SSD Reflections – Ashokkumar

In this reflective essay, I discuss the information and skills I acquired as well as the insights I obtained from this experience. I also describe the measures I took to improve my understanding of secure software development in order to design a Python-based command-line interface (CLI). When building a safe CLI-based tool for internet forensics for the Dutch Police.

Explored the waterfall and agile techniques to software development, focusing on the consequences of both for producing safe software. Learnt the Unified Modelling Language and how it can be used to facilitate software development. Acquired an understanding of the industry standards required to design safe software. Recognize the significance of fostering a culture of risk awareness within an organisation.

From our first group discussion, I learned that the UML flowchart is a useful tool for making the possible cyber security danger clear to all levels of staff, including upper management who may not have much experience in this area. As I've learned, it's not simple to draw a UML diagram that can be understood by anyone. If I want to make an easily digestible UML, I need to have a firm grasp of the following: I need a clean layout for my charts. When creating a UML chart for a team's workflow, for instance, it is important to arrange and represent the activity clearly so that everyone can track the movement of the item from one person to the next. I also need to keep things as easy as possible. Problems should be visible in the UML diagram and easy to pinpoint. Cyber security threat identification using a UML diagram, including the objects, flows, and output (Setapp,2021).

All of my work, from designs and code to tests, can be found in my GitHub repository, which you can access through the link provided. To view the secure CLI-based application my team and I built for a practise session, please go here. This repository on GitHub contains all of the automated unit test scripts I wrote for this CLI-based application. I used the "safety" module in Python to scan the open-source libraries we used for vulnerabilities, and the results are shown in this folder on GitHub (Ponta et al., 2020). Screenshots can be found in this directory on GitHub, and they demonstrate how I used two listing tools (the "flake8" and "pylint" libraries) to automatically evaluate code quality (Kapil, 2019) and make appropriate modifications.

Moreover, I used what I had learned about linters like "flake8" and "pylint" from our teacher and the numerous programming exercises we completed during the module to contribute to our team's work. A highlight of this course for me was engaging in stimulating and educational group discussions with the professor and other students. Using what I knew of the Unified Modelling Language (UML), I drew a sequence diagram showing the primary events that lead to cryptographic failures, and shared it at the outset of our conversation. It was thought that this would be a good way to show the several ways in which private information might be leaked. By taking part in the first group discussion, reading the posts of my peers, and offering constructive criticism, I realised how important it is to keep up with developments in security-

related evaluations and best - practise, and to put myself in the shoes of a cyber attacker so that I can use a mix of methodical and intuitive insights to make my software as secure as possible against the most common and cutting-edge threats. I was able to evaluate the "TrueCrypt" case study and create an ontology for the software in our second group session, which helped me understand how user behaviours might contribute to exploiting the application's security weaknesses and obtaining sensitive data.

I now have a better understanding of the technical difficulties associated with things like user interface design, database-code communication, database-front end alignment, securing and simplifying administrative tasks, and concealing irrelevant data.
When faced with a security difficulty, I've learned to keep my solution as straightforward as possible, particularly when character counts are tight. I need to explain the difficulties and possible solutions in the design and the code in the simplest possible words.

The Python language's modules were my entry point. The application's functionality may be simply achieved by assembling its many pieces, which I will do. Through Python's modules, I was also able to learn how to build up an OTP.

However, as a team member, I was able to conduct meetings and draught emails by relying on the knowledge of my co-workers to ensure that we all worked toward our common goal of creating a secure CRUD-capable CLI-based application. So, despite initial communication and prioritisation issues due to competing design ideas, I was able to resolve them by outlining why it would be more efficient to design a command line interface (CLI) based application that could handle encrypted data and multiple threads to accomplish the same goal. I took control of my tasks, going above and above to do things like write unit tests for encryption and validation code, centralise global logging, and lint existing code for quality and security.