# CS 405 Project Two Script Template

**Luke Kundinger**

**12/12/24**

**https://youtu.be/RvjRDDjeJGQ**

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | This is the Green Pace security policy by Luke Kundinger |
| **2** | Green Pace will be using the defense in depth strategy to ensure there are many layers of security. This makes it more difficult for attackers to breach the system. Below is a chart showing the layers of security that will be used. If one layer has been overtaken, there will be many more to keep attackers out. |
| **3** | Here is the threats matrix table with coding standards listed as likely, unlikely, priority, and low priority. Likely coding standards include INT-036-C, INT-033-C, STR-030-C, STR-002-C, MEM-051-CPP, ERR-051-CPP, DCL-030-C, ARR-038-C, and ENV-032-C. Unlikely coding standards include DCL-003-C. Priority coding standards include STR-002-C, MEM-051-CPP, ARR-038-C, ENV-032-C, andSTR-030-C. Low priority coding standards include INT-033-C, DCL-030-C, ERR-051-CPP, INT-036-C, and DCL-003-C. |
| **4** | Here are the 10 coding principles.  Principle 1: Validate Input Data.  Validating input data involves checking input from users for functionality and security purposes. It is used to make sure the input being entered meets what the program is looking for and that it is not malicious. Incorrect input length or type can lead to buffer overflows and injection attacks.  Principle 2: Heed Compiler Warnings.  Heeding compiler warnings involves paying attention to warnings given by the compiler. These warnings will flag vulnerabilities and other issues in the code. Noticing and fixing these warnings can help keep the code sure and functioning correctly.  Principle 3: Architect and Design for Security Policies.  Architect and design for security policies involve taking security into consideration during the early stages of the SDLC. This means addressing potential security vulnerabilities and developing to prevent them.  Principle 4: Keep it simple.  Keeping it simple means developing code to be as simple as possible. This is helpful for reducing errors and vulnerabilities that can be created with more complex programs.  Principle 5: Default deny.  Default deny means allowing user access to specific data and denying everything else by default. This can prevent users from accessing sensitive data that is meant to be secure.  Principle 6: Adhere to the principle of least privilege.  The principle of least privilege means the user should only have access to necessary information. Only giving the user access to what is needed reduces the risk of unauthorized actions.  Principle 7: Sanitize data sent to other systems.  Sanitizing data sent to other systems makes sure the data is safe and sent correctly. This involves cleaning the data by removing unwanted characters and formatting it correctly.  Principle 8: Practice defense in depth.  Defense in depth means implementing multiple layers of security into the program. Each layer strengthens security as there is more for an attacker to get through.  Principle 9: Use effective quality assurance techniques.  Using effective quality assurance techniques involves regularly testing the program for vulnerabilities. It is important to discover potential vulnerabilities early in the SDLC for better efficiency. Regular security testing can help discover security risks so they can be fixed right away.  Principle 10: Adopt a secure coding standard.  Adopting a secure coding standard means following security guidelines while developing code. It can help a development team ensure they are working toward the same security goals as they have a guideline to follow. |
| **5** | Here are the coding standards.  Converting a pointer to integer or integer to pointer. Converting pointer to integers or vice versa incorrectly can lead to incorrect results and undefined behavior.  Ensure that division and remainder operations do not result in divide-by-zero errors. Divide-by-zero errors can lead to unexpected behavior and integer overflow.  Do not attempt to modify string literals. Modifying string literals often results in an access violation because they are stored in read-only memory.  Sanitize data passed to complex subsystems. String data may contain special characters that can lead to malicious behavior.  Properly deallocate dynamically allocated resources. Deallocating resources that are not allocated dynamically can result in undefined behavior.  Use a static assertion to test the value of a static expression. Static assertions can be used to find incorrect assumptions during compiling.  Handle all exceptions. All exceptions thrown must be caught by a matching exception handler. Not doing so can cause the program to terminate or keep the exception uncaught.  Declare objects with appropriate storage durations. Objects have storage durations and attempting to access them outside of their duration can lead to unexpected behavior.  Guarantee that library functions do not form invalid pointers. Invalid pointers can be caused by using an incorrect element count and lead to pointers that do not point or unexpected behavior.  All exit handlers must return normally. Exit handlers must terminate by returning. If an exit handler has nowhere to return to or returns to the wrong place, unexpected behavior can occur. |
| **6** | Here are the Encryption policies for in flight, at rest, and in use.  Encryption at rest is used to protect data that is being stored and not in use. The data is encrypted and is not readable until it is decrypted with a key. It is used to prevent attackers from accessing sensitive information.  Encryption in flight is used to protect data when it is being transferred from one place to another. It uses encryption methods to keep the data safe if it is interception during the transfer. The data stays encrypted until it needs to be accessed as intended.  Encryption in use is used to protect data while it is being used. It ensures that the data is encrypted at all times until completely necessary. This prevents unwanted access to the data while it is being used. |
| **7** | The triple-a policies are for authentication, authorization, and accounting.  Authentication is used to verify users and make sure they are who they say they are. To do this, usernames and passwords are used along with more secure methods like multi-factor verification. This prevents attackers from logging into users' profiles and accessing personal information.  Authorization grants access to users based on who they are and what they are looking to access in the system. Full access can be given to administrators when changes need to be made. The default deny principle should be applied to give users access to only what they need to keep the program secure.  Accounting monitors changes in the system and records who made the changes and information regarding the changes. It is used to track activity in the system. It can also be used as proof when an attack may occur and find who, when, and what was done. |
| **8** | Unit testing will be done during the development stage regularly. After each block of code is completed a unit test can be done to ensure it meets the requirements. The next two slides will show an example of a positive and a negative unit test. |
| **9** | Here is a positive unit test example. It makes sure that the program functions as intended. In this example, the function erases the collection. The unit test ensures that the collection has been properly erased. |
| **10** | Here is a negative unit test example. This makes sure that an error can be thrown when given bad data. In this example, the input expects a positive number. When given a negative number a length error is thrown. |
| **11** | The existing DevOps structure is a great starting point for automation. During the pre-production stage automation tools can be used to conduct static code analysis, unit testing, and vulnerability testing. Static code analysis and unit testing should be done often during development. Vulnerability testing can be done during development but also should be done in the production stage. Additional automation can be added in the production stage like threat alerts, continuous integration, and accounting. Tools that monitor attempted attacks can notify the company as soon as they happen. Continuous integration tools can be used to instantly update the system when development changes have been made. Accounting can be done with tools that record information regarding changes that have been made to the system. All of these tools can be added to the existing DevOps structure to support the development and longevity of the program. |
| **12** | Integrating security into each stage of the DevOps pipeline will make the process more efficient and cost effective. There will be no backtracking, and the process will be linear. If security measures are left until the end, it will take more time and resources to address them.  By considering security measures in the early stages of the software development lifecycle, there is a guideline for developers. There will be security requirements for developers to meet while creating the software. During development security measures will be tested to make sure security is up to standard. This process will prevent easy access to attackers and make sure everything has been covered. |
| **13** | It is recommended to prioritize security measures on authentication and accounting protocols. An example of why these protocols are important relates to the Myspace data breach of 2013. Attackers were able to access Myspace accounts with a simple username and password. Authentication techniques like multi-factor verification would provide an additional layer of security to confirm a user’s identity. The Myspace data breach went unnoticed until 2016. There were no accounting methods to notify Myspace of the breach. Accounting measures should be added to notify Green Pace of potential threats as soon as they happen so that changes can be made if necessary. |
| **14** | The most important standards that should be applied in this scenario are defense in depth and don’t leave security until the end. Defense in depth will add many layers of security for great protection. It will make things difficult for attackers and keep data secure. Security measures will be added to each stage of the DevOps process using DevSecOps. This will make security a priority throughout the entire process. The DevOps process will be very efficient with security planned out in the early stages and not saved until the end. |
| **15** | Here are my references. Thank you. |