# CS 405 Project Two Script

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is Jonathan Boeglin, and this is the security policy presentation for Green Pace. |
| **2** | For an effective use of Defense-in-Depth Green Pace needs to implement security measures consistently throughout the development process. Making these policies efficient and uniform, so that any employee can follow them, will make the process easier, more efficient, and stop any gaps in security caused by deviations. This policy will define the core security principles, coding standards, triple-a standards, and encryption standards needed to create an effective barrier. |
| **3** | Using the threats matrix, we can determine the order in which to address vulnerabilities. Priority threats are those that are likely to happen, will cause a noticeable amount of damage, and require more resources to fix than others. Likely threats are more probable to happen but aren’t as dangerous or cost less to fix. Unlikely threats are less probable and not as severe. Finally, Low priority threats are unlikely, not as severe, or do not take a lot of resources to fix.  Automated code checkers can be used to help identify these threats and prioritize them. |
| **4** | The ten principles help to develop a security policy by making sure the standards being followed cover the basic areas of security. As you can see, not all areas are covered by the standards, and therefore it is good policy to also keep the principles themselves in mind when securely coding. |
| **5** | The ten coding standards that Green Pace needs to follow are listed here in priority order. As explained before, priority is given to the standards that will do the most damage and are the most likely to happen. On the bottom of the list is the least likely to happen, with the standards that need to be monitored but are not top concern landing in the middle of the list. |
| **6** | Encryption in Rest  Referring to the protection of data that is being stored or is not actively being used, this applies as sensitive data needs safe storage. Varying levels of security need to be implemented to protect data based on its volatility.  Encryption in Flight  This level of protection is for data that is currently being sent between destinations, such as emails, various app messages, or other communication done through public means. This applies to security for possible mishandling of data, such as data breaches or leaks.  Encryption in Use  Encryption in use is the most important, as the data it protects is the most likely to be tampered with. It is used for protection while data is actively being accessed and manipulated, such as reading, writing to, or creating a file. An application example of this policy would be protection from attempts to steal user information, like passwords. |
| **7** | Authentication  This is the process of a user giving details on themselves to be applied in the future for verification. Authentication is used to confirm people accessing a program are genuine with user logins. The addition of new users is also handled by authentication of details provided, such as email addresses by way of verification emails.  Authorization  Authorization protects data from the wrong people by user access levels. This allows users to access different parts of a network or program depending on the privileges given to them and prevents them from seeing the rest. An example of the application of this policy would be the restriction of accessing and making changes to a website’s database. Only those with the proper credentials would be allowed to do so, preventing unwanted tampering by those that have access to the site, but not the database itself.  Accounting  This part of the strategy helps prevent future issues by keeping a record of past events. This is done through recording of users through their data usage-how they used it, what files they accessed, and the length of time they were connected. This policy can be applied when a security breach occurs, as all users should be accounted for, and the offender can be determined from the access data gathered during the attack. |
| **8** | Next, in these unit tests coding vulnerability tests concerning containers will be shown alongside results. It is important to make sure that both positive and negative outcomes are correctly tried. |
| **9** | First up is the test to make sure adding an entry to an empty vector increases the collection’s size to one. Asserting the collection is empty, adding an entry, and then asserting that the collection is no longer empty confirms this to be accurate. |
| **10** | Next is the test of resizing a collection by a positive amount increases the collection’s size. When run correctly, the test checks the vector’s current size, resizes it by adding one to its size, and then asserts that the new size is exactly one larger than the previous. |
| **11** | For the first negative test, verifying that an Out-of-Range Exception will be thrown correctly is checked. When run, the test asserts that the collection is of a certain capacity, and that an OOR Exception will be thrown when attempting to access an entry that would fall just outside of that capacity. |
| **12** | For the next negative test, a Length Exception is checked to make sure its functionality is correct. This test asserts that the vector is of a certain capacity, then resizes the collection to zero and asserts that this is true as well. This makes sure that the collection capacity always equals zero for the actual test. Then, the test asserts that a Length Exception is thrown when the collection is resized to a negative amount. |
| **13** | This is a cumulative screenshot of all the tests running correctly and shows that each test can be taken a step further. For instance, the resize increase test was recreated to check that resizing also decreased the capacity of the vector properly. |
| **14** | Automation of the security implementation process consists of the Pre-Production and Production sections of the DevSecOps timeline. Switching from DevOps to DevSecOps requires the use of new tools and methods in both halves. |
| **15** | Automation of security practices can be inserted into the DevOps process during pre-production and production. Security needs to be planned for, built properly, tested, and maintained, so adding secure procedures to the planning, design, building, testing, and maintenance stages of the process are the best places for modification. This changes the production process from just maintenance and release of updates to also monitoring for threats and responding to any attacks. Maintenance includes stabilization after attacks, and rollout of updates include live testing of security measures.  On the pre-production side the steps stay mostly the same, with the addition of security planning, design, and integration during the appropriate times. Testing and verification will require the creation of new tests for security coding.  Regarding how automation tools will help enforce the standards within this policy, various tools listed within each section can be used to check the code for breaches of each of the specific standards. This will take place during the building and verification and testing phases of pre-production. In addition, any updates to the tools that catch new issues can be run during production on the code during maintenance and any necessary updates can be added to the cycle. |
| **16** | The risks of implementing a security policy such as this include the need for resources and proper timing. Not only will implementing secure code take more time and employee power, costs for maintenance and updating may increase noticeably. Also, implementing a policy without fully grasping the nature of the threats to Green Pace may cause gaps to be left in areas thought to be of low priority.  Benefits to this policy outweigh these risks, as having any type of secure practice is better than nothing. In addition, immediate implementation allows time to be saved that would be spent on refactoring for security later, and the coding process is always secure because of this continuous integration. Finally, certain security measures inherently make code more readable and concise. |
| **17** | While this is a decent start to a well thought out security policy, certain recommendations can be made for future additions. To start, the ten main coding standard rules are not completely covered. Several areas of C++ need to be addressed as only a few have been covered by the current ten that are implemented. Also, not all ten coding principles are covered by security standards. Making sure that the standards added from the missing rules also relate to the missing principles will help solidify the policy. Finally, the policy needs to be peer reviewed regularly, so that a bias is not created from the authors and editors only having expertise in security coding. |
| **18** | In conclusion, additional standards will help fill any gaps left in the policy by adoption from any missing rules. An example with a link is included below to the fact that iterator issues are becoming more prevalent in modern coding languages. This container standard is not covered in the current policy and should be added.  Following the ten principles, even without relevant standards, should also be commonplace as it will help prevent any gaps. Also, using compartmentalization correctly will make any security additions to the code easier down the line. |
| **19** | Information included in this presentation can be found at the following sites. Thank you for your time. |