CO331: Network and Web Security

Tutorial 6: Code review for PHP and SQLI*

February 13, 2018

For each fragment of PHP code that follows:

- Find an SQL injection vulnerability.
- Write an exploit that delivers the requested payload.
- Fix the vulnerabilities in the PHP code that makes the SQL injection possible.

The database server runs MySQL and contains a `users` table that was created with the following SQL command:

```sql
CREATE TABLE users (
  userid INT(4) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(30) NOT NULL UNIQUE,
  password VARCHAR(30) NOT NULL,
  forename VARCHAR(20) NOT NULL,
  surname VARCHAR(20) NOT NULL
);
```

1. The payload for your SQL injection attack should log in the attacker with the username `admin`. Assume that the `login()` function is defined elsewhere, and logs in the user with the username provided.

```php
<?php
$u = mysql_real_escape_string($_GET['username']);
$p = $_GET['password'];
$q = "SELECT * FROM users WHERE (username = '$u') AND (password = '$p')";
$result = mysql_query($q) or die(mysql_error());
$matching_users = mysql_numrows($result);
if ($matching_users === 1) {
  $username = mysql_result($result, 0, "username");
  login($username);
} else {
  die("Account not found");
}
?>
```

*Thanks to Chris Novakovic `c.novakovic@imperial.ac.uk` for preparing this material.

2. The payload for your SQL injection attack should show the usernames and passwords of all registered users.

```php
<?php
$uid = $_GET['userid'];
$q = "SELECT forename, surname FROM users WHERE userid = '$uid'";
$result = mysql_query($q) or die("Database error");
$users = mysql_numrows($result);
$i = 0;
while ($i < $users) {
  $forename = mysql_result($result, $i, "forename");
  $surname = mysql_result($result, $i, "surname");
  echo "User's full name: $forename $surname";
  $i++;
}
if ($users === 0) echo "No user with given user ID";
?>
```

3. The payload for your SQL injection attack should log in the attacker with the user ID `1`. Assume that the `login()` function is defined elsewhere, and logs in the user with the username provided.

```php
<?php
$uid = mysql_real_escape_string($_GET['userid']);
$p = mysql_real_escape_string($_GET['password']);
$q = "SELECT * FROM users WHERE userid = $uid AND password = '$p'";
$result = mysql_query($q) or die("Account not found");
$matching_users = mysql_numrows($result);
if ($matching_users === 1) {
  $username = mysql_result($result, 0, "username");
  login($username);
} else {
  die("Account not found");
}
?>
```

The following resources may be helpful:

- PHP APIs for MySQL:
  - `mysql` (old, deprecated): `https://secure.php.net/manual/en/book.mysql.php`
  - `mysqli` (newer): `https://secure.php.net/manual/en/book.mysqli.php`
- MySQL SELECT syntax: `https://dev.mysql.com/doc/refman/5.7/en/select.html`
- PHP Data Objects (database abstraction layer): `https://secure.php.net/manual/en/book.pdo.php`

Sample Answers

1. Vulnerabilities:

   - Line 4: unsanitised attacker-controlled parameter `password` concatenated with SQL query
   - Line 5: output of `mysql_error()` shown in response body

   Possible exploit: `vulnerable.php?username=admin&password=x' OR 1=1)--`
   Possible fixes:

   - Line 4: sanitise `$p` (e.g. with `mysql_real_escape_string()`) before concatenating
   - Line 5: use parameterised queries instead of `mysql_query()`
   - Line 5: don't include `mysql_error()` in `die()`

2. Vulnerabilities:

   - Line 3: unsanitised attacker-controlled parameter `userid` concatenated with SQL query
   - Line 4: malformed SQL query gives noticeably different output to that given when no matching users are found

   Possible exploit: `vulnerable.php?userid=' UNION SELECT username, password FROM users --`
   Possible fixes:

   - Line 3: sanitise the value of `$uid` (e.g. with `mysql_real_escape_string()`) before concatenating it with the rest of the query
   - Line 4: use parameterised queries instead of `mysql_query()`
   - Line 4: also display the "No user with given user ID" error message if a malformed query is executed, to leak less information

3. Vulnerability:

   - Line 2: attacker-controlled parameter `userid` passed into SQL query but its value is not expected to be a string, so `mysql_real_escape_string()` performs insufficient input validation

   Possible exploit: `vulnerable.php?userid=1 -- &password=x`
   Possible fixes:

   - Line 2: instead of using `mysql_real_escape_string()`, check that `$uid` only contains digits (e.g. with a regular expression)
   - Line 5: use parameterised queries instead of `mysql_query()`