Chad Ballay

CYBR 350-342

Week 5

7/12/2020

Hack.Me

First observation is that I love that it spawns off a sandboxed environment for me to learn in. While I know that in most situations I really can’t break anything while trying to learn, it helps free things up a bit to know that I \*really\* can’t. Takes some of the unfounded worry off of me. Not rational but it still is meaningful.

The second is that they released this under the Creative Commons. I’m a copyleft kind of guy. That’s a bigger discussion outside of the scope of the class but so much wasted effort is spent dodging around the minefield that Disney and other Intellectual Property types have made the laws into that it stifles stuff like this. Good on them for doing this. <https://en.wikipedia.org/wiki/Copyleft> I found a two minor capitalization type errors. They may never accept my PR but hopefully does some iota of value here. <https://github.com/powderflask/u-hack-it/pull/1>

Both of those are more of meta observations about the learning and not the actual learning.

**SQL Injection**

Right of the bat seeing that it’s returning a wildcard for the columns gives me a bit of heartburn. Table structure shouldn’t change often but always enumerate the columns you want in the order you want.

Nevermind the fact that the users could short circuit the logic of the where clause due to unsanitized input. Looks like it takes the result and reads the first record so in my instance this was [pooch@dogs.ca](mailto:pooch@dogs.ca).

A screenshot of a cell phone

Description automatically generated

Since XKCD has a relevant comic for everything I tried to recreate my own Bobby Tables type experience. MySQL isn’t my strong suite so I couldn’t figure out how to get the -- comment to eat the last quote and semicolon. But the gist is that if they are going to take my input and run it as part of the SQL directly then with enough effort this could allow me to run arbitrary SQL commands.

A screenshot of a cell phone

Description automatically generated

**XSS (JS Injection)**

This one was fun considering the discussion topic for this week. Jumped right in and had it pop the alert.

A screenshot of a cell phone

Description automatically generated

Decided to see if I could use an iframe tag to load a different webpage. Couldn’t get that to work but was fun trying to figure it out.

The last part was phenomenal!!! You could leave a specially crafted comment that would then mean any future loading of that page would run the code that had been left behind. My pop up blocker stopped it but that potential for abuse is huge.

A screenshot of a cell phone

Description automatically generated

**Session Hijacking**

This one was way over my head at first and it took a lot of external reading to grok it. So session id’s are unique and the design of this application is that we trust the cookie to keep that info to allow for persistence. HTTP is stateless so there needs to be some thread that keeps things all together. If the hacker gets my session id then they can impersonate me without having to know my userid/password. What I don’t understand is in this example the sessionid is unencrypted and accessible by the user. Is this a case of making a contrived example or do I not understand how “HTTP Only” cookies would be used. This kind of info should only be read and written by the server. So why are we able to see it in this example?

A screenshot of a cell phone

Description automatically generated

**Tutorial of web app hacking using SQLi and XSS**

Loaded the “image” of monitor.php and was amazed at now having a full blown UI to explore the rest of the system.

A screenshot of a cell phone

Description automatically generated

Since the login functionality is the first place where it probably starts hitting the database I went and checked it’s code. Found the part for the SQL connection string.



From there I had fun exploring the DB schema.

A screenshot of a computer screen

Description automatically generated

From there it was a pivot to the Blog part of the lesson. Here again we were able to see the session id and I don’t know if my understanding about HTTP Cookies is correct.

*<SCRIPT>alert(document.cookie);</SCRIPT>*

A screenshot of a cell phone

Description automatically generated

But it still goes to show that you can never trust user input. Ever. Was glad to see the XKCD comic referenced as well.

A screenshot of a cell phone

Description automatically generated

**Lessons Learned**

* XSS is really hard to get right. So many shortcuts that developers can take. Users exert pressure to do the least amount of work. In the end those things add up and stuff gets through.
* Never trust users. Sure most of them aren’t going to push at the soft spots in your defense. But all it takes is one.
* Parametrize your SQL and never take input and run it.
* Get your code on to the webserver and then exploit.