# Dhevie Miller

CS 405 – 10932

Project Two: Security Policy Presentation  
August 13, 2024

[YouTube Link](https://youtu.be/BKC8LvF3V0s)

# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello there! My name is Dhevie Miller. I will be presenting the Green Pace security policy guide, explaining its security standards and policies, as well as other important topics regarding attacks and vulnerabilities. This is to ensure that everyone understands the principles and best practices as our team continues to grow. |
| **2** | This security policy is important to have because it provides clear rules and guidelines for how everyone in an organization should handle information and protect it from risks. It helps manage potential threats, ensures the organization follows legal requirements, and prepares for quick responses if something goes wrong. By clearly defining what’s expected, it makes sure everyone knows their role in keeping things secure.  The illustration demonstrates the best practice of defense-in-depth using layered security. In the context of defense-in-depth, a security policy is essential for ensuring the implementation of multiple layers of security measures. It guarantees that these layers are seamlessly integrated and regularly updated to stay ahead of emerging threats. The policy also encompasses employee training to enhance their ability to identify and address security issues, and it governs access levels to mitigate risks in case of a breach. |
| **3** | This slide contains the 10 coding standards separated into the likelihood of the security risks of either ‘likely’, ‘unlikely’, or ‘probable’. In this case, the coding standards are listed either ‘likely’ or ‘probable’. It is also separated into their level of priority based on their threat levels from 1 to 5, with 5 being the greatest threat.  Under the category “likely” for the probability of a specific issue or vulnerability to occur, we have:  The Data Type Coding Standard, labeled DTP-010-CPP, ensures that using the right type of data stops errors and makes the code clearer and safer.  The Data Value Coding Standard, labeled DTV-020-CPP, makes sure that data values are within the expected range to help avoid errors and keep the system secure.  The String Correctness Coding Standard, labeled STR-030-CPP, ensures that strings are handled correctly to help prevent potential risks of buffer overflows and other security vulnerabilities.  To prevent SQL injection attacks, in the SQL Injection Coding Standard, labeled SQL-040-CPP, user inputs should be properly validated and sanitized before it is used in SQL queries.  The Memory Protection Coding Standard, labeled MEM-050-CPP, involves managing memory correctly to prevent leaks and security problems.  Validating user inputs in the Input Validation Coding Standard, labeled VAL-080-CPP, prevents security problems and makes the program more reliable.  Handling files correctly in the Secure File Handling Coding Standard, labeled SEC-090-CPP, keeps them safe, preventing unauthorized access and data corruption.  Under the category “probable” for the probability of a specific issue or vulnerability to occur, we have:  In the Assertions Coding Standard, labeled AST-060-CPP, using assertions correctly helps find bugs and ensures the code works properly.  The Exceptions Coding Standard, labeled EXC-070-CPP, involves properly handling exceptions to ensure error management is strong and secure.  Properly initializing variables in the Safe Variable Initialization Coding Standard, labeled INI-100-CPP, ensures they don’t have random values, which can cause unpredictable behavior and security issues. |
| **4** | These are the 10 Core Security Principles, each accompanied by a list of coding standards that apply to that principle.  In the Validate Input Data Principle, the data your system receives should always be checked to make sure it’s in the right format and is safe to use. This helps prevent attacks where bad data can break or take over your system. By validating input, you ensure that your system only processes good, expected data. The coding standards that apply to this principle are Data Value, String Correctness, SQL Injection, and Input Validation.  The Heed Compiler Warnings Principle involves paying attention to the warnings your code compiler gives you. These warnings often point out potential issues that could become security vulnerabilities. Fixing these early helps keep your code safe and robust. The coding standards that apply to this principle are Data Type and Safe Variable Initialization.  The Architect and Design for Security Policies Principle involves planning for protection from the very beginning of your system’s design. You should build in security measures as fundamental parts of your architecture. This proactive approach makes your system more secure and easier to manage. The coding standards that apply to this principle are Assertions and Exceptions  The Keep It Safe Principle involves creating simpler designs to make it easier to understand, maintain, and secure. Complex systems often hide bugs and vulnerabilities. By keeping your code and systems straightforward, you reduce the risk of mistakes and security issues. The coding standards that apply to this principle are Data Type and Exceptions.  The Default Deny Principle involves starting off by denying all access and then allowing only what is necessary. This minimizes the number of ways an attacker can exploit your system. By restricting access by default, you control what can and can’t be done, improving security. The coding standard that applies to this principle is Secure File Handling.  The Adhere to the Principle of Least Privilege Principle involves only giving systems and users minimum access to get their tasks done. This reduces the risks and negative effects associated with an account being compromised. Access levels should be regularly reviewed and adjusted to keep them appropriate. The coding standard that applies to this principle is Secure File Handling.  The Sanitize Data Sent to Other Systems Principle involves cleaning and validating data to ensure safety prior to being dispatched to other systems, preventing it from harmful content or exploiting vulnerabilities. Sanitizing data helps maintain the integrity and security of all connected systems. The coding standards that apply to this principle are String Correctness and SQL Injection.  The Practice Defense in Depth Principle involves security measures that should consist of several layers for system protection. In case a layer fails, other layers still provide protection. This layered approach makes it harder for attackers to breach your systems completely. The coding standard that applies to this principle is Memory Protection.  The Use Effective Quality Assurance Techniques Principle involves regularly testing your code and systems to find and fix security issues early. This includes code reviews, automated tests, and security assessments. Thorough testing ensures your system is secure and works as expected. The coding standard that applies to this principle is Memory Protection.  The Adopt a Secure Coding Standard Principle involves adhering to established guidelines and ensuring using best practices when creating a safe and secure code. These standards help you avoid common security mistakes. Consistently applying secure standards makes your software more resilient to attacks. The coding standards that apply to this principle are Data Value, Assertions, Input Validation, and Safe Variable Initialization. |
| **5** | I have ranked the 10 coding standards in priority order. Most of them are labeled either medium or high based on their overall rate using a scale from 1 to 5, with 5 being the greatest threat.  Data Type is ‘medium, level 4’ because incorrect data types can cause bugs, but they’re usually caught during testing, so it’s important but not critical.  Data Value is ‘medium, level 4’ because keeping data values in the right range prevents errors, but these issues are less likely to cause major problems.  String Correctness is ‘medium, level 4’ because mishandling strings can lead to problems like buffer overflows, but careful coding can manage these risks.  SQL Injection is ‘high, level 5’ because SQL injection is a serious threat that can lead to data breaches, so preventing it is a top priority.  Memory Protection is ‘high, level 5’ because poor memory management can cause crashes or security vulnerabilities, making this a critical area.  Assertions are ‘medium, level 3’ because using them can catch bugs during development, but missing them typically affects only development, not production.  Exception is ‘medium, level 4’ because proper exception handling is important for stability, but well-handled exceptions are usually not catastrophic.  Input Validation is ‘high, level 5’ because validating input is key to preventing many security vulnerabilities, making it a high priority.  Secure Variable Handling is ‘high, level 5’ because doing so prevents data leaks and unauthorized access, so it’s crucial for protecting data.  Safe Variable Initialization is ‘medium, level 3’ because properly initializing variables avoids unpredictable behavior, but the risks are usually manageable. |
| **6** | Here are the policies for encryption at rest, in flight, and in use.  Encryption at rest encrypts stored data on physical device, protecting it from unauthorized access and complying with regulations.  Encryption in flight encrypts data during transmission over networks, preventing interception and tampering, which is essential for securing network communications.  Encryption in use encrypts data in memory during use, preventing unauthorized access while it is being processed. |
| **7** | This next slide explains the Triple-A Policies that support authentication, authorization, and accounting.  Authentication involves making sure that only approved users or devices can get into a system by verifying their identity using passwords, Multi-Factor Authentication (MFA), or certificates.  Authorization involves controlling what authenticated users or devices can access and do, using role-based access to enforce permissions and protect resources.  Accounting involves logging and tracking users and device activities for monitoring, auditing, and analysis, helping to detect incidents and ensure accountability. |
| **8** | In the next 4 slides, I will be going over Unit Testing.  This slide shows a positive test that passed, and here we ask the question, “Does push back add the correct value?” This test verifies that the ‘push\_back’ method correctly adds a value to the collection and that the last value in the vector is as expected.  To take this test a step further, the test can be expanded by adding several values to quickly check if the collection keeps the correct order and all data stays accurate. |
| **9** | This slide shows another positive test that passed, and for this test we ask the question, “Does reserve increase capacity without affecting size?” The test verifies that calling ‘reserve()’ increases the collection’s capacity but does not change its size.  To take this test a step further, the test can reserve a much larger capacity, such as 1000, to see how it behaves. |
| **10** | This slide shows a negative test that passed, and here we ask the question, “Does calling ‘at()’ Throw ‘std::out\_of\_range’ for Invalid Index?” This test verifies that calling ‘collection->at(10)’ throws an ‘std::out\_of\_range’ exception when the collection only contains 5 elements.  To take this test a step further, we can test other out-of-bounds scenarios by trying to access negative indices or very large indices to ensure consistent exception handling. |
| **11** | This last unit test is another negative test that passed, and here we ask the question, “What happens when we reserve with a negative value?” This test verifies that reserving a space with a negative value throws an ‘std::length\_error’ exception.  To take this test a step further, we can test with other invalid inputs, such as extremely large values, to see how the collection handles different edge cases. |
| **12** | Automation will ensure the standards in this policy are enforced and followed. Green Pace already has a strong DevOps setup in place. This is an illustration of the DevSecOps Diagram. |
| **13** | In this process, security tools are embedded at every stage to ensure continuous protection. In the pre-production cycle during Assess and Plan, tools assess risks and ensure compliance. In the Design stage, secure practices are validated automatically. The Build stage uses the compiler and security scans to ensure safe code. Verify and Test involves automated security tests and vulnerability scans. As code moves to production, Transition and Health Check tools enforce secure configurations. Once in production, systems are continuously monitored for threats, and automated responses handle incidents. Finally, Maintain and Stabilize ensures systems return to a secure state after any issues. This approach keeps security integrated and effective throughout the entire process. |
| **14** | Failing to include security early in development can lead to expensive fixes and security breaches later. Without consistent security practices, vulnerabilities might slip into the code. Addressing security only after issues arise is riskier and less effective than building it in from the start.  Starting with security makes the application stronger and reduces the risk of breaches, though it might take more time and resources initially. Delaying security might speed up development, but it increases the chances of costly breaches and damage to reputation later on.  The strategy might lack continuous monitoring after deployment, leaving it vulnerable to new threats. Additionally, developers may not receive enough ongoing training to stay updated on security practices. Early security implementation can also slow down development and may face resistance from teams used to different methods.  To overcome these challenges, integrate security from the beginning with secure coding practices and risk assessments. Use recognized standards like CERT to ensure consistency. Train developers regularly and implement continuous monitoring post-deployment to keep the system secure and adaptable to new threats. This approach reduces risks and ensures the software stays secure throughout its lifecycle. |
| **15** | The MGM Resorts cyberattack in September 2023 serves as an example of how critical it is to continuously evaluate and update security policies. Despite the existing security measures, MGM Resorts fell victim to a sophisticated cyberattack that exploited vulnerabilities through social engineering and ransomware. This incident highlights several gaps that could be present in this security policy.  There may be a lack of continuous security training. For example, the attackers used social engineering to gain unauthorized access, indicating that MGM employees might not have been adequately trained to recognize such tactics. Without regular training, employees may fall prey to phishing or other deceptive practices, which can bypass even the most robust technical defenses.  There may be insufficient incident response training. For example, the widespread disruption caused by the attack on MGM's systems, including slot machines and online services, suggests that their incident response plan may not have been comprehensive or tested regularly. A well-prepared incident response plan should minimize downtime and reduce the overall impact of an attack.  There may be a lack of regular vulnerability assessments. For example, the attackers exploited vulnerabilities that could have been detected and mitigated through regular security audits and vulnerability assessments. These assessments are crucial for identifying and addressing potential weaknesses before they can be exploited. |
| **16** | To prevent future problems from occurring, we should adopt some standards.  There should be regular security awareness training by conducting mandatory training sessions for all employees on recognizing phishing, social engineering, and other common attack vectors. This training should be updated regularly to address new and evolving threats.  A comprehensive incident response plan should be implemented by developing and regularly updating an incident plan that includes specific steps for different types of attacks. This plan should be tested frequently through simulations to ensure all team members know their roles and responsibilities.  Routine vulnerability assessments and security audits should be implemented by performing regular vulnerability scans and security audits to identify and fix weaknesses. Tools like penetration testing can help simulate attacks and assess the system’s defense.  Continuous monitoring and real-time threat detection should be improved by implementing continuous monitoring tools that provide real-time alerts on suspicious activities. This would allow for immediate responses to potential threats before they escalate into full-blown incidents.  In conclusion, the MGM Resorts cyberattack underscores the need for continuous evaluation and improvement for security policies. By addressing current gaps, such as the lack of regular training, insufficient incident response plans, and the absence of routine vulnerability assessments, we can better protect ourselves from future threats. Adopting a proactive approach with continuous monitoring and continuing to use the Defense in Depth strategy will further reduce the risk of cyberattacks and enhance overall security resilience. |
| **17** | Last but not least, the Reference page. Thank you for your time! Have a wonderful day! |

**fo**