# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Intro |
| **2** | The security policy for GreenPace is an essential tool to ensure the ongoing protection of our applications and data from threats and vulnerabilities.  With our multi-layered approach to security, implementation of standards and tools, and multiple levels of redundancy in our implementation, we are exemplifying the defense-in-depth best practice. |
| **3** | 1. This Threats Matrix visualizes the various security risks our system can face and their likeliness and priority level.    1. Likely/High Priority       1. SQL Injections – When user inputs are not properly sanitized SQL injection attacks can occur leaving a database vulnerable to unauthorized access or manipulation.       2. Noncompliant Data Type Usage - Use of improper data types can cause unexpected program behavior and potential vulnerabilities.       3. Memory Protection Violation - Improper memory management can lead to vulnerabilities such as buffer overflows, memory leaks, and double frees.    2. Unlikely/High Priority       1. Data Value Misuse - Incorrect or unsafe handling of data values can cause unexpected behavior or security vulnerabilities.       2. Implicit Type Conversion – Using improper data types can lead to loss of precision and other unpredictable program behavior.    3. Likely/Low Priority       1. String Misuse - Misuse of string operations can lead to vulnerabilities such as buffer overflows, and other unexpected behavior.       2. Function Complexity Excess - Overcomplicated functions can become sources of bugs and their complexity can make it difficult to locate and understand where bugs might be occurring.       3. Incorrect Assertion Usage – If improperly implemented, code errors may be missed leading to unexpected behavior.    4. Unlikely/Low Priority       1. Improper Exception Handling – Poor exception handling can leave programs unstable and can also lead to unexpected behavior.       2. Loop Variable Scope Misuse – Variables with an improper scope can lead to accidental use of values outside of their intended use, leading to vulnerabilities. |
| **4** | Here we have the 10 Principles and the coding standards associated with each principle. |
| **5** | * Level 5   + Memory Protection   + SQL Injection   + If not properly adhered to, these issues can allow for unauthorized access and control of the system and its sensitive information * Level 4   + Data Type   + Data Value   + Function Complexity   + Implicit Type Conversion   + If not properly adhered to, these issues can result in system errors, unpredictable behavior, and security vulnerabilities. * Level 3   + String Correctness   + Assertions   + Exceptions   + Loop Variable Scope   + If not properly adhered to, these issues can lead to more minor bugs, and less severe security vulnerabilities. |
| **6** | Encryption-at-rest - Requires that all sensitive data stored must be encrypted using algorithms that are sufficient to protect the data from attackers and to meet compliance standards. Encryption should take place before the data’s storage on any device.  Encryption-in-flight - Requires that data being transmitted over a network be encrypted using algorithms that are sufficient to protect the data from attackers and to meet compliance standards.  Encryption-in-use - Requires that sensitive information be encrypted in memory until needed and then encrypted again once it is no longer in immediate use. |
| **7** | * Authentication   + Verification of users’ credentials prior to accessing a system   + Account management can reduce the risk of compromised accounts. * Authorization   + Differentiation of permission and access levels   + Principle of least privilege   + Default Deny * Accounting   + Logging of all user actions within a system   + Helps to monitor and identify when unauthorized access occurs and helps to mitigate and identify the effects of unauthorized access. |
| **8,9,10** | Using unit tests, we can help to ensure vulnerabilities are properly managed.  For instance, in the case of input sanitization, we can implement different unit tests to ensure that different inputs are handled properly.  This code snippet shows a unit test that tests if a function handles an empty input properly.  This code snippet shows a unit test that tests if a function handles an input containing alphanumeric characters properly.  This code snippet shows a unit test that tests if a function handles special characters properly.  Together these tests and others can be applied across multiple functions that take user inputs, ensuring that the functions have been made to properly handle any input. |
| **11** | Different tools and processes can be implemented through the pre-production and production cycle to help enforce our security standards and mitigate any vulnerabilities.  Starting with the Assess and plan stage, we can relay to our team the set standards and tools we have chosen to help withhold the standards.  During the design stage, we must actively utilize static code analyzers such ass Cppcheck and SonarLint to enforce our standards.  During the Building stage, we can again utilize tools like SonarLint to ensure the source code complies with our standards.  During the Verification and Testing stage, we will utilize automated unit testing to ensure the code does not have any unexpected behaviors, memory leaks, improperly handled inputs, etc.  Any issues that arise during the Transition and Health Check stage or the Monitor and Detect stage that cannot be directly addressed in the Respond stage will then be addressed in the Assess and Plan stage to address any changes in standards or tools that might be needed to address those issues. |
| **12** | The tools we will be using to help enforce our standards are SonarLint and Cppcheck.  SonarLint and Cppcheck are both static analysis tools.  While Cppcheck is a standalone code analyzer, SonarLint is an addon for Visual Studio, which will actively analyze your code as you write it.  Both are essential tools, as different code analyzers can provide different results or interpretations in their analysis. |
| **13** | Along with any new policy come benefits and risks. The benefits of implementation of the security standards include:   * Having a set standard is necessary to maintain consistency and continuity. * If developers meet the standards, the workflow will transition quicker from the development stage. * Users will trust the system knowing that security standards have been implemented   The risks include:   * Because of the ever-changing nature of software design, there is always a risk that the security standards are insufficient. With adaptation, this risk can be mitigated. * Implementation of any additional standards can lead to increased design, development, and testing stages. |
| **14** | With advancements in computing, many encryption algorithms currently in use are quickly approaching obsolescence.  With this in mind, I recommend implementing additional coding standards to address encryption security. |
| **15** | A standard should be implemented that enforces the use of a standard encryption algorithm for locally stored data; this algorithm should be complex enough to ensure security while allowing for relatively quick access from the system.  Another standard should be implemented that enforces the use of a standard encryption algorithm for data shared over a network; this algorithm should be more complex as the information is more susceptible online. |
| **16** | [Insert text.] |