Tyler Jackson
HW4
Software Security
Write-up

Process, Pitfalls, and Explanation

Configuration

For this project I reviewed Owasp Broken Web Application Mutillidae. Initially I tried to set up Hacme Bank but after hours of trying to configure my windows machine to allow me to deploy the application I gave up and switched to Owasp. There were a lot of problems with using newer windows and visual studio editions with those old file types. To set up Owasp I was able to download a virtual machine image from their website and open it up in virtual box which was super easy. There were a few pages that just wouldn't load on the newest version they have packaged up from their website. Specifically the register page and the add blog post page were both blank white screens (other than the sidebar and the header). However, if I enabled my vm to access the outside network, pulled down the latest mutillidae code from github I was able to get these pages to work, so they must have pushed up some fixes in the code. However, with the virtual machine offered on Owasp's website they have changes not in github that have the db configuration already configured. So, I had to copy these changed files to a temporary location, pull down the updated code from mutillidae and then copy these temporary files back into mutillidae. After all this was done I returned my VM to a host-only network and was ready to finally actually start the code review.

Process

My process for this project was to first try and find vulnerabilities on the live web application and exploit them without looking at the code. Mutillidae comes programmed with hints about how to exploit the application and ways to increase the security to make it harder. Being a noob relatively speaking, I kept the security on the lowest setting they had (none). I also tried to ignore the hints as much as possible as they were more helpful if I found myself stuck with a particular exploit. The web application is huge, and the way it is structured is by the type of attack. This was nice, because I at least had a hint about what type of attack would be possible. For each vulnerability I found, I would navigate to the page in my browser on my host machine and play with various inputs. I would try sql injection, javascript, and html. I used Chrome's Inspect Element dev tool quite a bit to help me see how some of my exploits were being interpreted. After I found and took pictures of 15+ vulnerabilities I tried to figure out how to fix each one in the code. Luckily the pages' names are very similar to the files' names in the code base. For the sql injection attacks, most of them could be fixed by validating the input in the applications main database class. This was nice because one fix often times fixed multiple vulnerabilities. For most of the html injection and XSS I was able to fix them by simply encoding the output based on what it was being used for (CSS or HTML). Lastly, I wrote up each vulnerability, where it came from, and what type of attack it would be susceptible to. I included lots of screenshots to how the vulnerabilities would be exploited, and also to where in the code they should be fixed. I also suggested for each one how a developer could go about fixing them. I then ranked the vulnerabilities using a likelihood x impact matrix. This felt somewhat arbitrary as the application was designed to be exploited. I tried to pretend as best as I could that this was an actual application, and just the risk to the “company” accordingly.

Failures and Successes

Definitely the hardest part of this assignment was the configuration process. That being said, I would imagine in the real world this would be much less of an issue as a company would have environments in place hopefully for an audit team to work with. It was also harder than I thought to just jump into a huge application like this. If I did this assignment over again I would adjust how I went about looking through the code for the first time. I immediately went to particular pages and tried to figure out how to fix some of the vulnerabilities. It would have been helpful to have started by looking at some of the configuration files, see how the application is interacting with the database, look at the utility classes, etc. Most of the functions that I recommended the developers use to prevent the exploits were centralized. Things that went well were the actual exploits. I was having a lot of fun with this project and probably spent way too long on tinkering with vulnerabilities to see just exactly what I could do. I also tried to tone down my evidence of the vulnerabilities so that they would provide proof of the exploit, but not anger the company. I could probably play with this application for weeks and still not find all the exploits.

Conclusion

Overall I think this project was really fun, and really useful to get hands-on learning for how these vulnerabilities can be prevented. I have had 2 internships and a contract job and am pretty surprised at how little code review has been done at these companies. The code analysis isn’t really aimed at all towards security, but rather towards what the end product is supposed to be capable of. The only barrier between vulnerable code and what was live in the application was if the person who happened to be reviewing my code noticed any vulnerabilities.