate data in that website. What if an attacker input a short line of code such as,

‘; DROP TABLE login; #
This input from an attacker can drop the entire table of users since whenever a command take place, it’s also shown in the address bar, not all but quite a number of them will shown in the address bar and therefore an attacker can take this opportunity to study how the database works and input his/her own commands.

The second popular type of SQL injection is through an input field, such as a login form where the web page needed information about its users. Here,

 ‘OR 1’

can also be used, what this looks like behind the scene is,

 SELECT \* FROM usersTB WHERE username = ‘ OR 1’

which select and display information in the table of users since ‘OR 1’ will always be true.

This normal code in PHP and SQL will look like this,

$name = “minh”;

$queries = “SELECT \* FROM usersTb WHERE username = ‘$name’”;

An input box asked for a name from the users and used SQL code to execute that command, in this case returned all the users table where the username has been assigned to $name. An attacker that knows how to take this opportunity will enter in ‘OR 1’ in the login field, thus gain access to data. See below,

 $name = “‘ OR 1’”;

$queries = “SELECT \* FROM usersTb WHERE username = ‘$name’”;

There are more dangerous exploits that an attacker can use, one of which is after an attacker gain access to important data; he/she could inject more code to manipulate them, such as,

'; DELETE FROM usersTb WHERE 1 or username = '

As you might have realized, this code delete the users table when the statement becomes true. Here is an example of what the code in the web page will look like,

 $ name= "'; DELETE FROM usersTb WHERE 1 or username = '";

$query = "SELECT \* FROM usersTb WHERE username = '$name'";

What this translate to is,

 SELECT \* FROM usersTb WHERE username = ' ';

DELETE FROM usersTB WHERE 1 or username = ' '

The attacker input an empty username in addition to another line of command in which it deletes the table of users if it is 1, a 1 is always true therefore delete the table of users if it is true, and since an empty username equals to an empty username, the injection executes and wiped out everything in the table of users. This is very dangerous since everything in a database that the webmaster has been building up has been deleted, more importantly if this data belong to a big company, lots of money and time goes to waste. Rebuilding take an inconsiderably amount of effort, and one wouldn’t want that.

 To prevent these injections from carry out, we need to create guidelines to help with the process of creating a safe secure web page. A webmaster sometimes forgets or believes that it is unnecessary to add extra line of codes to prevent exploits but these extra lines of code are required to prevent attackers from getting what they want.

First prevention is to limit the length of characters that can be input from users. For example, a username can be limited to an amount of character lengths, depends on the requirement of the web page administrator. This wouldn’t mean that the attacker will not succeed in his/her exploits, but a way to make it harder to carry out, “an eye for a finger”. One more prevention is to have data type validations, if an input field is asking for an age input than the webmaster should check for integer in that field and nothing else. Of course, an age of 16 is the same as “sixteen” but this is when the webmaster should clarify that he/she wanted an integer input rather than a string. Same goes if the input field is asking for a date input, then the date inputted should be validate to see if it is a date and not something ambiguous. Again, this prevention will not stop an attack from succeeding but it makes it harder for attackers. Last prevention but not the least is the use of a method called mysql\_real\_escape\_string(), which calls MySQL’s library functions mysql\_real\_escape\_string and appends a backlash to a single quote. As mentioned above, the single quotes are used to end the string part of the SQL queries, this prevention check users input to see if there are single quotes, if there are, then the method mysql\_real\_escapt\_string() appends a backlash before the single quote, render it useless. This prevention turned the input to a string of ambiguous characters which could not be any of the username in the database; it would be an unusual input if there is. To clarify more on this prevention, a code is required; let’s take injection from above, ‘OR 1’

 $name = “‘ OR 1’”;

$name = mysql\_real\_escape\_string($name);

$queries = “SELECT \* FROM usersTb WHERE username = ‘$name’”;

First line takes the input of ‘OR 1’ in which the attacker used to gain access to the database, second line takes that input and filtered it with our method, mysql\_real\_escape\_string(), which add a backlash to a single quote of OR 1, and behind the scene, this would looks like,

 SELECT \* FROM usersTb WHERE username = '\' OR 1\''

Instead of an injection that could execute, ‘OR 1’ turned to an ambiguous string. This prevention can be use on any exploits that has single quotes in the injection, and to clarify more on this prevention, more examples are required, let’s look at one more,

$ name= "'; DELETE FROM usersTb WHERE 1 or username = '";

$name = mysql\_real\_escape\_string($name);

$query = "SELECT \* FROM usersTb WHERE username = '$name'";

Same as above, after this code execute, it looks like this,

 SELECT \* FROM usersTb WHERE username = '\';

DELETE FROM usersTb WHERE 1 or username = \''

This is another ambiguous string that will not execute, and will be treated as a ridiculous name in the username field.

 These three preventions guideline should be in all webmaster’s list if they want to create a safe secure database, although, it will not prevent all form of attacks, we can be certain that it will keep attackers on their toes, and even have them give up if things get harder and didn’t goes as planned. One thing can be certain are that these three guidelines are not all of the preventions out there. There are as much attacks as there are preventions, but these preventions are more than enough to get started on a secure web page.

**References**

[**http://www.learnphponline.com/security/sql-injection-prevention-mysql-php**](http://www.learnphponline.com/security/sql-injection-prevention-mysql-php)

[**http://php.net/manual/en/function.mysql-real-escape-string.php**](http://php.net/manual/en/function.mysql-real-escape-string.php)