# CS 405 Project Two Script

Michael Schultz

CS 405

4/23/24

Video Link: <https://youtu.be/TWUN1738cEo>

| **Slide Number** | **Narrative** |
| --- | --- |
| **1**  **INTRO** | Hello everyone, my name is Michael Schultz, and this presentation will serve as an outline for the Green Pace Security Policy. This presentation will provide answers as to what the security policy is, what it hopes to achieve, and how it can be utilized to reinforce the security across all applications at Green Pace. |
| **2**  **OVERVIEW: DEFENSE IN DEPTH** | To meet the requirements of standardizing security compliance while continuing to provide a secure platform across all applications within the organization, it was necessary to construct a security policy that closely aligned with the Green Pace security mission. The Green Pace security policy defines core security principles, C/C++ coding standards, encryption policies, and Triple A policies, which through secure software design and development, strives to achieve the Green Pace security mission of Defense in Depth. This is accomplished through the implementation of secure coding best practices that provide multiple layers of defense in protecting against threats and vulnerabilities. |
| **3 THREATS MATRIX** | The threats matrix is used to assess and categorize the risks of potential vulnerabilities. In this threat matrix, the vulnerabilities that each standard addresses within the Green Pace Security Policy are categorized based on four different categories: Likely, Priority, Low Priority, and Unlikely. The category of each risk is selected based on the severity, likelihood, and impact that the potential vulnerabilities have.  In the likely section are the coding standards for data type, data value, and string correctness. In the priority section are the coding standards for SQL injection, memory protection, and integer operations. The low priority section contains coding standards for assertions and exception handling. Lastly, the unlikely section contains the coding standards regarding appropriate use of exceptions and file handling. |
| **4**  **10 PRINCIPLES** | The Green Pace Security Policy contains ten core security principles. These principles are Validate Input Data, Heed Compiler Warnings, Architect and Design for Security Policies, Keep it Simple, Default Deny, Adhere to the Principle of Least Privilege, Practice Defense in Depth, Use Effective Quality Assurance Techniques, and Adopt a Secure Coding Standard. Starting with the first principle, Validate Input Data, this principle covers the importance of utilizing proper input validation techniques to combat vulnerabilities through malicious input. The coding standards that apply to this principle are the Data Type, Data Value, String Correctness, SQL Injection, and Integer Operations Standard.  The second principle, Heed Compiler Warnings, stresses the importance of never ignoring compiler warnings and properly modifying code to address these warnings to avoid vulnerabilities. The coding standard that applies to this principle is the Assertions Standard. The third principle, Architect and Design for Security Policies, reinforces the designing of software architecture with the implementation of security policies in mind. This policy is supported by the Appropriate Exceptions, File Handling, and Memory Protection Standards. The fourth principle, Keep it Simple, highlights the importance of designing software as simple and as small in scale as possible to avoid vulnerabilities caused by complexity. The coding standards that support this principle are the Memory Protection and Appropriate Exceptions Standards.  Default Deny is the fifth principle and ensures incorporating permission-based systems when appropriate that sets the default permissions to be denied mitigating risks of unauthorized access. This principle is supported by the File Handling Standard. The sixth principle is named Adhere to the Principle of Least Privilege and enforces the idea that the least privilege necessary should be granted to complete a task within a system to protect against unauthorized access and reduce the risk of arbitrary code being executed with elevated privileges. The coding standard that applies to this principle is the File Handling Standard. The seventh principle, Sanitize Data Sent to Other Systems, stresses the importance that all data passed to external systems should be properly sanitized to avoid injection attacks, buffer overflows, and string vulnerabilities. The coding standards that this principle applies to are the Data Value and SQL Injection Standards.  The eighth principle is called Practice Defense in Depth and refers to the practice of utilizing multiple defensive layers to prevent vulnerabilities and enhance the overall security of a system. This principle is supported by multiple coding standards that include String Correctness, SQL Injection, Assertions, Exceptions Handling, Integer Operations, Appropriate Exceptions, and File Handling Standards. The ninth principle is Use Effective Quality Assurance Techniques and ensures the usage of quality assurance techniques as they can help in identifying and addressing vulnerabilities. This principle is supported by the Exceptions Handling and File Handling Standards. Lastly, the tenth principle, Adopt a Secure Coding Standard, reinforces the idea of developing or following secure coding standards to mitigate security risks. The coding standards that apply this principle are the Memory Protection and Appropriate Exceptions standard. |
| **5**  **CODING STANDARDS** | The coding standards presented in the Green Pace Security Policy have been ranked based on the severity, likelihood, and impact of the vulnerabilities that each coding standard aims to address. In the first spot is the Data Type Standard that deals with data type issues and enforces the correct use of data types for values to prevent issues such as integer overflow and underflow. Second is the Data Value Standard which ensures that data values are validated and sanitized to reduce the risk of buffer overflows and format string vulnerabilities. Third is the String Correctness Standard which focuses on the implementation of bounds checking and null termination to prevent issues like buffer overflows. Fourth is the SQL Injection Standard which highlights the importance of using parameterized queries to mitigate SQL injections. Fifth is the Memory Protection Standard that reinforces the use of proper memory management practices like vectors with bounds checking to reduce the risks of buffer overflows and memory leaks.  Sixth is the Integer Operations Standard which ensures the use of secure integer operation practices to prevent integer overflow and division by zero risks. Seventh is the Assertions Standard that enforces the use of assertions to validate assumptions to detect unexpected behavior during program execution in the debugging process. Eighth is the Exceptions Handling Standard which covers the importance of utilizing proper exception handling to prevent potential resource leaks and other vulnerabilities through improper error handling. Ninth is the Appropriate Exceptions Standard that promotes the use of exceptions for handling exceptional circumstances rather than regular control flow to maintain code clarity and prevent potential vulnerabilities within the code. Lastly, in the tenth spot is the File Handling Standard that enforces secure file handling practices to mitigate file security like unauthorized access and resource leaks. |
| **6**  **ENCRYPTION POLICIES** | The encryption policies utilized within the Green Pace Security Policy cover encryption at rest, encryption in flight, and encryption in use. Encryption at rest is the encryption of data that is stored in a resting state. This can be data that is stored in a physical hard drive, database, or even in the cloud. Enforcement of this policy will allow for the security of data at rest by encrypting the stored data so that, in the event of a compromise, the data remains secure. Applying encryption at rest can be accomplished through full disk encryption, database encryption, and cloud assets encryption.  Encryption in flight is the encryption of data that is moving or in transit. This usually refers to the encryption of data that is being sent over a network. By following this policy, data being sent over a network will remain secure by encrypting any sensitive data during transit. Encryption in flight can be applied by using SSL/TLS transport layer protocols to encrypt data in transit. Encryption in use is the encryption of data that is being utilized by an application or system. This refers to the practice of encrypting data that is being processed or used by the application or system. Adhering to this policy ensures that data remains secure by encrypting sensitive data during data processing. A method for applying encryption for data in use is utilizing a memory encryption tool such as SEV that encrypts RAM for data in use. |
| **7**  **TRIPLE-A POLICIES** | In addition to encryption policies, the Green Pace Security Policy also employs Triple-A policies for authentication, authorization, and accounting. Authentication refers to the process of verifying and identifying those trying to access the application or system. This process checks credentials for access, commonly implemented as a username and password system but can also check for physical keys or biometric keys like a fingerprint. Following the authentication policy ensures that appropriate user login and password requirements are enforced for all users, old and new, so that only those with the proper credentials are granted access to data or services that the application or system provides.  Authorization is the process of granting system or application privileges to a user. With authorization, the user level of access is defined so that only the necessary permissions are granted for the expected uses of the application or system. Enforcing this policy includes adopting a principle of least privilege approach and ensuring that every user within the system is only granted the necessary privilege in completing their expected tasks. This can help mitigate the risk of unauthorized users compromising sensitive data in the application or system. Accounting is the process of logging and keeping track of user activity and data within a system. This covers the various logging of activities and can be changes to the database, addition of new users, as well as the files accessed by users. Adhering to this policy means keeping thorough system and user logs that can be utilized to detect suspicious user or system activity. |
| **8**  **UNIT TESTING** | Unit testing allows for the individual testing of software components. It provides an isolated environment making it easier to ensure functionality and find bugs or exploits early in the development process. The following examples highlight how unit tests can be utilized to test software components using assertions. These unit tests include a mixture of positive and negative tests and include testing adding a collection to an empty vector, if at exceptions are thrown with an empty collection, if resizing increases a collection, and if at exceptions are thrown with an index out of bounds. |
| **9**  **CanAddToEmptyVector** | The CanAddToEmptyVector is the first unit test of the examples and tests adding a single entry to an empty collection. The test starts with the first line asserting that the collection is empty by using an ASSERT\_TRUE macro. Then, the size of the collection is verified to be zero with an ASSERT\_EQ macro that asserts that the size equals zero. Once the collection is verified to be empty and the size is zero, an entry is added to the collection. This is checked with an ASSERT\_FALSE macro that the collection is empty. Lastly, ASSERT\_EQ is used to assert that the collection size equals 1. |
| **10**  **CanAtExceptionsBeThrownWithEmptyCollection** | The second unit test is named CanAtExceptionsBeThrownWithEmptyCollection. This is a negative test that checks that an out of bounds exception is thrown when trying to access an index at position 0 on an empty collection. The test starts by using an ASSERT\_TRUE macro to verify that the collection is empty. Then, an ASSERT\_EQ macro is used to assert that the size of the collection is equal to zero to ensure that the collection is empty, and its size is zero. Lastly, an ASSERT\_THROW macro is used to see if an out-of-range exception is thrown at index position 0. |
| **11**  **CanResizeIncreaseCollection** | The next unit test is named CanResizeIncreaseCollection. This test checks if resizing increases the size of the collection. The test begins by verifying that the collection is empty and that the size of the collection is zero. This is done by using ASSERT\_TRUE that the collection is empty and ASSERT\_EQ that the size is equal to zero. Next, a variable is used to store the current size of the collection called size\_state. Resizing is then performed on the original size of the collection plus one. Finally, the ASSERT\_GT macro is used to see if the collection size after resizing is greater than the original snapshot of the collection size stored in the size\_state variable. |
| **12**  **CanAtExceptionsBeThrownWithIndexOutOfBounds** | The final unit test example is named CanAtExceptionsBeThrownWithIndexOutOfBounds.  This is a negative test that checks that an out of bounds exception is thrown when attempting to access an index at position 6 on a collection of 5 entries. Like the previous examples, the test begins by ensuring that the collection is both empty and the size is zero. This is accomplished using ASSERT\_TRUE for the collection being empty and ASSERT\_EQ for the size equaling zero. Next, five entries are added to the collection and verified with the ASSERT\_EQ macro that the collection size equals five. Lastly, an ASSERT\_THROW macro is used to see if an out-of-range exception is thrown at index position 6. |
| **13**  **AUTOMATION SUMMARY** | The following image depicts a Dev Sec Ops diagram. The diagram aims to show how security can be implemented throughout various stages of the development process. The left side of the diagram represents the pre-production phase, and the right side of the diagram the production phase. Starting with the Assess and Plan stage in pre-production, this is where it is recommended to perform risk assessment and planning. Next, the Design phase implements test-driven design and best practices. The Build phase encourages secure code building by using trusted repositories and secure open-source usage. The Verify and Test phase is designated to vulnerability scanning and testing.  Moving to the Production side of the diagram, the Transition and Health Check phase is next where the project is configured and deployed with the appropriate security settings. Next is the Monitor and Detect phase where system logs are collected and analyzed for any intrusion. If intrusions are detected, attacks are blocked, and services stopped and rolled back if needed in the Respond phase. Lastly, the system is assessed against security baseline and returned to a stable state in the Maintain and Stabilize phase. |
| **14**  **TOOLS** | Modifying the existing Green Pace process to a DevSecOps process as described can be done by incorporating automation in compliance with the security standards in the Green Pace Security Policy. This will first require adjustments to the Assess and Plan stage in pre-production. Here, appropriate training can be held to utilize the listed automation tools and how they are configured in detecting violations of the coding standards. This applies to the configurations used for Microsoft Visual Studio Code Analysis and compiler warnings, Cppcheck, and clang-tidy checkers. Next, in the Verify and Test stage during pre-production, the described automation tools can be utilized for thorough vulnerability scanning and functionality, compliancy and security testing.  Moving on to the Transition and Health check stage in production, the automation tool settings used during the Verify and Test stage to enforce coding standards can be utilized to configure and deploy security settings for production. Next, during the Monitor and Detect stage of production, the automation tools can be used to perform static analysis and continuously analyze the code base for security issues and vulnerabilities. Lastly, the automation tools mentioned can be applied to the maintain and stabilize stage of production for any security issues or vulnerabilities addressed during the Monitor and Detect stage. These tools can be used to ensure that the fixes applied for the found vulnerabilities return the system to baseline while aligning with the coding standards of the Green Pace Security Policy. |
| **15**  **RISKS AND BENEFITS** | There are many risks involved in leaving security to the end. Not prioritizing security opens the door to inevitable dangers of errors and vulnerabilities. This can lead to security breaches, data breaches, and other malicious attacks. Those attacks can cause reputational harm to the organization and financial losses. Additionally, the longer it takes to address errors and vulnerabilities the more difficult and expensive the process becomes.  By acting now, these challenges can be mitigated through early and continuous implementation of security measures like outlined in the DevSecOps process. Prioritizing security throughout the project lifecycle reduces the dangers of errors and vulnerabilities that lead to malicious attacks and data breaches. This negates the potential reputational damage and financial costs associated with waiting to the end for security. Proper considerations should be made regarding the sensitivity of the data being utilized by the system and the upfront budget available for implementing security measures to protect that data. |
| **16**  **RECOMMENDATIONS** | While the goal of the Green Pace Security Policy is to standardize security compliance and provide a secure platform across all applications within the organization, it is important to remember that not every security risk and secure coding best practice may be covered in the policy. Over time, new and emerging threats may change the security landscape and new approaches may be necessary in addressing security risks. For this reason, consider the policy ongoing and use it as a guide as it may be required to adapt new coding standards and policies in the future. |
| **17**  **CONCLUSIONS** | As we conclude this presentation, it is imperative that security is treated as a priority and not an afterthought. As stated, waiting to implement security measures leads to inevitable dangers of errors and vulnerabilities. These dangers, such as security breaches, data breaches, and other malicious attacks, can cause great reputational harm and financial loss to the organization. Moving forward, Defense in Depth should be adopted as a standard by adhering to the principles and coding standards within the Green Pace Security Policy. Additionally, automation will be key in providing continuous integration of security through the DevSecOps toolchain and aligning Green Pace with their security mission of “defense in depth.” |
| **18**  **REFERENCES** | This slide is for the references used in the presentation. Thank you for your watching! |