Tammie Pease

CS 499

Artifact One

Software Design and Engineering

The artifact for fulfilling my Software Design and Engineering credential is my SQL Injection code. This software is from my SNHU Computer Science Secure Coding Course, created in early November of 2021. It was initially a small piece of code that was meant to show what could happen with an ‘or statement SQL injection’ and making a fix within the code to help prevent that specific SQL Injection. It was an additive part of that course’s module in which it wasn’t the main focus of the section. It did, however, do what it needed to do with a single if statement to check if there was ‘or’ and that was it for the program. What the code holistically did was set up an SQL database within the C++ code using MySQL and then ran a query to check for a statement to pull information from a part of the database. However, an SQL injection was done as well, in which it revealed the entirety of the database instead of solely the part.

This artifact touches upon many skills, as this artifact started as a small little example project, but then with time and polish was able to do much more than originally intended. Expanding upon the processes that it was used for and using them for something new and innovating the old. This item was selected because it was one of the best of my choices that could be updated with enhancing the sum of its parts to do more than it did. Not to mention, this code that allowed me to sit down and work on something that wasn’t my full set of code alone and it challenged my understanding of the project. Because of needing to understand the full depth of the project I learned every line and how they all weaved together. I would have had modules of python code in some other school projects, but with the functions in the same space which created a new layer of challenges to the system. The way the SQL injection works in of itself is a somewhat challenging concept to understand. It can create many layers of problems for C++ users and even MySQL itself so it would be good to understand it and more so how it works to prevent it. Later on, I was able to go into more depth with my knowledge within the software Visio and make sure that the files themselves would be able to be read. By using a proper pathing sequence that after zipping and extracting multiple times still got the project to run properly.

The sole enhancement that I intended to do was proper use of prepared statements to help prevent the attacks. However, the scope of the project was shifted to adding new measures and systems into the framework. The skills most applicable with this artifact are of 03 and 05 of CS-499. With 03, the idea of designing and evaluating computer solutions with algorithmic principles was implemented with the idea of working with the code and making prepared statements. Additionally, the extra enhancement of creating a new system within the solution such as allowing the addition of more SQL data within the small code through prepared statements rather than direct insertion. Refurbishing the former SQL injection prevention method as another stopper for bad actors was done as well. By adding quotes around the statements before being executed but also checking to confirm that or is not ‘or’ such as in a normal name or word but ‘ or ’ for the spacing of an or statement. Not to mention with these all being implemented, I had to understand the trade-offs to the solution in what would be possible within a few weeks, and what would likely be done against stopping every single variation of injection because the trade-off of ultimate prevention is very cluttered and hard to read code. Trying to make code that worked alongside the former code was a trade-off as well, in keeping the former injection code the same. However, I went with this as that code that should be considered from a different source and not from the code but from a ‘malicious user’. Even if it would be easier to remove the trade-off would be the inability to accurately test if the SQL injection was prevented and would remove what made the SQL injection code the former code it was.

Secondly, the idea of skill 05 is the idea of trying to help mitigate security flaws, along with the enhancement of security for data and resources. With that, using prepared statements in to help with the security of the system for ensuring no SQL injections happen is one of the valuable security skills to carry. This idea of protection can be expanded upon but having multiple different layers of security with the system, it can allow for the best possible security we can have by using defense in depth ideologies. By having multiple systems and ways to help prevent security flaws from being found you add layers of protection that dissuade attackers from continuing their attack. The security of checking for or injection, preparing statements, and sanitizing them before implementation can help ensure the best results for security possible within the scope of limited time and resources I had available to me here.

Overall, I believe that I improved the system with many varied enhancements by having properly prepared statements but also adding in creating new information and also having multiple ways to help prevent MySQL injection. I believe that I met the course objective of designing and evaluating solutions with algorithmic principles and computing standards for the fact that I did change what I could. This was done with the idea of trying to not jeopardize the main aspects of the system that were intrinsic to it, being the runtime injection code as a main example. Being that it caused many problems for me in that it was meant for specifically only the prevention of OR SQL Injection within my Secure Coding course. So, it was a process of learning what to accept versus reject along with and the benefits and trade-offs that it caused from such. A lot was added, but trade-offs for what could be done with that scope of time, such as not adding update or delete operations to the system could not be done in a reasonable amount of time. The code met what I needed to do and reaching that point is an accomplishment. So, in the end, the updates to the coverage plans were added upon from what I set as the minimum!

I did a lot, but it was a struggle to work on anyway. Coding is not my best aspect when it comes to the courses, but I was able to do what was needed and fulfilled what I needed to do. The documentation I found for MySQL didn’t carry any examples that I could help base the design on so Stack Overflow was my friend when it came to errors or how it worked. Understanding how it worked and would be structured, such as sqlite3\_bind\_text was incredibly useful. This alone helped show me that not every piece of software is going to have the best documentation but in the process of such, it helped me go for out-of-the-box solutions. This was probably the most function crossover I encountered in a piece of code so far. And so, to understand it all, I took a separate document and then mapped out exactly how each piece of the code connected. In this, taking a few steps back and figuring out how everything within the code worked helped me develop processes and where I would put functions to work the way I intended. Having separate documentation for notes allowed for the code to not become over-cluttered and set reasonable goals compared to overreaching which I had a major problem with previously. As this then allowed for the code to be the best it could be without trying to be what it wasn’t able to do. Not to mention with the reduction of comments as well and adding them where they are important, helped the readability of the code as well. So now to me at least, it looks better and a touch more professional than before! There were many challenges such as time, ability, understanding of the code, and trying to see everything that happened within the code it allowed me to learn how to work with time, use other documents to help understand what is going on, and that official documentation isn’t always the end all be all. For what it’s worth, I think I did well!