Ryan Mitchell

CS 405

February 27, 2022

Journal: Portfolio Reflection

Secure coding is a skill that I always thought would be an incredibly convoluted and technically demanding philosophy that would only become relevant when coding professionally. I’m happy to discover that I was totally wrong, and that some of the principles are very straightforward and don’t require an incredibly high level of mastery. In fact, one of the best coding principles is “keep it simple” and I think that’s beautiful. On the other hand, between both this course and graphic design, my biggest secure coding strategy takeaway is “don’t use C++ ever”. I’m joking of course, but it definitely feels like C++ has many, many vulnerabilities that require a certain level of attention.

In regards to not leaving security to the end, I felt that this was the best course of action ever since I took a class which talked about scrum and the development pipeline. When I saw that development and testing were distinctly separate, I initially didn’t bat an eye. When I started getting more into the class and the principles of the development process in the real world though, I strongly recall coming to the conclusion that testing should be intertwined. In fact, one of the most prominent coding practices I follow to this day is “don’t wait until the end of coding to debug, do it in pieces”. So why wouldn’t this philosophy apply on a larger scale?

I think risk and cost benefit mitigation is extremely subjective to the company’s status. I think lower end startups, or budding groups of developers coming together don’t exactly have a target on their back (but I could be wrong). I think risks can be taken early on, but within reason. In addition to this, I think there should be a dynamic policy of adaptation that gauges the threat level based on success. What I’m trying to say is that risks should be taken early, but taken in a way where a safety net is already pre-planned. Obviously if resources are abundant, or the nature of the project is highly sensitive, then it’s most likely wise to work against the risk factor. It’s hard for me to judge until I’ve worked in the field more closely with professionals and had a chance to learn directly.

My first security policy recommendation is to use java, java is awesome. My second security policy that I would recommend if java is clearly not an option, then try Python. Ok, maybe I’m still a little salty from my OpenGL class… I truly think the only missing piece that makes C++ a more comfortable language is a more thorough debugging system. Perhaps there’s IDE’s or applications for C++ coding that I haven’t had the time to investigate. Just like we learned, constant testing and automation are your best friend when developing, and while I may have a love hate relationship, C and C# and C++ have all existed for a long time for a reason. I think many security policy principles are not only tips to keep your system safe, but also to ease the development process. In all honesty, my understanding of pointers has historically been weak for reasons I cannot explain, something wouldn’t click. However, by studying the properties of pointers and being exposed to them in two simultaneous classes, I noticed that I was generating a lot less errors, since memory management is such an important focus in C++ development. So my first security policy recommendation is to focus on data structures and memory management from a conceptual standpoint, then implement your techniques.