# CS 405 Project Two Script

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hi everyone, my name is Michael Turco, and I will be presenting the new Green Pace Security Policy. This presentation will also cover risks and benefits of the policy, and some recommendations for further improvement. |
| **2** | The Security Policy is necessary to have clear guidelines and compliance with secure coding standards. This policy will define the core security principles, the C and C++ coding standards, the Triple-A standards, and data encryption standards. This policy applies to all staff that create, deploy, or support custom software at Green Pace. |
| **3** | This table shows our 10 coding standards, and how they map to priority level and likelihood. The highest priority standards are ‘STD-003-CPP’ and ‘STD-004-CPP’, which relate to reading uninitialized memory and SQL injections respectively. Some of the lower priority standards are related to unit testing and assertions, and don’t directly affect security as much compared to the other standards. |
| **4** | Here are our 10 principles for secure coding. By following these principles, we can avoid all vulnerabilities described in the 10 coding standards. The different standards are listed below each principle which they apply to. |
| **5** | And here are our 10 coding standards, with the more critical ones highlighted in blue. Adopting these 10 secure coding standards will help to prevent vulnerabilities from being created in code, and provides an easy way to get consistent and predictable results.  The critical coding standards include: SQL Injection, Exceptions, Memory Protection, and Input Output. These standards have the potential to cause a lot of damage if not followed correctly, and must always be taken into account. |
| **6** | Moving on to encryption policies, it is important to understand the three different types of encryption; Encryption at rest, in flight, and in use.  {read off all three types, word for word}  By encrypting data in these three stages, we can be sure that user data is protected every step along the way, and will stop attackers from being able to easily read sensitive data. |
| **7** | The next part of the security policy is the Triple-A framework, which stands for Authentication, Authorization, and Accounting.  {read off all three types, word for word}  These three policies will help to verify users identity, limit their access, and track their activity, which all play a crucial part in maintaining a secure system. |
| **8** | Unit Testing is a way to create small and isolated pieces of code that serve to test all different kinds of functionality that your system implements. Unit testing helps improve security by significantly reducing the amount of effort required to check that all systems are functioning as intended.  For example, this unit test here tests to see if adding an element to an empty vector will increase its size. This can seem minor or insignificant at first, but after hundreds of these tests are made, it will serve as a near instant way of making sure that no unknown bug was introduced into the system, or no feature was broken by a dependency update, and makes sure everything is still working as intended.  On the bottom of the slide, you can see that this test has passed, and took an insignificant amount of time to run. Unit tests can be run automatically with each build of a project, so you can be sure that everything is still functional, without spending hours each time to test everything out yourself. |
| **9** | This is another slide for Unit Tests, and this one shows the ‘Resizing Increases Size’ test. In the code, you can see that elements are added to a vector, then the size is asserted to equal a certain value. This is repeated for a few numbers, ranging from 0 to 10. |
| **10** | This unit test checks to make sure that the function call ‘pop\_back’ is decreasing the size of the vector by one each time it is called. You can see on the bottom that this is still taking 0 milliseconds, meaning these tests take an insignificant amount of time to run once they have been created, yet provide crucially valuable information if something were to go wrong. |
| **11** | Lastly, here is a negative unit test, which means it checks that a piece of code will throw an exception when run. This unit test tries to access an element of the collection first inside its normal range, then outside its normal range, and makes sure an exception is thrown when doing so. |
| **12** | This diagram shows a depiction of the DevSecOps cycle going around in an infinite loop. This is because DevSecOps is a continuous process, and security is integrated along each stage to make sure that the security evolves as the system does too. |
| **13** | When integrating security into the DevOps pipeline, it becomes DevSecOps. Some parts of integrating security involve threat modeling, vulnerability scanning, unit testing, health checks and monitoring, penetration testing, log collection, and determining a security baseline.  There are also some useful external tools to check C++ code for vulnerabilities or errors, which include CppCheck, Clang, Visual Studio Code, and GDB Online Debugger. |
| **14** | The strategy of integrating security into the development process is not a fool-proof plan, and doesn’t work for every project. Some negatives include a significant time commitment, longer times for development, and a higher short term cost.  However, for this project, the benefits of a stronger and more integrated security system, with multiple layers of defense and mitigation of vulnerabilities definitely outweighs the negatives. The initial cost to get these systems up and running outweighs the cost of potential damages that would be caused by a data breach or other attack. |
| **15** | Now, onto some future recommendations for new things to be added to this policy.  The security policy doesn’t mention anything about malware, firewalls, or insider attacks, and it would be helpful to document some procedures regarding ways to defend against these sorts of attacks.  There is also no mention of policy for using third party systems, plugins, software, or libraries, whose code we cannot as easily control.  Policy about audit logs would be important too, such as how long to store them for, what they should contain, and when they should be reviewed. |
| **16** | In conclusion, integrating security practices into development helps reduce long term costs and implementation problems compared to after the system is already in place.  The security policy helps establish a baseline and standard for the level of code that will be written, and helps ensure vulnerabilities are removed early on in development, as opposed to later on in the future, which increases the cost to remove them and potentially could interfere with the current system implementation.  Despite the higher starting cost and increased development time, DevSecOps leads to a much more scalable, automatable, and secure system compared to standard DevOps. It has long term benefits that greatly outweigh the negatives in the first few phases of development. |
| **17** | Thank you all for watching this slideshow about the new Green Pace Security Policy, I look forward to hearing your feedback, and answering any questions you may have! |