# Andrew McPherson CS-405 12/11/2022

**https://youtu.be/yCUcn1hOfVE**

# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is Andrew McPherson and through this presentation I will be explaining the security policy for Green Pace and the different principles, standards, and other policies from within. |
| **2** | By using the Green Pace security policy, we can pave the way of ensuring proper defense in depth is implemented into any software or system being developed. This is done through the defined security principles and the coding standards within the security policy.  This security policy is needed because it brings many of the aspects needed to have a solid foundation for defense in depth, such as the Triple A policy (Authorization Authentication, Auditing) and even some data encryption standards. By having all these secure policies in one place, it can be used as the road map for truly secure software. |
| **3** | Within this threat matrix we can see where each of the 10 coding standards from the security policy fall in relation to likely to unlikely, and the priority they would take during development.  Most of the coding standards have a high likelihood of occurrence, and an even higher level of priority, those listed in bold being the highest of them all. Even those with lower priority should still be addressed. |
| **4** | Below are the 10 principles outlined in the security policy. Though some standards do not have a direct relation to a principle found in the security policy, they are all equally important for an overall secure software.   * 1 – Validate Input Data: This principle applies to STD-001, STD-002, STD-003, STD-004. * 2 – Heed Compiler Warnings: This principle applies to STD-001, STD-002, STD-003. * 3 – Architect and Design for Security Policies: This principle applies to STD-001, STD-004, STD-007, STD-009. * 4 – Keep it Simple: This principle generally applies to all standards. * 5 – Default Deny: Overall general principle * 6 – Adhere to Principle of Least Privilege: Overall general principle * 7 – Sanitize Data Sent to Other Systems: This principle applies to STD-003, STD-004, STD-008. * 8 – Practice Defense in Depth: This principle applies to STD-003, STD-004, but should generally be applied to all standards. * 9 – Use Effective Quality Assurance Techniques: This principle applies to STD-001, STD-005, STD-006, STD-007, STD-010. * 10 – Adopt a Secure Coding Standard: This principle generally applies to all standards |
| **5** | For ranking these from highest to lowest for each of these standards, this relates also to the remediation cost if these standards were not employed. 1, 2, 6, and 9 all have a high remediation cost. For overall severity, the standards 1-5, & 8-10 all have high severity, while only 6 & 7 have a lower severity. Then, based on priority, the standards 3-5, 8, & 10 all have the highest priority of 18. And 1-2, & 9 have a priority of 9, the remaining two, standards 7 and 6 have a priority of 4 and 1 respectively.  Based on this, I set up the previously seen threat matrix to represent the priority and likelihood. But I find that those with the highest priority and level, such as standards 3-5 & 8 & 10 are the top of the ranking system. |
| **6** | The three types of encryption policies are for encryption in flight, at rest, and in use.   * Encryption in rest:   + Encryption in rest is when you are encrypting data that is being stored (potentially archived and no longer actively being used). This is done to protect the data while it is being stored either physically or within the cloud. * Encryption at flight:   + Encryption at flight is when you are actively transferring data from one point to the next, sometimes to other recipients. Encrypting the data while it is undergoing this moment protects it so it is not intercepted or stolen, even if just to see the contents of such data * Encryption in use:   + Encryption in use is when you have data that is actively being used, either the data is being edited, erased, viewed, or created. This is the sort of data that should be encrypted as it is at a higher risk for and may be a target of data breaches. |
| **7** | The Tripe-A policies are of Authentication, Authorization, and Accounting   * Authentication:   + Authentication is what verifies a user’s identity, and that a specific user is who they claim to be. This can be done using username/password for verification of users and expanded to have something like a 2-factor authentication to those user accounts. * Authorization:   + Authorization is what controls what a user is capable doing, or what resources they have access to. An example of this is a normal user may have only read access to files, while a higher-ranking user (possibly administrator) would have full read/write access. * Accounting:   + Accounting, or auditing, is a record or log of data that would show what recourses were accessed, at what time, by which user and so on. Every action done by a user has a trace which is logged. This is done to essentially keep track of what is always going on, and by who. |
| **8** | Though most unit testing will make use of assertions, this unit test specifically applies to the STD-006-CPP: Assertions.  This first test involves testing if the clear function erases a collection, and then asserting the collection size is now 0 |
| **9** | This test is a negative one, where we want the test to fail by using the ASSERT\_THROW when accessing a vector out of range.  This unit test applies to the STD-002-CPP: Data Value, as to not go out of range  This test is defining a vector with the size of 5, and then asserting a throw will occur when using the at function for a number out of range. |
| **10** | Another negative test by using ASSERT\_FALSE by asserting the collection is size 6 even after using the pop\_back function.  This test starts by adding 6 entries, using pop back to remove the last item, and then asserting that the number of entries is still 6, though it would now be 5 |
| **11** | Finally, a positive test, to assert that the collection size is equal to the value it has been increased to by using the push\_back function.  This test starts by adding 4 entries to the collection, using push back, and then asserting the size is 5 |
| **12** | During the “build” and especially “Verify and test” phase of pre-production, automatic static code analysis tools can be implemented to test code for bugs or other known vulnerabilities before the software reaches production stage.  As for when in production, automation can be implemented during the “Monitor and detect” phase, and the “Maintain and stabilize” phase, during Monitoring you can use automation to test for any newer threats that may arise like before unknown bugs and vulnerabilities. And once in the Maintain and the potential threats have been resolved, before exiting the production process you can ensure all other security flaws and other vulnerabilities are truly accounted for. |
| **13** | Some tools listed within the security policy for different coding standards are as follows:   * + Astrée: A static code analyzer that proves the absence of runtime errors and invalid concurrent behavior in safety-critical software written or generated in C or C++   + CodeSonar: A static code analysis tool from GrammaTech that provides industry leading SAST solutions or enterprise software security teams. It is used to find and fix bugs and security vulnerabilities in the source code.   + Parasoft C/C++ test: A fully integrated software testing solution for embedded safety-critical industries. |
| **14** | The main problems, such as the risks and benefits, of waiting to act, or doing it now are that some problems can be extremely costly to fix later. Having defense in position for the potential attacks you may (and most likely will) experience later down the line is better. It is better to be preventative and prepared than to wait until after the fact to install some sort of security.  Though it may take more time to develop with security in mind to have defense in depth, and there may be some costs involved with this approach as well, but leaving defense or security till the end, or until after you have been attacked is much more costly, not only for your company, but your customers. That sort of cost is not only financial, but it costs your reputation and trust within your customers, which can damage your public image. |
| **15** | * + - Potential gaps within the security policy would be further explanation of the cloud side of security, as most policies and standards relate more to code security than security aspects of the cloud.     - The security policy does not go into incredible detail about the layers of defense in depth beyond what was shown in the image. More explanation and detail could be added per each level and aspect for a full defense in depth understanding.     - The security policy is not an exhaustive list of all vulnerabilities, as more will be discovered in time, and other standards and best practices may also evolve over time. There could be more to upgrade and expand this security policy, as this is just a decent starting point.     - The security policy glosses over “devsecops” and could use more time explaining the intricacies of how this works for a more in depth understanding. |
| **16** | * The main standards to be implemented would be incorporating security and defense in depth at a start and at every step or phase of the software development life cycle. * Follow the best practices and security principles, such as within this security policy. Much of the roadmap is laid out and well defined on the way to remain secure. * Stay up to date and continue to evolve your security. Security is not static or stagnant and is constantly changing, things that were safe before may eventually be exposed to be vulnerable. Adapt with these changes and threats to security and continue to upgrade your security and patch up any weak spots in your defense. * When developing software, testing should be done on the regular to ensure no bugs or vulnerabilities slip through the development phase and into production. Either manually or automatic static code analysis tools can assist with this. |
| **17** | Here are the references for this presentation. |