# Chris Gollnick | 16 April 2022 | Project Two: Security Policy Presentation

[**https://youtu.be/tpxPlAgKu-I**](https://youtu.be/tpxPlAgKu-I)

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | My name is Chris Gollnick. Welcome to my Security Policy Presentation for Green Pace. |
| **2** | This presentation is to present a solution for development at Green Pace to address security. The hope is that through this we can develop a guide of standardized and unified principles, standards, practices, and guidelines for developers to follow. Moving forward all development security should be to adhere and build to a security profile consistent with Defense in Depth, Triple-A frameworks, and Encryption policies. |
| **3** | The standards we will choose to employ will be broken down into how likely they are to be issues as well as what their priority levels are. The Likely and Unlikely categories are referenced in the SEI Cert guides as a standardized volume of rules to follow. The priorities are also taken from there. Anything lower than P6 will be deemed a low priority while everything else is a higher priority due to its severity and cost. |
| **4** | There are 10 main principles that should stand as the commandments of secure coding that everything ties back to. These principles will be the guiding influence to keep us inline with DiD standards and ensure we are testing for vulnerabilities and exploits. Validate input data refers to validating the data sources and ensuring their compliance, this links to Standards 2 & 3. Heed Compiler warnings is ensuring all warnings from the compiler are read, reviewed, and addressed. This affects standard 1. Architect and Design for security is developing an architecture utilizing sub-systems and ensuring the proper access levels and affects standard 10. Keep it simple is about coding features without being needlessly complex as complexity can lead to more security issues and affects standard 8. Default deny is reference to security policies and that the default action is to deny access and only give access when necessary. Adhere to the principle of least privilege is about performing tasks with the lowest level of privilege by default and ensuring things do not run as an elevated user. Sanitize data sent to other systems is about cleaning and removing data after it is sent from other systems as to ensure the ability to recover at rest data is not a possibility. This affects standards 4,6,and 9. Practice defense in depth is about ensuring that we have multi-layered security so when an attack occurs if they get through one defense there are others there to prevent further access to information. Use effective quality assurance techniques covers using the proper techniques to ensure there are no vulnerabilities due to weak programming or lack of security and using automation to help identify those weak areas. This affects standards 3,4,7,9,and 10. Adopt a secure coding standard is about adopting security practices, secure coding standards, and security policies early on to ensure the proper security awareness and proactive measures are in place. This affects standards 1,5,6,and 10. |
| **5** | There are 10 coding standards taken from the SEI Cert guides that are to be integrated into our standardized practices. These are ordered by highest severity, then highest cost, then highest likelihood, and finally by priority. |
| **6** | We also need to consider encryption policies. There are three main types: In Rest, or when data is not actively used; At Flight, or when data is in transit; and in Use, or when data is actively used in volatile storage. At Rest should be employing the use of algorithms to monitor this data to prevent data breaches and ransomware attacks. At flight should be utilizing secure transfer protocols, encrypted packets, and other such methods to protect the data as it transfers over the web as to prevent over the air attacks. In use data should be protected based upon its usage and allocation. This may mean ensuring memory clear protocols are used, files are closed when done, and other similar methods to prevent tactics like memory scraping that would allow someone to re-access the information when you are done with it. |
| **7** | The triple A framework consists of three elements. Authentication is who and identifies who is trying to access the program, data or system.  Authorization is the what and determines what an authenticated user can see or interact with and also determines with security policies and access levels the user has access to and are applied.  Accounting is when and how and is used to log what happened when accessing the system. This may be logging successful or failed login attempts, why they failed, and where they occurred from or may be logging all changes to the data or personal settings to hold an authenticated user accountable when issues arise from what actions they performed as well as present a starting point of how to recover from such issues. |
| **8** | The next few slides focus on unit testing standards with examples and explaining what went well and what can be approved. This particular slide is a negative unit test that checks for an out of range exception. Here we can see it has a clear name to the test so any team member knows what this test is supposed to do. There are also clear comments at each line that indicate what is being done in case another developer needs to modify it. Since it is a negative test we are expecting it to fail so the use of the try block and the expect statement work well for this usage. I think to improve upon this creating a second test or modifying this one to adjust the size when the exception is caught and testing again would be a way to convert to a positive test and see if that type of action can prevent such errors. |
| **9** | This unit test looks at being able to modify the collection by adding a single value to it. It is clearly named and commented with steps that occur with in it. Assertion statements are used to test the functionality and verify the results. This could be expanded or a second test created to test what happens when a non integer number is entered for the size. |
| **10** | This unit test looks at the size and verifies that the max size is equal to or greater than the collection size. This is to ensure that there is enough room for the entries. Clearly named and commented for clarity. Uses assertions as well to verify each size. The modifications on this test can be what happens when an enormous integer is entered such that it defies the maximum limit or what happens when a negative number is entered. |
| **11** | The final unit test example tests for removing an item from the collection. Again clearly named and commented for easy understanding later. Assertion statements are used to verify the results of the test. This particular test only tested the removal of the last entry and could be modified for what happens when removing an entry in the middle or beginning. |
| **12** | This is the DevSecOps diagram. It displays the movement through the development phases during pre-production and production and how they intertwine into each other. This should be referred to as the progress of how applications are developed in planning and in execution to ensure security and accountability. |
| **13** | While the pipeline diagram shows how the phases move, we need to act within them on top of that to ensure security. Each of our standards has automation tools that must be deployed at various steps. In pre-production we should be running automations at the build or verify & test phases. These locations are clear indicators of when major coding steps are accomplished and make the most sense to run there. Best case it should be run at both phases to ensure we are catching as many issues as we can in pre-production. In production, automations can be run at any phases and it is recommended that they be run in multiple phases. At the minimum automation must be run during the Maintain & Stabilize phase. This location will verify that whatever fixes we apply to correct a problem are up to date with the standards. |
| **14** | Why should we act upon this policy sooner than later? Doing it sooner ensures that our teams are operating on the same standards. IT will also unify the teams in their practices and ensure all features integrated are capable of working together. Waiting on this means increased time and spending to re-build features and refactor code later that could have been done from the beginning. By doing this we operate our development teams at a more efficient standard and eliminate errors while developing instead of trying to handle them all at the end. The removal of needless redundancy of tasks will help with the flow of development, ensure higher morale amongst development teams, and save the company revenue by not having to find errors and send back to development to fix entire features. |
| **15** | The current issues we are looking to solve are lack of standard and unified rules, frameworks, and guidelines that allow developers to work independently of the style of the team and lead to larger security issues. As it stands, testing for vulnerabilities and exploits may not currently occur and as such leaves any product released by the company to have to deal with reactive security measures or potentially lawsuits. BY administering the proposals in this policy we can mitigate many of those issues and ensure proper security testing and frameworks exist and are adhered to. We should look at the Microsoft Security Development Lifecycle as a good way of planning the integration into the pipeline. |
| **16** | In closing, this serves as a draft of the security policy that we should adopt. IT will allow for adherence to the Principles of Secure Coding as well as identify coding standards we should follow. New threats are created daily and this document will allow us to remain hypervigilant of these threats and know how and when to look for them as well as can be continually and regularly updated to have the latest information regarding those threats. This will allow us to incorporate security earlier and thus be able to deal with issues sooner or before they’re a problem. It will also allow us to standardize and unify the team and their practices from the beginning so all team members are aware of how they should be progressing forward and maintaining their parts of the project. |
| **17** | For reference, the location and details of the SEI Cert Coding Standards guide and information on the Microsoft Security Lifecycle. |