# CS 405 Project Two Script Template

Complete this template by replacing the bracketed text with the relevant information.

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Richard Pashko |
| **2** | My newly created security policy is a collection of 10 C/C++ secure coding standards, with example applications of the 10 core security principles such as least privilege, defense in depth, and fail securely. They also contain encryption policies for data at rest, in flight, and in use with Triple A controls and suggested automated static code analysis using tools like Cppcheck, Clang-Tidy, SonarQube, Fortify, and Valgrind. Lastly, it contains detailed audit and management policies to track user actions, access changes, and system modifications. |
| **3** | This is the threat security matrix, displaying the risk of errors amongst each of the 10 coding standards. As we can see, not all errors amongst the coding standards are likely to occur. However, the priority of all but the 6th and 7th are without exception rated High and above. Mistakes amongst the prioritised coding standards can lead to considerable damage to the company and any software we create. |
| **4** | As we can see here, these are the ten secure coding principles, and under them are the coding standards that apply. While there are many principles and standards, the most important is the tenth, which is to fix issues correctly by strictly following this security policy. |
| **5** | I ranked vulnerabilities based on four factors: Severity, Likelihood, Remediation Cost, and Priority.  Severity reflects how damaging a vulnerability would be if exploited. For example, SQL injection (STD-004-CPP) received the highest severity because it could expose or compromise entire databases.  Likelihood estimates how likely the issue is to occur in real-world development. String correctness issues (STD-003-CPP) and buffer overflows (STD-005-CPP) are very likely, especially in C++, where manual memory management is common.  Remediation Cost considers how difficult or time-consuming it would be to fix the issue. For example, smart pointer usage (STD-008-CPP) requires some refactoring but is not as costly as redesigning unsafe input validation.  Priority is based on the combination of the previous factors. High severity and high likelihood issues like SQL injection, string correctness, and buffer overflows were assigned the highest priority because they pose an immediate and serious risk.  This ranking system allowed me to focus on vulnerabilities that are both severe and likely, ensuring the highest risks are addressed first while also planning long-term improvements for lower-priority issues. |
| **6** | In this slide, we can see an explanation as to how exactly the policies for encryption in flight, at rest, and in use are implemented in my security policy.  …Continue reading the explanation in slide. |
| **7** | This slide is similar to the previous one. In this slide, we can see an explanation as to how Triple A works in my security policy.  …Continue reading the explanation in slide. |
| **8 - 11** | Here I conducted several unit tests demonstrating proper positive tests on C++ Vector Memory Safety. (Show each slide and don't say anything else.) |
| **12** | (Nothing to say here) |
| **13** | ..Read from slide. |
| **14** | …Read from slide. |
| **15** | …Read from slide. |
| **16** | …Read from slide. |
| **17** | …Read from slide. |