# Security Policy Presentation

# Michael P Clisbee

CS 405

# Southern New Hampshire University

<https://youtu.be/pM8-XwDUsLg>

# CS 405 Project Two Script

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | My name is Michael and today I am presenting the use of external testing methods to identify potential vulnerabilities by presenting the Green Pace security policy guide, and to provide implementation guidelines and recommendations for maintaining it in the future |
| **2** | Here we see the Defense in depth, which is a cybersecurity concept that involves constructing layers upon integrated layers of defensive mechanisms to protect sensitive data and information. For example: Multi-factor authentication. Multi-factor authentication is an electronic authentication method which grants a user access to an application application only after presenting two or more authentic pieces of evidence to an authentication mechanism: knowledge (something only the user knows), possession (something only the user has), and inherence (something only the user is). |
| **3** | The Security threat matrix can be divided into four sections as seen in this matrix. We have the likely, priority, low priority and unlikely. Like is the loss of confidential information. Priority is when the system goes down, those will be the first things to get back up. Low priority is where there is inadequate exception handling. Unlikely covers Loss of integrity. |
| **4** | In this slide, we will identify the 10 different types of security principles.  • **Validate Input Data** - Obey the one-definition rule  • **Heed Compiler Warnings** - Do not cast to an out-of-range enumeration value  • **Architect and Design for Security Policies** - Do not attempt to create a std::string from a null pointer  • **Keep It Simple** - Be careful using functions that use file names for identification  • **Default Deny** - Use a static assertion to test the value of a constant expression  • **Adhere to the Principle of Least Privilege**  • **Sanitize Data Sent to Other Systems**  • **Practice Defense in Depth**  • **Use Effective Quality Assurance Techniques**  • **Adopt a Secure Coding Standard** - Handle all exceptions |
| **5** | We have different coding standards. Below are the 10 coding standards.  • **Data Type** - Obey the one-definition rule  • **Data Value** - Do not cast to an out-of-range enumeration value  • **String Correctness** - Do not attempt to create a std::string from a null pointer  • **SQL Injection**  • **Memory** **Protection** - Do not call a deallocation function on anything other than nullptr  • **Assertions** - Use a static assertion to test the value of a constant expression  • **Exceptions** - Handle all exceptions  • **Input Output** - Be careful using functions that use file names for identification  • **Namespace Modifications**: Do not modify standard namespaces. New declarations in the namespace can cause undefined behavior  • **Value Inputs**: Value returning functions must return a value from all code paths |
| **6** | I am now going to define different types of encryption policies.   * **Encryption in rest** - Encryption for data at rest involves securely encoding data as it is written into storage and decrypting that data as it is pulled from storage for use. Using an encryption key when the data is written into storage protects it from unauthorized access. It should be used for all data of any level of sensitivity and would cause harm if accessed by unauthorized actors. * **Encryption at Flight** - Encryption of data at-flight involves securely encoding data as it is being transmitted. How you will be transferring any data will determine how to apply this encryption. Implement secure protocols when using web browsers. When sending emails encrypt before sending and use digital signatures. * **Encryption in Use** - Encryption of data in-use involves protecting data as it is utilized in memory, via password protected profiles protecting the memory of each profile |
| **7** | Now we will summarize the Triple-A policies that support authentication: • **Authentication** - This process used to prove who a user is, their user ID, passwords, higher-level security such as secure tokens, CAC or PIN and other dual authentication.  • **Authorization** - Once a user is authenticated and allowed access, they are only granted specific access to parts of that system. Authorized access to certain drives, folders, programs, and other data is allowed by the system administrators  • **Accounting** - After authentication and authorization, you need to monitor and record |
| **8** | The diagram here shows the results of 13 unit tests which include google ASSERT and EXCEPT, bringing back a mix of negative and positive results. These were created using the Google Unit Testing Framework. |
| **9** | Here we have a diagram of the automation summary, also known as the DevSecOps pipeline. |
| **10** | The DevSecOps pipeline is a secure coding method that has a continuous delivery using the security-focused software development life cycle (SDLC). This is a solid structure for the system, that goes through 6 segments to include Planning, Coding, Testing, Building, Release and Deployment. Ensure you are testing early and often to detect any flaws and resolve them before time of release.  One effective Application tool is called DAST: which stands for Dynamic Application Security Tool. This analyzes applications while they run, breaks encryption algorithms from the outside and verifies permissions. |
| **11** | There is no such thing as 100% secure code but adopting secure coding practices removes common vulnerabilities. As well, priming your security policy at the start will reduce long term hazards and costs. Always remain vigilant in the assumption that all systems have implied threats and flaws to be identified. Continuous personnel training is essential in staying ahead of the issues that are ever evolving. |
| **12** | All security policies can contain gaps. The key is to identify all possible avenues of attack approach and use that as you baseline for security creation.  All standards should be adopted to prevent future problems. Test defenses before the breaches occur and develop a plan to respond when security fails.  An example of gaps in an existing security policy took place in 2012 and made the news because it impacted over 167 million user accounts that were still actively being used with the users unaware of the intrusion.  Hackers cracked over 167 million user accounts, stealing over 117 million email accounts and passwords.  It was only discovered when the hackers posted the user accounts on a Russian password forum.  This data was placed in a database that was passed around to various hacker networks who then attempted to sell the information for a minimal amount of money.  Breach notification site researchers discovered this as well and contacted numerous victims to confirm that these were current passwords and emails still actively used. This was a combination of both security and data breaches. After the fact, it was pointed out that LinkedIn did not have sufficient security protocols in place.  Once it was verified that the breach was real, LinkedIn fixed the security vulnerability and instructed all users to reset their passwords.  If any user account did not reset their passwords, the account was deactivated.  LinkedIn initially offered protection tools such as email challenges and dual authentication protocols, but these were not enforced until after the fact.  What preventative measures could have been taken? Some lessons that were learned were that many users had the same password used over several active accounts, which made it easier for hackers to access more information.  Recommendations for improvement:  Scanning is the process of analyzing code to ensure that it is safeguarded from security vulnerabilities. This includes both manual and automated code  review. AppSec tools — such as SAST and DAST — are used during this phase.  This phase enables developers to address security vulnerabilities and bugs  earlier in the software development life cycle.  During the Analyze phase, all of the collected data and metrics from the previous phases are reviewed to identify all of the security risks. Then, those risks are  compiled into a list ranging from most to least severe  Remediate  Some DevSecOps tools —  like DAST — can recommend solutions for the vulnerabilities, errors, and bugs that it has identified. This makes it easier to address security issues as they arise.  In addition, it may be beneficial to also track and  manage the differences between the actual and target metric values. This helps to make informed data-driven decisions during the software development lifecycle. |
| **13** | • In conclusion, based on the principles and standards described in this presentation it can be determined that all possible avenues of attack approach have been covered to create and maintain a secure and proficient program. A zero-trust policy will be implemented when it comes to internal and external data access throughout the company network, creating a situational awareness necessary to maintain security and privacy and to keep all sensitive information safe and secure.  All the coding standards and security policies should be adopted to prevent future problems. |
| **14** | Thank you for taking the time to read this |