### Assignment 8: The trouble with non-prepared SQL statements

Consider the following (amateur) PHP script for a login page:

$username = $\_GET[user];

$password = $\_GET[pwd];

$sql = "SELECT \* FROM usertable

WHERE username = ’$username’

AND password = ’$password’ ";

$result = $db->query($sql);

if ($result->num\_rows > 0) { /\* Success \*/ }

else { /\* Failure \*/ }

1. Give a value that an attacker can enter in the field called *user* that will result in a successful login? Assume the attacker does not know any usernames or passwords for the site.
2. Suppose we change lines 1 and 2 to
3. $username = addslashes($ GET[user])
4. $password = addslashes($ GET[pwd])

The *addslashes* PHP function adds a slash before every quote. That is *addslashes("a'b")* will output the string "a\'b". Explain why this prevents the attack from part (a).

1. Does *addslashes()* completely solve the problem? Consider the GBK Chinese unicode character set. Some characters in GBK are single bytes while others are double bytes. In particular, the following table shows a few GBK characters:

|  |  |
| --- | --- |
| 0x 5c | = \ |
| 0x 27 | = ' |
| 0x bf 27 | = 뼧 |
| 0x bf 5c | = 뽜 |

1. That is, the database interprets 0x bf 5c as a single chinese character. Show that using a simple *addslashes()* that adds a slash (0x5c) before every quote (0x27) as in part (b) leads to a SQL injection attack. What value of *user* will result in a successful login?