Network Security Project Report

Shuo Li sli53@jhu.edu Yuge Gong ygong@jhu.edu

April 11, 2013

1 DDos Attack

2 SQL Injection

In SQL injection part, a website written by jsp was build to implement SQL injection. Normal users can log in this website by using their own user name and password. Attackers, on the other side, are trying to log in this website by malicious input of user name or password. In order to show the effects of the detection and prevention of SQL injection of this website, we build two login page for comparison. One only permits normal login, while the other cannot avoid malious SQL injection.

2.1 SQL Injection Detection

Before we detect SQL injection, we need to understand how a SQL Injection happens. Let's first have a look at how a website without detection works when someone input his user name and password to log in.
In the jsp page of log in, there are following sentences by which we can gete the input strings from the user.

1. String user = request.getParameter("userid");
2. String pwd = request.getParameter("pwd");
3. String query = "SELECT * FROM `Tbl_User` WHERE `u_id` = '" + user +"' AND `u_password` = '" + pwd +"';";
4. ResultSet rs = st.executeQuery(query);

The first and second sentence are used to get user name and password input from user. The third sentence is used to do SQL query on Mysql database based on the informations provided in sentences above. We implement our SQL query in sentence 4.

Normally, a good person would provide legal informations, that is, a username and a password that had already been stored in database before. Then the server would use these informations to do SQL query to verify whether these informations are matching those in the database.However, a website attacker can utilize this verifying procedure to log in the website without verification. We will explain it then.

As the website without detection will not check the user input, any input will be accepted by server as long as the length of the input satisfying the requirements. Take an malicioius input of username as a SQL injection example. An attacker cannot just input "Alice" as his user name because the database does not store such a user name or the attacker cannot provide a matching password for the existing user name "Alice". To continuing his attacking, he can input "$1'$ OR $'1' = '1$'' instead of "Alice". Then he can input any password (except an empty one) to log in the website without any forbid. This is because the server is now doing the SQL query " SELECT * FROM `Tbl_User` WHERE `u_id` = $'1'$ OR $'1' = '1'$ AND `u_password` = $'anyword'$ ". As $'1' = '1'$ will be true under any condition, the attacker can always log in the website without verification.

After knowing how an attacker user SQL injection to attack the website, we can then develop the method to detect the situation. In our case, we use regular expression(`[^0-9a-zA-Z]`) to check user's input. If a user name or password contains any character that is not a letter or figure, then the server would determine that the input contains illegal character(s). Thus, the username or password is invalid and cannot be used to log in the website.

2.2 SQL Injection Defense

Using regular expression can standardise the input from users, which means implementing a detection of SQL injection can filter the malicious input from attackers to a large extent. Detection, however, cannot promise server a prevention of a SQL injection. A more efficient and safer way to defense attacker from SQL injection is being used here.

Instead of using simple SQL query to do search on database, we prefer prepared statements and storage procedure here. By implementing prepared statements, we can first define our SQL query, which make attacker cannot modify our SQL query later. By implementing a storage procedure, server will not just patch the incompleted SQL query sentense with input strings from users to get a completed one. On the other hand, we will regard these input string as input parameters of a single storage procedure stored in database before.

1. String user = request.getParameter(“userid”);
2. String pwd = request.getParameter(“pwd”);
3. CallableStatement cs = con.prepareCall(”CALL verifyUser(?,?)”);
4. cs.setString(”user”, user); cs.setString(”pwd”, pwd);
5. ResultSet rs = cs.executeQuery();

We can get the input strings of user name and password as usual, but in the step 3, we will not use the “SELECT *” sentence any more. Instead, first define our parameterized query, then we set parameters for this prepared statements. After this, we call the storage procedure “verifyUser”. By doing so, we avoid using the input strings from users directly, which can reduce the posibility of SQL injection to server.

Table 1: Website without SQL injection detection

Input Username	Input Password	Reaction
		Username cannot be empty!
Alice		Password cannot be empty!Wrong password!
Yuge	admin	Log in successfully!
1’ OR ’1’ = ’1	Any Password	Wrong password!
1’ OR ’1’ = ’1	1’ OR ’1’ = ’1	Log in successfully!

Table 2: Website with SQL injection detection

Input Username	Input Password	Reaction
		Username cannot be empty!
Alice		Password cannot be empty!Wrong password!
Yuge	admin	Log in successfully!
1’ OR ’1’ = ’1	Any Password	lllegal characters!
1’ OR ’1’ = ’1	1’ OR ’1’ = ’1	lllegal characters!