# CS 405 Project Two Script

Abby Farnsworth

October 17, 2021

YouTube Link: <https://youtu.be/_2cPsLxXbDQ>

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Welcome to the Green Pace security policy presentation. My name is Abby Farnsworth. I am the developer of this policy, and I will be walking through it with you today. |
| **2** | Every company needs to have security policies and procedures in place. A security breach is a very damaging event to a company. They are expensive, challenging to resolve, and they can do major harm to a company’s reputation. This security policy is meant to support defense in depth best practices.  Defense in depth, or DiD, is an approach to cybersecurity that involves a series of layered defense mechanisms set up to protect data. Using DiD will not successfully stop every attack launched against your organization, but it will give you an all-around more secure environment. |
| **3** | This Threats Matrix table shows the likelihood of insecure coding being exploited, and the priority that fixing this issue will be given. Red is a worst-case scenario, green is the best-case scenario, and yellow is an event that falls somewhere in the middle. |
| **4** | Let’s review the 10 core security principles and the coding standards that apply to them. #1, validate input data. Input validation is the proper testing of any input supplied by another user or application. This input must be verified before use because it could have been created by a malicious user. If the new input is malicious, it could open your program up to injection attacks or other security breaches. #2, heed compiler warnings. Compiler warnings are intended to warn designers of bugs in the code. These warnings are important because some bugs cannot be picked up by testing. Warnings should be addressed, and the code corrected. Ignoring the warning or decreasing the warning level will lead to buggy, non-secure code. #3, architect and design for security policies. Software architecture involves designing your software in such a way that security policies are implemented and enforced. This design may include requiring credentials to access certain documents or applications, or only allowing IT admin to install software. #4, keep it simple. Ideally, the design for software is as small and as simple as possible. The more complex the design is, the more likely it is that errors were made. Also, the more complex a program is, the harder it is to properly test and examine for bugs. |
| **5** | #5, default deny. Default deny means that access to a system or program is based on permission. Thus, by default, users are denied access. Certain conditions must be met in order to use the system or program. #6, adhere to the principle of least privilege. The principle of least privilege refers to the concept of giving users the minimum amount of access to a system or program that they need to do their jobs. You can also limit the amount of time a user has to access a system. These restrictions reduce an attacker’s chance to gain access to important applications or data. #7, sanitize data sent to other systems. Data sent to complex systems (command processors, external programs, relational databases, etc.) should be cleansed of any special characters that may trigger actions or commands. If this event occurs, it may result in a software vulnerability that hackers can exploit. |
| **6** | #8, practice defense in depth. Defense in depth is a way to manage risk by layering multiple defensive strategies together. If one layer of security fails, another is there to keep the line of defense going. Without defense in depth, a system has a single point of failure, and the risk of a security breach is high. #9, use effective quality assurance techniques. Quality assurance techniques are meant to identify and eliminate security vulnerabilities. A company may try fuzz testing on its systems, which is the process of feeding invalid or unexpected inputs into a program to find coding errors or security loopholes. They may also implement penetration testing, which is a simulated attack against your systems to expose vulnerabilities. #10, adopt a secure coding standard. A secure coding standard should be established and applied for your chosen programming language and the platform in which you create your code. This ensures that all developers are on the same page and following the same guidelines, eliminating the chance that someone is using non-secure coding practices. |
| **7** | These are our 10 secure coding standards listed in order of priority and threat level. Each of these standards is explained in depth in our security policy. I include examples of insecure and secure code, and I also provide a list of automation tools that seek out vulnerabilities like these. I have the first 5 standards on this slide- |
| **8** | -and here is the second part of the list. |
| **9** | This graph shows how to calculate priority. When facing an issue, we consider 3 factors: severity of the issue, likelihood that it will be exploited or cause unexpected behavior in the system, and the cost to fix the issue. Each factor is assigned 1, 2, or 3. We then multiply those numbers together and see which ring it falls into. |
| **10** | These are the 3 types of encryption: in rest, at flight, and in use. Data in rest—meaning data not being used--is stored in a secure location and protected by a firewall. As an additional layer of security, it is encrypted. This is called encryption in rest. Data in transit is considered at flight. Encryption at flight will protect this data while in transit. Methods of encryption at flight include third-party certification, always-ON encryption, independent keys, and fast key rotation intervals. Data in use utilizes encryption in use. The XOR Encryption Algorithm will encrypt the data you are using by applying a mask, and it create a cipher text. Reapplying the same mask will un-encrypt the data and revert the file back to plain text. |
| **11** | These are the Triple-A policies, authentication, authorization, and accounting. Authentication refers to the process of identifying the user attempting to access the system. To gain access, a user must provide their username and password, and complete multifactor authentication, such as providing the security code texted to their phone or emailed to them. Authorization determines what a user can access once they have entered the system. The principle of least privilege states that a user should have access only to what they need to perform their duties. A user’s level of access determines what files they can create, read, update, and delete, it determines what drives they can access, and what programs they can open. Accounting is the act of monitoring and logging what a user does inside the system. The system must track what files the user opens, when they enter a program and how long they use that program, and any changes they make, such as updating or deleting files. |
| **12** | Unit testing is a good way to check for coding vulnerabilities. C++ is subject to buffer overflows. Moving data from one vector to another without first checking the size of the receiving vector can lead to a buffer overflow. The below code is an example of this. |
| **13** | In this next slide, we have unit tests checking the security of a vector. We can ask our unit tests to give us positive or negative results. Positive results prove that the code works as intended when tested. Negative tests prove that an error or exception is thrown when the program is fed bad data. The first test here is a positive test checking that the Resize function sets the size of the collection to 0. The second is a negative test to see what happens when you try to call an index that is out of bounds. |
| **14** | We can use Assert statements in unit tests and ask them to return either true or false. In the first test, we ask the computer to verify that it is true that the collection increases to the right size. In the second test, we ask the computer to verify that it is false that the collection size has not changed. |
| **15** | This is the DevSecOps pipeline. There are 5 primary stages in software development: planning/development, code creation and commitment, building and testing, production, and operation. Every stage of development and operation includes a plan for security. In the DevSecOps design here, planning and development encompasses the “Assess and plan” and “Design” sections. Code creation and commitment and building and testing are covered in “Build” and “Verify and test”. Production is handled in “Transition and health check”. Operation covers “Monitor and detect”, “Respond”, and “Maintain and stabilize”. Planning and development should include threat modelling and the use of the IDE security plugins. In the code creation and commitment phase, you can perform static security testing and security unit testing. For building and testing, you can move to dynamic security testing and security acceptance testing. Once the project goes to production, you can perform security smoke tests and penetration testing. Lastly, once the project is operating, there should be continuous monitoring, threat intelligence should be kept up to date, and further penetration testing can be applied. |
| **16** | To make the security process more comprehensive and efficient, underneath each coding standard in the security policy is a table listing automated tools used for testing for security vulnerabilities. This slide shows an example of automated tools provided for one coding standard. There are multiple tools listed per coding standard, but these are not an exhaustive list. There are many other security tools available that are useful in doing automated security checks. |
| **17** | Immediate steps must be taken to adopt the security measures laid out in this security policy. Without these standards in place, there is a high likelihood of code vulnerabilities being found and exploited by bad actors. These standards have been created to ensure that applications developed by Green Pace reflect secure coding practices, which reduce the chances of such events like malicious code getting inserted in software. None of these new security standards pose risk to the company; rather, they offer the rewards that come with creating better, safer products. |
| **18** | As comprehensive as I have tried to make this security policy, it is impossible to include everything. There are many rules and recommendations laid out in the [SEI CERT C++ Coding Standard](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682) book that provide great information for developers looking to code more securely in C++. It should also be noted that secure coding is not a static field. The [SEI CERT C++ Coding Standard](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682) is continuously being upgraded as developers find new ways to create secure code. There are also other security practices that can be looked into in more depth. Two additional security principles that might be added to this policy are cryptographic practices and error handling and logging. Harvey explains how to use cryptographic practice: “Using quality modern cryptographic algorithms with keys stored in secure key vaults is a practice that increases the security of your code in the event of a breach”. Error handling and logging are two useful techniques for minimizing the impact of buggy code. Error handling attempts to catch errors in the code before they result in failure. Logging documents these errors so that developers can diagnose and correct these bugs. |
| **19** | Software development here at Green Pace requires the implementation of these security policies and these secure coding standards. The governance provided in this security policy will ensure that all team members have the knowledge and guidance needed to produce secure code. When the entire development team is following these best practices, it ensures that Green Pace is creating secure, functional, and consistent products. |
| **20** | Please see this slide for a list of references used in the creation of this presentation. |