# CS 405 Project Two Script Template

John Brungard | CS405 | Project Presentation | 10/15/2023 | Link: https://youtu.be/cCEOwQy1dE4

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is John Brungard and I have been asked to present the Green Pace security policy guide and to provide implementation guidelines and recommendations for maintaining it in the future. The developers have been employing best practices and, as the team grows, it’s critical that everyone remains in sync with principles and best practices. Let’s jump right into it. |
| **2** | Defense in Depth involves adding multiple security layers to enhance the general security of your system. The logic is that if one layer of defense is insufficient for the threat that another layer can prevent an inadequacy in the security design from becoming an exploit. |
| **3** | The threats in this matrix range from priority levels of 1 (low) to 18 (high). Higher priority threats should be addressed more immediately than threats of low priority although they will also need to be addressed too.  Threats also are categorized as being likely or probable to happen as well as unlikely. Threats more likely to happen need to be monitored carefully and require a remediation plan that may be more intricate than threats that are unlikely. |
| **4** | Now we will look at the ten Principles of Security and how they align with our collected coding standards:   * Validate Input Data aligns with standard IDS-000-J: Prevent SQL injection. * Heed Compiler Warnings aligns with standard ERR-051-CPP: Handle all exceptions. * Architect and Design for Security Policies * Keep It Simple aligns with standard MEM-050-CPP: Do not access freed memory and standard CTR-053-CPP: Use valid iterator ranges * Default Deny * Adhere to the Principle of Least Privilege      * Sanitize Data sent to Other Systems aligns with standard STR-002-C: Sanitize data passed to complex subsystems. * Practice Defense In Depth aligns with standard MSC-011-C: Incorporate diagnostic tests using assertions. * Use Effective Quality Assurance Techniques aligns with standard INT-030-C: Ensure that unsigned integer operations do not wrap and standard STR-031-C: Characters and Strings- Guarantee that storage for strings has sufficient space for character data and the null terminator. * Adopt a Secure Coding Standard aligns with standard INT-031-C: Ensure that integer conversions do not result in lost or misinterpreted data and standard EXP-000-J: Do not ignore values returned by methods. |
| **5** | Here are the coding standards to be followed collectively:   1. Ensure that integer conversions do not result in lost or misinterpreted data. 2. Ensure that unsigned integer operations do not wrap. 3. Characters and Strings- Guarantee that storage for strings has sufficient space for character data and the null terminator. 4. Prevent SQL injection. 5. Do not access freed memory. 6. Incorporate diagnostic tests using assertions 7. Handle all exceptions 8. Sanitize data passed to complex subsystems. 9. Do not ignore values returned by methods. 10. Use valid iterator ranges |
| **6** | These are the three types of encryption policies which include Encryption in rest, Encryption at flight, and Encryption in use.  Encryption in rest refers to encrypted data that is not being used. An example of this is passwords in a database or other sensitive information that may not always be in use or hard drives.  Encryption at flight is the process of encrypting data that is leaving or entering a specific part of the application. An example of this is sending a message that uses asymmetric encryption.  Encryption in use is the encryption of security of data that is being used or accessed. An example of this is using a public key to decrypt a message. |
| **7** | Next, we will review the Triple-A Framework that consists of Authentication, Authorization and Accounting:  Authentication is the process of identifying the user trying to access the system. This can apply to user logins where a user supplies a username or password, and the system checks against those credentials.  Authorization is where the system determines the user’s level of access depending on their role within the system. An example of this are the available file access and privileges that an admin has on a website compared to a new user.  Accounting is the process of keeping documentation and accountability for the system and its data. This can include changes to the database and the addition of new users. |
| **8** | Additionally, we will view the unit tests and how they can benefit exploring potential faults within our system.  The first test is a negative: We explore the question “Is the collection size 10 even though 15 values were added to the vector?”  We are specifically looking at the add\_entries() function and push\_back function for this example and checking that they can be used in combination. |
| **9** | The second test is negative as well and asks, “Does accessing an element past the bounds of a vector throw an exception?”  With this example we try to access the 10th element of a vector containing 9 elements. This should expect to throw an exception. |
| **10** | We also explore the assign() function with the question: Does the assign() function successfully alter the vector’s size and values? This is a positive test.  We assign the collection with an element containing 50 and our tests check both the size and the value after the assign() feature is used. |
| **11** | We look at Vector resizing with the question: “Does the vector correctly resize after a decrease to its collection?” This is also a positive test.  After adding entries, we resize the vector to 0 and test whether the size was adjusted accordingly. |
| **12** | The next unit test is much like the last but this time with a resize increase and the question “Does the vector correctly resize after an increase to its collection?”  We use a placeholder variable in initialSize after adding an entry then we resize the vector to 2 and test that it is greater than the placeholder variable. |
| **13** | The last unit test pertains to capacity with the question “Does adding entries to the collection affect the capacity?”  Here we add 15 entries and recognize whether the capacity is greater than 0, 1, 5, and 10. |
| **14** | This slide verifies the unit tests we have just covered as well as a few others which all ran successfully. |
| **15** | This slide portrays the DevSecOps pipeline. Please take a minute to review each of the phases in both pre- and post-production stages.  An automation factor can be established in the verify and test phase. We can implement powerful unit testing tools such as Junit5 or something like this such as QUnit.  Furthermore, we can identify the dependencies that will be used in the application using static testing and update our list of dependency vulnerabilities to reduce potential risks.  Moving onto the production phase, we may automate our penetration tests. In specifically the domain of monitoring and detection, we can introduce log notifications and document them for future reference. |
| **16** | Instead of tacking security on to the end of projects with point-in-time audits and penetration tests after code is deployed, DevSecOps bakes security in at every step of the process. This includes building, testing, and deploying software where security was often an afterthought.  Here are some tools that assist in DevSecOps:  Parasoft C/C++ Test: An automated testing suite (DAST & SAST). Used in the verify & test stage in pre-production. Used during the transition and health check as well as the maintain and stabilize stage in post-production. (https://www.parasoft.com/)  Astrée is a static code analyzer that proves the absence of run­time errors and invalid con­current behavior in safety-critical software written or generated in C or C++. Being a static code analyzer, it is used in the design stage in pre-production and the monitor & check in post-production.  Jit is a continuous security platform that provides security-plan-as-code (SaC) for your DevOps pipeline. It acts as an orchestration layer that makes installing, viewing, and managing all open-source security tools for your software easier. Would be used in the build section in pre-production, and the respond section in post-production.  Craft.io is an innovative and user-friendly product backlog tool designed to streamline the product development process and enhance collaboration among teams. With Craft.io, product managers and development teams can efficiently manage their product roadmap, prioritize features, and track progress in a centralized platform. Used during the assessment and planning stage in pre-production. |
| **17** | We now look at the consequences of waiting to apply security at the end versus acting now and applying it during the earlier stages of the SDLC.  When we wait, it:   * Can create a sense of distrust amongst clients due to loss or invasion of privacy. * May allow time for attackers to manipulate, steal, or delete important data. * Could create more cost in the repair of an attacked system.   When we act, it:   * May prevent emerging or further damage of reputation involving your organization. * Mitigates any possible or future damage to the system. * Can save costs by testing and procedures compared to repairs.     An example I will share of a consequence of not being thorough with acting now on security is the Toyota vs. Zveare case:   * Toyota recently learned of an issue with its supplier portal, through which a hacker could access email accounts, documents and other confidential information. (Teague, 2023). * The hacker generated a web token in the portal using a Toyota email address in which the system gave them authentication without use of a password. * The hacker notified Toyota of the vulnerability in November of 2022. |
| **18** | Based on what has been reviewed so far, here are some recommendations to address potential gaps in the security policy:   * Consider addressing the most probable threats that contain the highest priority levels first. Explore remediation options and costs during pre-production. * Further investigate standards that will align with principles not yet addressed such as Default Deny having more examples of compliant and non-compliant code for the development process. * Conduct more unit tests outside of vectors operations for finding faults within code such as function verification. * Conduct a risk assessment that is well-documented and outlines a plan during the SDLC. * Identify areas in both pre and postproduction where Encryption and Triple A policies would be best applied. |
| **19** | We will conclude this presentation with a few key points of what needs done pertaining to security in the future:   * Validate user input, which may include checking the length of Strings. * Heed all compiler warnings and consider enabling the highest warning level. * Keep it simple in your coding to increase readability and maintenance. * Deny access by default and allow only authorized users and events. * Give each user’s role in your system the least amount of privilege they need to perform their job successfully. * Ensure that your system has multiple layers of security for a decreased risk of a breach. * Explore other quality assurance techniques such as hiring a QA Analyst or purchasing/getting more automation tools.   Questions or concerns will be available after the presentation. Thank you and have a great day! |