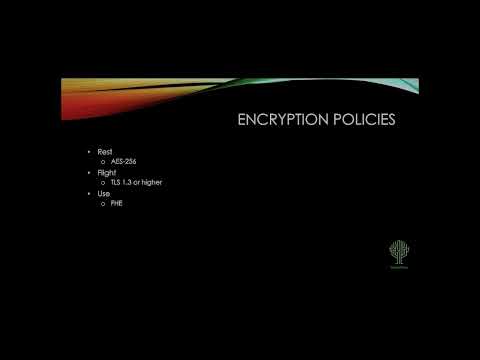
# CS 405 Project Two Script Template

Presentation link <https://youtu.be/J0KMIeJuCmE?si=fFmKk130lV79vf_S>

[](https://youtu.be/J0KMIeJuCmE?si=fFmKk130lV79vf_S)

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello my name is Trenton Mendiola and this is an overview of the Green Pace security policy |
| **2** | The security policy is needed as Green Pace is moving from a DevOps practice to a DevSecOps. The policy serves as a guide following it will ensure that each element of the system is following proper security protocol. This policy covers 10 coding standards that serve as rules and guides so that individuals follow each of the 10 coding principles. The policy also covers how to implement the three encryption polices, the tripe A polices, unit testing, and automaton. |
| **3** | This is the threats matrix which provides an overview of how likely each threat is to be taken advantage of. This ranges from unlikely to priority. Where unlikely is as it suggests unlikely to occur and should not take priority. While Priority indicates that it is likely to occur and can cause serios issues if not address so they should take priority. The vulnerabilities present in priority as stated by Carnegie Mellon University in the wiki “SEI CERT C++ Coding Standard” all pose threats that will compromise a system and use common tactics to do so like buffer flows which is why I have placed them in priority (Carnegie Mellon University, n.x). Where the threats in the likely category can all be taken advantage of through common exploits but pose less of a threat (Carnegie Mellon University, nd). Low priority may not be commonly taken advantage of and does not pose as serios of a threat (Carnegie Mellon University, n.d). Lastly Unlikely features threats that are just not likely to be taken advantage of although they should still be addressed (Carnegie Mellon University, nd). |
| **4** | The ten coding principles are Validate Input Data, Heed Compiler Warnings, Architect and Design for Security Policies, Keep It Simple, Default Deny, Adhere to the Principle of Least Privilege, Sanitize Data sent to Other Systems, Practice Defense in Depth, Use Effective Quality Assurance Techniques, and Adopt a Secure Coding Standard. The coding standards listed below some of the principles are the standards that apply to that principle. Validate input data is associated with coding standard 4 as this standard works to ensure user input is valid data. Heed Complier warnings is associated with coding standards 1, 2, 4, 7, and 9. Each of standards address issues that would present themselves as compiler warnings and by following the standard you are then following the principle. The principle Use Effective Quality Assurance Techniques is associated with coding standards 6, 8, 9, and 10. Each of these standards all work to apply quality assurance techniques into your code to following the principles guidance. Lastly the principle of adopt a secure coding standard is associated with coding standards 1, 2, 3, 5, and 10. Each of these coding standards enforces applying a set of rules or guidelines that dictate what should and should not be done throughout all of your code working towards the principle of applying a secure coding standard. |
| **5** | The coding standards are ranked in ascending order with the top being least priority and bottom being the highest priority.  The rankings are based on the potential risks that these vulnerabilities bring as stated by Carnegie Mellon University in the wiki “SEI CERT C++ Coding Standard” The list begins with INT35-CPP and ERR50-CPP vulnerabilities which are likely to result in inaccurate variables or a loss of data (Carnegie Mellon University, n.d) As the list moves down to ERR51 the vulnerabilities is likely to result in a leak of information from the system (Carnegie Mellon University, n.d) from there STR50, FIO30, MEM50, MEM52, MEM51, MEM56, ending with INT30 all of these vulnerabilities are likely to result in unexpected behavior that can either give the attack access to functionality of a system or information within it (Carnegie Mellon University, n.d) |
| **6** | Encryption at rest, flight, and use as stated by Google in the "Google Cloud Security White Pages" refer to applying encryption to data at different stages(Google, 2025) Rest refers to encrypting data when it is being stored in the system(Google, 2025) Flight refers to encrypting data when it is moving from one place to another. Lastly use refers to encrypting data when it is actively being used by users (Google, 2025). The encryption policy states that in order to apply encryption at rest an Advanced Encryption Standard or AES must be applied to the data that is stored in the system. AES-256 will be used because of its staple in the industry as a result of its effectiveness and large 256 bit key size. The encryption policy also states that in order to properly apply encryption in transit Transport Layer Security must be applied to the data that is moving from one place to another (Google, 2025) TLS 1.3 or higher must be used because as stated by Cloudflare in the article "What is the difference between TLS 1.3 and 1.2" it is the latest version offering more security than previous and it is also receiving current support (CloudFare, n.d). Lastly the encryption policy state that in order to properly apply encryption to data in use Fully Homomorphic Encryption or FHE must be applied to that data. This is because as stated by Andreja Velimirovic in the article "What Is Encryption in Use?" This encryption method applies directly to the data that is being used and while uses more resources to run provides more security than PHE which is partially Homomorphic encryption (Velimirovic, 2023). |
| **7** | The triple a framework is designed to authenticate who a user is provide the accurate authorization, and account for what users do in the system. The security policy states that in order to achieve proper authentication and authorization, the Role-based Access Control model must be implemented. This model as described by Gregg Lindemulder in the article "What is role-based access control (RBAC)?" sees that a system has multiple different roles that are created and assigned by admins, this ensures that certain access is only given to individuals that are pre-determined to have that access (Lindemulder, 2024). This model supports the authentication and authorization policies as it creates a system that requires both authorization to access content as users would have to get into their assigned account and provides the correct authorization to users (Lindemulder, 2024). To better enforce authentication the security policy also states to implement Password less Authentication which as described by John Martinez in the article "9 User Authentication Methods to Stay Secure in 2025" provides users with a way to access their account without a password through means such as email links (Martinez, 2025). This supports authentication as it enforces the login process to be even more personal doing away with the password which can be accidently given out or even guessed by attackers (Martines, 2025). To enforce accounting the security policy states that User Activity Monitoring or UAM will need to be implemented. UAM as stated by Fortra in the article "What is User Activity Monitoring? How It Works, Benefits, Best Practices, and More" is the implementation of active monitoring for what users are doing within a system (Fortra, n.d). This supports accounting as user activity is tracked and accounted for providing a easier way to keep account of any suspicious or unusual activity (Fortra, n.d). |
| **8** | Unit testing is an essential part in developing a system as you can use it verify that behavior that is supposed to happen in your system does and behavior that is supposed to be prevent is.  Here are five examples of unit testing that will guide how to implement unit testing into a system. |
| **9** | This is a positive test; it is important to remember when writing positive tests that you are verifying that expected behavior occurs with no errors. This tests for example, tests that adding values to the collection multiple times will result in the excepted number of entries at the end. It is important to verify that expected results are occurring throughout the test which is why when the start starts, we check to see if the collection is empty. From there we perform some logic adding 1 entry and verify that the entry was added. This is done by checking that the size is total to 1. After that we repeat 2 more times until the last assert statement that verifies the collection size is a total of 10 entries. |
| **10** | This is a negative test; it is important to remember when writing negative tests you are assuring that an error occurs when you perform actions that would cause those errors. In this test we are testing that using the resize method with an value greater than the max size will result in a length error being thrown. Notice again in this test we are asserting behavior throughout the test to make sure everything works properly. |
| **11** | This is another negative test that is asserting the out-of-range error will be called when calling at() while using an index that is out of bound. Again throughout the test we are testing functionality asserting the collections empty on start and that the exception is thrown at the end. |
| **12** | This is another positive test, in this test we are asserting that reserve increases capacity but not size. As you can see in the test we are asserting functionality throughout it checking that the size is empty at the beginning. Then at the end of the test that the collection capacity when up to 10 but size remained 0. |
| **13** | This is a visual representation of the DevSecOps pipeline |
| **14** | The DevSecOps pipeline is a representation of each of the processes involved in developing and maintaining a system. This diagram represents an infinite loop that matches the constantly moving and repeating phases involved in developing a system. There is a constant flow from pre-production to Production where each step progresses the system.    Automation lands within the verify and test phase of the pipeline occurring as the last step in this phase. During this process the Sonar static code analysis tool is used to perform static code analysis on the system. This as stated by Sonar in their product page titled "Sonar static code analysis" is performing a check that compares the code to their define best practice static code analysis rules (Sonar, n.d). After making the comparison the tool will then present where vulnerabilities are present and offer ways to correct them (Sonar, n.d). After receiving the results from the test fixes based on it will be implemented and the test will be run again. This processes will repeat until the tool does not detect any vulnerabilities. This process is ensuring the the system holds up to the standards established by the tool which align with best security practices in term guiding the code to be clean and secure (Sonar, n.d). |
| **15** | When applying security intrinsically there are several factors that must be considered and it must be applied correctly, with this being said the benefits of this will be worth the risk and work involved. To successfully implement this there is a lot of planning and follow through involved. As stated by The Federal Trade Commission in the article “Start with Security: A Guide for Business” when developing a project, you should start by establishing the security that will be implemented (Federal Trade Commision, 2024). This will create the excepted level and type of security that should be implemented throughout your project (Federal Trade Commission, 2024). From the start of the project to the end this established rule for security should be implemented. When implementing this strategy you run the risk of creating and implementing a security structure that is not adequate or does not work for your project. This is why it is important to do the hard work researching and creating a security policy like the one representing in this presentation. With a good security policy that supports and implements proper security like the principle of least privilege for example the risk involved of applying security intrinsically are diminished and you reap the rewards of having a system developed to support security. |
| **16** | One gap that is present in the security policy is a lack of SQL injection specific coding standards. While coding standard STD-004-CPP FIO30-C. Exclude user input from format strings. Does help to prevent SQL injections by working to sanitize user input data. This standard does not directly relate to SQL injections and a specific standard should be added to address this.    Another gap that is present in the security policy is more tool that should be used. Aside from automated testing there are many tools that can help implement policies like Authentication which can be used to more easily apply this principle.    The last gap that I can identify in the security policy is that there is not an updated graphic of the DevSecOps pipeline that represents the pipeline with automation included. This will need to added so individuals do not miss automation. |
| **17** | As stated by OWASP in the article “SQL Injection Prevention Cheat Sheet “ utilizing parametrized queries and proper storage procedures are an effective standard to reduce the risk of SQL injection vulnerabilities. Which is why I suggest adding STD-011-CPP Use Parametrized Queries and STD-012-CPP Use Proper Storage Procedures to the security policy to improve the gaps present in it which includes a lack of SQL injection specific standards. |
| **18** | Here are all of the resources that helped in the creation of this presentation |

References Carnegie Mellon University. (n.d.). SEI CERT C++ coding standard - SEI CERT C++ coding standard - Confluence. Retrieved June 20, 2025, from <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

Cloudflare. (n.d.). What is the difference between TLS 1.3 and TLS 1.2? Cloudflare - The Web Performance & Security Company | Cloudflare. Retrieved June 20, 2025, from <https://www.cloudflare.com/learning/ssl/why-use-tls-1.3/>

Federal Trade Commission. (2024, February 14). Start with security: A guide for business. Retrieved June 20, 2025, from <https://www.ftc.gov/business-guidance/resources/start-security-guide-business#start>

Forta. (n.d.). What is user activity monitoring? Enterprise IP and DLP Software | Digital Guardian. Retrieved June 20, 2025, from <https://www.digitalguardian.com/resources/knowledge-base/what-user-activity-monitoring-how-it-works-benefits-best-practices-and-more>

Google. (2024). Default encryption at rest. Google Cloud. Retrieved June 20, 2025, from <https://cloud.google.com/docs/security/encryption/default-encryption>

Google. (2025). Encryption in transit. Google Cloud. Retrieved June 20, 2025, from <https://cloud.google.com/docs/security/encryption-in-transit>

Lindemulder, G. (2024, August 20). What is role-based access control (RBAC)? IBM - United States. Retrieved June 20, 2025, from <https://www.ibm.com/think/topics/rbac>

Martinez, J. (2024, June 10). 9 user authentication methods to stay secure in 2024. StrongDM: Your Partner in Zero Trust Privileged Access. Retrieved June 20, 2025, from <https://www.strongdm.com/blog/authentication-methods>

Owasp. (n.d.). SQL injection prevention. Introduction - OWASP Cheat Sheet Series. Retrieved June 20, 2025, from <https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html>

Sonar. (n.d.). Sonarstatic code analysissince2008. Retrieved June 20, 2025, from <https://rules.sonarsource.com/>

Velimirovic, A. (2023, November 16). Data encryption in use explained. phoenixNAP Blog. <https://phoenixnap.com/blog/encryption-in-use>