# CS 405 Project Two Script Template

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

Briana Carlson

04/14/2023

[**https://youtu.be/lakXCJC0cU8**](https://youtu.be/lakXCJC0cU8)

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is Briana Carlson and in this presentation I will be explaining the security policy for Green Pace. |
| **2** | The Green Pace security policy is designed to prevent potential security vulnerabilities in code development and system architecture to keep sensitive data safe and prevent breaches.  It supports the defense-in-depth approach by deploying multiple layers of security controls to provide a layered defense.  The policy outlines security measures, principles, and best practices for all team members to follow and includes a threat matrix to identify potential threats from coding vulnerabilities. |
| **3** | This threat matrix illustrates the likelihood and priority of each of the 10 coding standards outlined in the security policy. It reveals that the majority of the standards are highly likely to occur and have a correspondingly high priority during development. |
| **4** | Not all of these principles have a standard that directly applies to them, but all are equally important.  Here you can see the coding standards that did directly apply to one of the ten principles that are  1. Validate Input Data  2. Heed Compiler Warnings  3. Architect and Design for Security Policies  4. Keep It Simple  5. Default Deny  6. Adhere to the Principle of Least Privilege  7. Sanitize Data Sent to Other Systems  8. Practice Defense in Depth  9. Use Effective Quality Assurance Techniques  10. Adopt a Secure Coding Standard |
| **5** | I ranked the standards based on the severity, likelihood, remediation cost, priority level, and their potential impact on the code quality. I started by identifying the standards with the highest severity and likelihood, as well as the standards with the highest priority levels. I then considered the remediation cost of each standard and the potential impact of addressing them on the code quality. Standards with a high impact on the code quality and moderate remediation costs were ranked higher than standards with low impact and high remediation costs. Based on this analysis, I ranked 4 and 5 as the most important standards to address, followed by 2, 3, 9, 10, 7, 8, 1, and 6 as the least important. |
| **6** | The three types of Encryption policies  Encryption at Rest:  Encrypts data while stored on a storage device to prevent unauthorized access. It's crucial for data security and privacy, especially on vulnerable devices. It's widely used in various settings to protect sensitive information.  Encryption in Use:  All data, including confidential information, must be encrypted in real-time within a secure and isolated area of the computer's processor. This policy is important in environments where multiple users access and process data, such as cloud computing and virtualization. Enforcing this policy minimizes data breach risks and ensures data security and privacy.  Encryption in Flight:  Encrypting data during transmission to prevent interception or tampering. Data is encrypted before sending and decrypted upon arrival to prevent interception or tampering. Enforcing encryption in flight ensures data security and prevents data breaches. |
| **7** | The triple-A policies are a set of security policies that aim to ensure the confidentiality, integrity, and availability of data. These policies include Authentication, Authorization, and Accounting.  Authentication verifies the identity of users and devices attempting to access the system.  Authorization determines what level of access each user or device should have once they are authenticated.  Accounting tracks the activities of users and devices and ensures that all actions are logged and audited.  Together, these policies help ensure that data is protected from unauthorized access, modification, or destruction, and that actions taken by users and devices are traceable and accountable. |
| **8** | This unit test checks if the reserve() function of a collection class correctly reserves space for the specified number of elements and does not modify existing elements. It verifies this by initializing an empty collection, adding 5 entries, calling the reserve() function to reserve space for 20 elements, and checking that the size is still 5 and the capacity is 20. |
| **9** | This unit test verifies that an exception is thrown when attempting to access an out-of-range element in the collection. It initializes the collection by clearing it and checking that it is empty and of size 0, and then checks that an exception is thrown when trying to access the element at index 1 using the at() function. If the exception is thrown successfully, the test will be considered a positive test. |
| **10** | This unit test checks the functionality of the at() function in the collection class. It initializes the collection by clearing it, checking that it is empty and of size 0, adds 5 entries using add\_entries(), adds a new element with a value of 4 to the end of the collection using push\_back(), and then checks if the element at index 5 is not equal to 7 using ASSERT\_FALSE(), and if the element at index 5 is equal to 4 using ASSERT\_TRUE(). The test is positive if both assertions pass, confirming that the at() function of the collection class is functioning correctly. |
| **11** | This unit test verifies whether the resize() function of a collection class throws an exception when called with the argument of the maximum size. It checks if an exception is thrown by calling the resize() function on the collection with the argument of the maximum size and using ASSERT\_ANY\_THROW(). If the assertion passes successfully, the test confirms that the resize() function throws an exception as intended. |
| **12** | When writing code, you should utilize automated testing to create a program with the main goal to protect the project from mistakes and warnings. We should use automated testing to check the programmer's work. Code snippets and project areas can be automatically checked by a software with minimal human input thanks to the DevOps process automation. By reviewing and testing the code before it is permitted to be pushed into the main branch, the automated process will safeguard the project's main branch. These tests shield the project from faulty and insufficient pieces of code that can endanger the main project's security. Each team member's developed code can be subjected to the same testing and standards thanks to an automation software. |
| **13** | The DevSecOps pipeline is an approach to software development that integrates security practices throughout the entire development process, from design to deployment. This ensures that security is a top priority and reduces the risk of vulnerabilities being introduced into the final product.  Some tools listed within the security policy for different coding standards  Astree is a static code analyzer that is specifically designed for safety-critical software written or generated in C or C++. Its main purpose is to verify the absence of runtime errors and invalid concurrent behavior in such software, ensuring that they meet the highest standards of safety and reliability.  Parasoft C/C++ test is a comprehensive and integrated software testing tool designed specifically for industries dealing with embedded safety-critical systems.  CodeSonar, developed by GrammaTech, is a powerful static code analysis tool that offers cutting-edge SAST (Static Application Security Testing) solutions for enterprise software security teams. Its main purpose is to identify and rectify defects and security loopholes in the source code, making it an essential tool for ensuring the robustness and security of software applications. |
| **14** | In assessing the risks and benefits of taking action versus waiting, it is crucial to consider the potential impact on costs. Delaying action may result in certain problems becoming more expensive to address in the future. Proactively preparing for potential attacks by establishing a defense in depth strategy is more advisable. It is essential to be preventative and prepared instead of reacting to security threats after the fact.  While developing a security plan may require additional time and resources, it is a wise investment that can save an organization significant costs in the long run. Neglecting to prioritize security measures until after a breach occurs can be much more expensive and detrimental, not only financially but also in terms of damage to a company's reputation and public trust. Therefore, it is essential to prioritize a defense in depth approach and prioritize security measures to prevent costly mistakes down the road. |
| **15** | The security policy may have gaps that require further explanation of cloud security, as it tends to focus more on code security aspects. Additionally, the policy only briefly covers the layers of defense in depth, and more detail is needed for a complete understanding.  It is important to note that the security policy is not an exhaustive list of all vulnerabilities and may need to be updated as new vulnerabilities and best practices emerge over time.  The security policy provides limited information on DevSecOps and could benefit from additional explanation to help individuals gain a more in-depth understanding of how it works. |
| **16** | To prevent future problems, it is crucial to incorporate security and defense in depth at every step of the software development life cycle. Best practices and security principles should be followed, and security measures should be continually updated and evolved, as security is not a static concept and is always changing.  Regular testing is necessary when developing software to ensure that no bugs or vulnerabilities make it to production. Both manual and automatic analysis tools can be used to assist with testing. |