# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Welcome to our presentation of the new Green Pace Security Policy. This policy includes policies, standards, principles, and best practices to prevent security vulnerabilities in the Green Pace code development and systems architecture. |
| **2** | This security policy is needed to identify and correct coding vulnerabilities, maximize automation to ensure compliance and keep costs down, and to work through these issues collaboratively. As security threats multiply, it is important to implement secure coding standards to counteract threats. The security policy will provide a guide for us to implement automation, best practices, and continuous testing. These principles, best practices, and industry standards will support our overarching Triple-A security framework that uses a defense-in-depth best practice as its foundation  Defense in depth is the practice of adding multiple layers of defense for a network or system to protect it from malicious attackers. The hope is that if a hacker is able to get through one layer, they will be thwarted by the next layer and unable to get through the various methods of protection. |
| **3** | Each standard is assigned a likelihood, which shows how likely it is that a flaw introduced by ignoring the rule can lead to an exploitable vulnerability.  Each standard also has an assigned priority. The priority level is assigned using a metric based on Failure Mode, Effects, and Criticality Analysis. Generally, standards at a level 1 should be addressed first, then level 2, then level 3. |
| **4** | 1. **Validate Input:** Input validation is a check applied to input that makes sure the information entered matches the expected format. This prevents unexpected data from being entered as input that could be used maliciously, for example as a SQL injection attack. 2. **Heed Compiler Warnings:** Compiler warnings can show issues with code that would otherwise be overlooked. Set the compiler to a high warning level setting and don’t ignore warnings as they come. 3. **Architect and Design for Security Policies:** Security is important to each level of development and should be factored into decision-making from the very start of the project. Best practices in programming and coding include utilizing security methods and keeping it top-of-mind every step of the way. This should also include protocols set ahead of time for any deviations from security policy. 4. **Keep It Simple:** Clean, concise code is easier to protect because it leaves less room for errors, so developers should make sure not to overthink their code and make it harder to secure. 5. **Default Deny:** Default deny means to deny access to everything that has not been given access ahead of time. Protocols will be put in place beforehand to grant access. 6. **Adhere to the Principle of Least Privilege:** Least privilege means that users are only granted access to the things they need to have. This ensures that a compromised account has a limited amount of potential damage. 7. **Sanitize Data Sent to Other Systems:** Data sanitization is the process of disposing of old data that is no longer needed. You should only keep what you need to have, and delete data in a timely manner once it is no longer useful (this reduces the risk of hackers gaining access to sensitive data, such as credit card information). 8. **Practice Defense in Depth:** As mentioned in the first slide, defense in depth is the practice of adding multiple layers of defense for a network or system to protect it from malicious attackers. The hope is that if a hacker is able to get through one layer, they will be thwarted by the next layer and unable to get through the various methods of protection. 9. **Use Effective Quality Assurance Techniques:** QA techniques help identify vulnerabilities and weaknesses to help strengthen the security of code. This may involve acting on compiler warnings, using assertions, and penetration testing (black box, white box, gray box). 10. **Adopt a Secure Coding Standard:** Adopting a secure coding standard helps ensure that all developers are taking the necessary precautions to protect their code. For example, each programming language has its own set of security techniques to implement, but there are also practices that apply to any language that a programmer should use. Having a set standard helps guide the developer’s choices and to follow best coding practices throughout development. |
| **5** | These coding standards are used to recognize coding vulnerabilities, create standards, and ensure policy compliance for coding within Green Pace. They are prioritized here in order of severity, because the severity demonstrates how serious the consequences are:   1. **Data Value:** Do not declare or define a reserved identifier. *(low severity)* 2. **Exceptions:** Handle all exceptions. *(low severity)* 3. **Assertions:** Use a static assertion to test the value of a constant expression. *(low severity)* 4. **Declarations and Initialization:** Do not write syntactically ambiguous declarations. *(low severity)* 5. **Data Type:** Do not cast to an out-of-range enumeration value. *(medium severity)* 6. **String Correctness:** Guarantee that storage for strings has sufficient space for character data and the null terminator. *(high severity)* 7. **SQL Injection:** Sanitize data passed to complex subsystems. *(high severity)* 8. **Memory Protection:** Do not access freed memory. *(high severity)* 9. **Characters and Strings:** Use valid references, pointers, and iterators to reference elements of a basic\_string. *(high severity)* 10. **Memory Management:** Detect and handle memory allocation errors. *(high severity)* |
| **6** | * **Encryption at rest**: Encryption in rest is essentially encrypting stored data. When data is “in rest”, that means it is not being actively used and is not moving between devices or networks. This data should be encrypted when put into storage and decrypted upon use with an encryption key. This ensures that if stored data is obtained by a hacker, they will not be able to decrypt it without the key. * **Encryption at flight**: Encryption at flight protects data that is moving from one location to another between devices, services, or across a network or the internet. The method of transmission will change how the encryption at flight is implemented. For example, if you are sending an email with sensitive data, you could encrypt it before sending or send it securely through a service that requires recipient authentication. * **Encryption in use**: Encryption in use ensures that data is secured while being used. It can use protected memory or transform the data for use. One example of this is if you are entered a password and it is hashed. |
| **7** | Triple A Policies:   * **Authentication**: Authentication is a way of determining that the user is who they say they are and that they have legitimate access to the system. Having multiple levels of authentication increases security to ensure that hackers can’t easily gain access to a user’s account. * **Authorization**: Authorization for certain access to a network and resources should only be granted as-needed. This is good practice for any type of software or application that a company uses because it protects sensitive information by giving access only to permitted users. * **Accounting**: In AAA security, Accounting is like an audit trail to show who did what in the system and when they did it. This is useful for security purposes, because if a hacker is breaching the system they can hopefully get an idea of how they got in and whose credentials they’re using. |
| **8** | Testers should understand how a hacker might infiltrate the system. They can work with developers to identify weak areas that could more easily be attacked and secure those areas.  One way to ensure that security is prioritized early on is to make use of penetration testing. You can utilize black box, white box, or gray box for this type of test.  Unit testing allows you to test small blocks of code, which helps identify potential vulnerabilities and to keep code free of bugs. When adding additional features it helps to check and make sure it’s still functioning as intended, which reduces the amount of time the developer spends debugging. Testing is done early and often throughout all of the stages of the software development life cycle. Early testing is crucial to ensure thorough testing that is not rushed at the end to meet deadlines. Testing can be done as soon as a requirement document is available, which allows errors and defects to be corrected before they move onto the next stage. |
| **9** | **Black Box Testing:**  A “black box” is a device where you can see the inputs and outputs, but not the internal workings. Black box testing, therefore, involves submitting various inputs to view the outputs and make sure that there aren’t any bugs or unwanted responses. This is a good way to view the system from a hacker’s point of view, since they may not have access to the inner workings of the system. |
| **10** | **White Box Testing:**  White box testing involves analyzing the system on a deeper level; you can test the individual components in the source code. This also allows you to identify inefficiencies and unneeded code. |
| **11** | **Gray Box Testing:**  Gray box testing is a mix of the black and white box tests. In this type of test, the tester has a partial/limited amount of knowledge of the internal system, and can therefore see what is happening to create certain outputs through the user interface. This is designed to take the best out of the black and white box testing and merge it into one. |
| **12** | Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Maximizing automation ensures compliance and keeps costs down.  Automating security practices throughout each part of the process is more efficient and will ensure that human error doesn’t creep in. With more and more technologies available to consumers today, software testing has become complex and cumbersome for manual testers. Manual testing is prone to human error and extremely time-consuming for QA teams. These factors combined with high market demand for fast product delivery make automated testing vital for success. There will always be a place for some manual testing, but there is no need for testers to complete duplicate steps that could be handled by automated testing. This means that the role of a quality engineer has shifted as automated testing is handled by developers, and they are able to focus more on the system as a whole with things like integration testing.  Not only is it important to automate verification and testing, but the monitoring and detection will also improve efficiency and yield less errors (as long as warnings are being acted on). It makes sense to integrate the detection tools listed under each standard into the workflow to ensure that those tools are being utilized and producing the best code possible under each standard. |
| **13** | The DevOps Pipeline is a pipeline that integrates security practices into the Software Development Life Cycle (SDLC).  There are many external tools listed in the Security Policy to ensure compliance with each standard. These tools include, but are not limited to:   * Astrée * Axivion Bauhaus Suite * Clang * CodeSonar * Helix QAC * LDRA tool suite * Parasoft C/C++test * Polyspace Bug Finder |
| **14** | The risks of insecure software includes:   * A hit to the company’s reputation, which could cost the company important revenue * Exceeding the company or customer’s budget * Spending too much time on a project, which inevitably costs the company more money   The benefits of secure software:   * Earning customer’s trust * Staying within budget, which keeps the customer happy and continues to build trust * Staying within the allotted timeframe, which again keeps the customer happy, along with allowing the company to continue taking on projects and increasing revenue |
| **15** | It is critical that everyone remains in sync with the principles and best practices in this Security Policy. This document applies to all staff that create, deploy, or support custom software at Green Pace.  Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.  Not only should developers adhere to the principles and standards in the security policy, but they should make use of unit testing to catch vulnerabilities and weaknesses in their code early and often. |
| **16** | In conclusion, adhering to the following items in the security policy will ensure successful project completion:   * Defense in Depth * Risk Assessment and prioritization * Coding Standards * Coding Principles * Tools * Trip A * Encryption * Unit Testing * Automation |
| **17** | Reference List |