Chad Ballay

CYBR350-342N

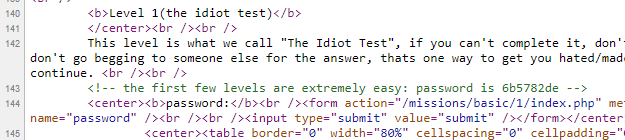
Week 6

08/08/2020

HackThisSite

# Basic – Level 1

This one starts off with a gentle mocking tone and not much else for direction. Starting with the beginning reconissence I viewed the page’s source to even see what kind of authentication and form submission it was doing. The password was in the HTML itself.



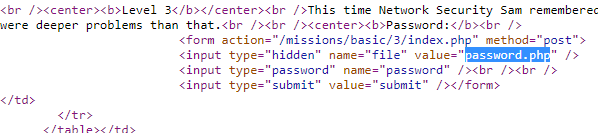
With a cut and paste then a carriage return, I had received a successful login. This highlights the need to not let comments make it into production releases. Most of the time this isn’t a thing to worry about but every once in a while someone leaves something important in a comment. A good practice is to have the CI/CD pipeline remove them or as a last resort have them not be sent at the webserver layer.

# Basic – Level 2

Didn’t see anything in the source of the page at first glance. Even tried a generic password to see if it worked and it failed... Upon re-reading the text of the page it dawned on me what they meant about failed to upload the file. A blank submission of the form and I received a successful login.

This one is a bit tougher to draw a quick lesson from it. Should the code do a file found check before trying to validate the form against the contents? Should the contents of the file have some sort of validation check? All of these and more should be done but no single one of them seems to be the end all solution for this one. Maybe the lesson is to have really good QA testing since this would have absolutely been one of the tests a decent QA would try.

# Basic – Level 3

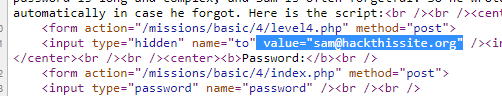
Starts off with an acknowledgement of the previous scenario now being rectified. But then goes on to highlight different ways of failing. In this case, the file that contains the password to validate against is listed. 



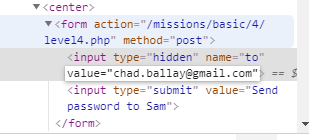
The end user should have no way to even know of this file’s existence much less a way to access it. Running a basic URL fuzzer might discover the file since it is generically named “password.php”.

# Basic – Level 4

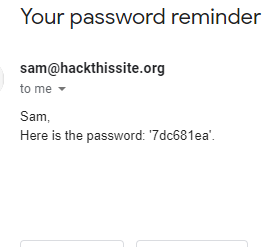
The fourth one gets to be kinda interesting. It relies on a hidden form value to specify the email address to send the password to. Trusting the end user for anything is always dangerous. In this case, the form contents can be modified, including hidden ones.



Using the developer tools within Google Chrome, I was able to change that value to my own email address.



In a short while the password was sent to my account.



Upon trying that I was able to login.

# Conclusions

All four of these exercises highlight how easy it is to have the best intentions and instead sow chaos. A forgotten file. A ham fisted attempt at security through obscurity. Etc, etc... Code reviews, best practices, and solid QA testing will do wonders for catching these issues. Sprinkle a healthy dose of not trusting users with any extra information then the bare minimum also helps. No extra comments, no accepting data from them, nothing. These all become additional attack surfaces.