# Bryan Chan

CS 405

Prof. Nguyen

13 December 2024

# CS 405 Project Two Script Template

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

**VIDEO LINK:** **https://youtu.be/U5uMfsAE\_b8**

**Bryan Chan**

**22 December 2024**

**Project Two: Security Policy Presentation**

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hi everyone!  In this presentation, we are going to be presenting our proposal for a security policy, that is going to satisfy each and every one of *Green Pace*’s security needs. |
| **2** | The security policy contains the following standards:   1. Data Type 2. Data Value 3. String Correctness 4. SQL Injection 5. Memory Protection 6. Assertions 7. Exceptions 8. Correct Integer Type 9. Arrays 10. Input Output (closed file)   Each and every one of the coding standards will satisfy the project’s security needs.  In addition to this, the coding standards satsifes the fundamental security principle of “defense in depth”.  For instance, according to the Fortinet article, “What Is Defense In Depth?” (n.d.), defense in depth is:“[...]a strategy that leverages multiple security measures to protect an organization's assets.” (Fortinet, 2024). In other words, more than one security solution is used in conjunction, in order for the best security policy possible.  In order to fulfill the defense-in-depth, careful examination of the project’s security need is required. After this is done, a variety of coding standards are selected, in order to satisfy each and every one of the security needs. |
| **3** | Next, in this slide, we are presented with the project’s threat matrix. The threat matrix is divided into four categories: 1) likely, 2) probable, 3) unlikely, and 4) low priority.  Beginning with likely, there are three coding standards that possess a likely vulnerable category:   1. IDS-00-J: SQL Injection (P18) 2. MEM-30-C: Memory Protection (P18) 3. ERR-56-CPP: Exceptions (P9)   Next, for the probably category:   1. MSC-52-CPP: Data Value (P8) 2. ARR-36-C: Arrays (P8)   Then, when it comes to unlikely:   1. STD-001-CPP (HIGH) 2. EXP-58-CPP: Data Type (P4) 3. FIO-74-C: String Correctness (P6) 4. DCL-03-C: Assertions (P1) 5. INT-35-C: Correct Integer Type (P2) 6. FIO-46-C: Input Output (closed file) (P4)   Last but not least, out of every category, the standards with the lowest priority levels are:   1. DCL-03-C: Assertions (P1) 2. INT-35-C: Correct Integer Type (P2) |
| **4** | **Validate Input Data**   1. IDS-00-J: SQL Injection (P18) 2. ERR-56-CPP: Exceptions (P9) 3. STD-001-CPP (HIGH) 4. EXP-58-CPP: Data Type (P4) 5. INT-35-C: Correct Integer Type (P2)   **Heed Compiler Warnings**   1. MEM-30-C: Memory Protection (P18) 2. EXP-58-CPP: Data Type (P4)   **Architect and Design for Security Policies**   1. ERR-56-CPP: Exceptions (P9)   **Keep It Simple**   1. MSC-52-CPP: Data Value (P8)   **Default Deny**   1. MSC-52-CPP: Data Value (P8)   **Adhere to the Principle of Least Privilege**   1. IDS-00-J: SQL Injection (P18)   **Sanitize Data Sent to Other Systems**   1. ARR-36-C: Arrays (P8) 2. MEM-30-C: Memory Protection (P18) 3. FIO-74-C: String Correctness (P6) 4. FIO-46-C: Input Output (closed file) (P4)   **Practice Defense in Depth**   1. IDS-00-J: SQL Injection (P18)   **Use Effective Quality Assurance Techniques**   1. DCL-03-C: Assertions (P1) 2. ERR-56-CPP: Exceptions (P9)   **Adopt a Secure Coding Standard**   1. ERR-56-CPP: Exceptions (P9) 2. IDS-00-J: SQL Injection (P18) |
| **5** | **Data Type [EXP58-CPP.]**  - Pass an object of the correct type to va\_start  **Data Value [MSC-52-CPP]**  - Value-returning functions must return a value from all exit paths functions must return a value from all exit paths  **String Correctness [FIO-47-C. ]**  - Use valid format strings  **SQL Injection [IDS-00-J. ]**  - Prevent SQL injection  **Memory Protection [MEM-30-C ]**  - Do not access freed memory  **Assertions [DCL-03-C]**  - Use a static assertion to test the value of a constant expression  **Exceptions [ERR-56-CPP]**  - Guarantee exception safety  **Correct Integer Type**  - Use correct integer precisions  **Arrays [ARR-36-C]**  - Do not subtract or compare two pointers that do not refer to the same array  **Input Output (closed file) [FIO-46-C]**  - Do not access a closed file |
| **6** | **Encryption in flight:**  According to Sabrina Lupșan in the Cyscale article, “Types of Encryption for in Motion, in Use, at Rest Data” (2022), encryption at rest involves data that has arrived at its destination, but is not used (Lupșan, 2022). This policy can be applied by adding an encryption algorithm to encrypt data that has been received, but not used. Although the data is not used, its contents should still be encrypted, due to the risk of it being accessed due to a data breach.  **Encryption at rest**  - Sabrina Lupșan, in the previously mentioned Cyscale article (2022), describes encryption in motion (flight), as involving data that is moved from different locations (Lupșan, 2022). Encryption in flight can be implemented by encrypting data, regardless of its destination. It is possible for security attackers to intercept data when it is in transit. Therefore, every data that is traveling should be encrypted.  **Encryption in use**  Lupșan, in the same Cyscale article, writes that encryption in use deals with data that is accessed and used (Lupșan, 2022). Encryption in use will be implemented by encrypting data that is used after arriving successfully at its destination. This will prevent data from being accessed by unauthorized users. |
| **7** | **Authentication**  - According to the Fortinet article, “What Is Authentication, Authorization, And Accounting (AAA) Security?” (2024), the first part of the Triple-A framework, authentication, involves: “[…] a user providing information about who they are.” (Fortinet, 2024). Authentication can be implemented by providing every user with a token, which will serve as an identifier. This is needed in order to provide the users with a proof of identity, which will prevent impostors from accessing the owner’s account. Authentication is usually done whenever a new user has been created.  **Authorization**  - The previously mentioned Fortinet article (2024), describes authorization in the following: “[d]uring authorization, a user can be granted privileges to access certain areas of a network or system.” (Fortinet, 2024). This second part of the Triple-A framework will be implemented, by granting users with certain privileges. By doing so, unauthorized activity within the program is prevented.  **Accounting**  - The same Fortinet article (2024) writes that accounting involves keeping track of every user activity (Fortinet, 2024). Accounting can be implemented by having the program keep a log of a user’s every activity, whether they have logged in, or have accessed or made changes to a particular database. By accounting for every time a user has accessed the system, the system administrator is able to monitor for any suspicious activity, as well as make a determination on an attacker’s identity. |
| **8** | Unit testing |
| **9** | Automation summary |
| **10** | Within the established DevSecOps process, there are eight steps: 1) Assess and plan, 2) Design, 3) Build, 4) Verify and test, 5) Transition and health check, 6) Monitor and detect, 7) Respond, and 8) Maintain and stabilize. All of this is found in the image shown above. All eight steps are used within the DevSecOps framework, in order to create a safe and reliable application.  Automation is a very important tool, that can be utilized in any phase of the software development life cycle (SDLC). In this context, automation would greatly improve the workflow of Green Pace’s DevOps process. Automation would be very effective in the fourth phase of the DevOps process (Verify and test), which pertains to the verification and testing of the program. The specific tool for this assignment, is the Polyspace Bug Finder, which will prevent two pointers from making comparisons or subtractions if they refer to the same array. |
| **11** | For automation to be effectively used for testing, the developer can use unit tests.  Unit tests allows the developer to test the program in a number of use cases, in order to test for its performance. If done manually, the process can take a long time. This can result in some crucial areas being overlooked. By having an automated testing system, the developer can verify that the program meets all of its requirements, by testing for every possible unit case. The importance of automation tools can be found in the tenth coding standard: “[ARR-36-C]: Do not subtract or compare two pointers that do not refer to the same array”. |
| **12** | - If the implementation of the security measure is delayed, then the system runs the risk of being infiltrated.  - A security breach could result in financial loss, as well as the loss of personal data.  - As a result, the system’s/company’s reputation is damaged |
| **13** | Recommendations |
| **14** | Conclusions |
| **15** | References  Confluence. (2024). Sei cert C coding standard. Retrieved December 22, 2024, from https://wiki.sei.cmu.edu/confluence/collector/pages.action?key=c  Fortinet. (n.d.). What Is Authentication, Authorization, And Accounting (AAA) Security? Retrieved December 22, 2024, from https://www.fortinet.com/resources/cyberglossary/aaa-security  Fortinet. (n.d.). What Is Defense In Depth? Retrieved December 22, 2024, from https://www.fortinet.com/resources/cyberglossary/defense-in-depth  Lupșan, S. (2022, August 9). Types of encryption for in motion, in use, at rest data. Cyscale. https://cyscale.com/blog/types-of-encryption/ |