# Brian Engel

6/23/2024

# CS 405 Project Two Script Template

[Link for MP4](https://youtu.be/QLAOLDn1dpg)

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hi, I’m Brian Engel and I’ll be going over a new security policy for Green Pace today. |
| **2** | This security policy utilizes a defense in depth strategy. While not every layer will cover every vulnerability, when a vulnerability passes through a layer, it is caught by subsequent layers and will ensure a secure system. |
| **3** | A variety of vulnerabilities were decided upon to start off with. They have been arranged into an order of descending priority to be dealt with. The priority is based off of how common the vulnerability is and how much damage it is likely to cause. One thing to note is that just because a threat is unlikely to happen it still must be corrected. We do not want any holes in our defense in depth strategy. |
| **4** | Every part of the security policy we put in place should be looking toward these 10 principles. Validating input ensures that input is valid and will not cause errors or unexpected behavior. Heed compiler warnings because if it pops up in a compiler warning, it will most likely pop up again in static testing and be harder to fix at that point. Architect and Design for Security Policies is basically not to staple security on at the end. Keep it simple because unneededly complex systems are hard to understand, secure, and maintain. Default deny is to deny access unless the user is explicitly granted access. The principle of least privilege is to give users access to the minimum level of access needed to perform their job. Even though other systems should validate their input, sanitize data sent to other systems. This ensures it is valid. Practice defense in depth ensures there are multiple layers to catch any vulnerabilities. Use effective quality assurance techniques like unit testing, code reviews, and static analysis to catch errors. Adopting a secure coding standard is important to make sure all developers have the same high standards for security. If you have any question on what they are or how they apply they are defined on pages 2 and 3 of the security policy. |
| **5** | These are the initial ten coding standards chosen for the security policy. They represent a diverse group of vulnerabilities and are in order of priority. These are all found in the security policy, and I will go over the first one to show what all is covered. The first on the list is sanitize data passed to complex subsystems. In the security policy this one starts on page 14. It begins with what the coding standard deals with, in this case SQL injection, the label STR-002-CLG, the actual name of the standard, and a description of the standard. Then there is an example of noncompliant code, followed by compliant code. It also shows what security principles are affected by this standard, in this case validate input data and sanitize data sent to other systems. Then there is a breakdown on severity and likeliness, in this case high and likely, which give it a priority level that is used to determine the position in the overall list, in this case P18 which is very high priority and the reason it is on the top of the list. After this is a list of tools that can be used for automated testing for the particular vulnerability. All of the other standards have entries in the security policy detailing the same information. |
| **6** | The next part of our defense in depth is the encryption policies. Any personal or sensitive data that is at rest in a database or hard drive needs to be encrypted. It also needs to be encrypted before sending internally or over the internet. Finally, it is impossible to use encrypted data for manipulation, since that is the point of encryption in the first place, but when using it, it shouldn’t be saved into a variable unencrypted since the memory for the variable can persist longer than expected. Instead, it should be decrypted, manipulated, and re-encrypted in line if possible. |
| **7** | Next in our defense in depth are the Triple-A policies. Authentication ensures that the person accessing the system is who they claim to be. This process typically involves verifying a user's identity through passwords, biometrics, or other authentication methods. Effective authentication is critical for preventing unauthorized access. Authorization determines what resources and systems a person is allowed to access once they are authenticated. It sets the permissions and access levels for users, ensuring that they can only interact with the data and functionalities necessary for their role. Accounting tracks and logs users in the system. It involves recording user activities, which helps in monitoring usage patterns, detecting anomalies, and auditing system access. This is essential for maintaining security and accountability within the system. |
| **8** | Next in our defense in depth system is unit testing. First, test names should clearly reflect the purpose of the test. A descriptive test name makes it easier to understand what is being tested and what the expected outcome is. Unit tests should avoid dependencies and other outside systems. Tests should be isolated and focus solely on the functionality of the unit being tested. This ensures that tests are reliable and not affected by external factors. Tests should be independent of each other. Each test should set up its own context and not rely on the results or state of other tests. This independence prevents cascading failures and makes it easier to pinpoint issues. Positive tests are designed to prove that the code functions as expected under normal conditions. These tests ensure that the code correctly handles valid inputs and performs the intended actions. Negative tests, on the other hand, are designed to prove that the code correctly handles error conditions or invalid inputs. These tests ensure that the system responds appropriately to unexpected or erroneous situations. |
| **9** | This test checks if the collection can be resized to 0 as the name implies. For all the tests we check to make sure the collection is empty and then set it up to ensure the tests are independent of each other, and in this one we check that the collection is empty and then add 10 entries. We then check that the size is 10, and then resize the collection to 0, and verify the collection is now at 0. This is a positive test since it is given a valid argument and expected to return a value and not an error or exception. |
| **10** | This test checks if the clear erases the collection as the name implies. Like the last test we ensure the collection is empty and then add 10 entries. We make sure the size is 10 and then clear the collection and then make sure it is empty. Once again this is a positive test. |
| **11** | This test checks if an out of range exception is thrown. As with the other tests we set it up by ensuring it is empty, adding 10 entries, and checking to make sure the size is 10. This is a negative test though and we make sure it fails by trying to access an element that is out of range at 20. This throws the proper exception and thus the test passes. |
| **12** | This test checks if an exception is thrown with a resize to a negative size. This test we just make sure that the collection is empty first, and then try to resize it to -1. A length\_error exception is thrown for this negative test, so the test passes. |
| **13** | The DevSecOps pipeline diagram shows the flow of development. Security should be integrated into the assess and plan, design, and build phases. Then in the verify and test phase automation can be used with unit testing, static testing, and dynamic testing for the coding standards that we implemented in this policy. Once the system is deployed, we can use automation to monitor and detect any threats to the system. If a vulnerability is found, we respond to the attack, maintain and stabilize the system, and loop back to respond to the new vulnerability that was discovered and start the process all over again. |
| **14** | Incorporating security into our DevOps pipeline, commonly referred to as DevSecOps, is essential for building secure and resilient systems. This process can be streamlined with the use of various external tools that automate and simplify security tasks. Unit testing is critical for catching vulnerabilities early in the development process. By thoroughly testing individual components of our code, we can identify and address security issues before they escalate. Astrée and Parasoft C/C++test can be automated in the verify and test phase to perform static and dynamic analysis while tools such as Prometheus with Exporters, Dynatrace, Boost.log and spdlog can be automated in the monitor and detect phase. These tools are in no way an exhaustive list and the best tool to detect each vulnerability should be used. |
| **15** | Not having a security policy leaves systems and data vulnerable to attacks. Without clear guidelines, there are gaps in security practices, making it easier for malicious actors to exploit weaknesses. Lack of a security policy may lead to non-compliance with industry regulations and legal requirements. This can result in hefty fines, legal actions, and damage to the organization's reputation. Without a security policy, there are often inconsistent practices across the organization. Employees may not be aware of the best security practices, leading to a higher risk of security breaches.  Having a security policy provides clear guidelines and procedures to mitigate system and data vulnerabilities. It ensures that there are standardized practices in place to protect against potential threats. With a security policy, employees and stakeholders are aware of the organization's dedication to security. It emphasizes the importance of not only system security but also the protection of personal information.  The primary disadvantage of implementing a DevSecOps system is the initial cost and time required. Setting up the necessary infrastructure, training employees, and integrating security into the development process can be resource-intensive. However, these initial investments are outweighed by the long-term benefits of a secure and resilient system. |
| **16** | One significant gap is the absence of a response plan to deal with security lapses. It's crucial to have a well-defined incident response plan that outlines the steps to take when a security breach occurs. This ensures that the organization can quickly and effectively mitigate the impact of any security incident. We can learn a lesson from the Target data breech from 2013. Their monitoring software (FireEye) alerted Target staff in Bangalore, India, who in turn notified staff in Minneapolis: but no action was taken. Despite the fact that Target reportedly spent a large sum on security technology utilizing encryption, their data was accessed in memory where it was unencrypted, another reason to be extremely careful with encryption in use. The lack of a response ended up costing Target an estimated $250 million. Another gap is our current security policy is primarily focused on employees in technical positions. However, non-technical employees also play a crucial role in maintaining security. We should implement training programs to educate all employees about common threats like phishing scams and other mitigatable exploits. As technology advances, new vulnerabilities will emerge. It's essential to continuously update our security policy to address these new threats. Regular reviews and updates of the security policy will help us stay ahead of potential risks. Rules and regulations regarding security are constantly evolving. Our security policy should be flexible and regularly updated to reflect new laws and regulations. This will ensure that we remain compliant and avoid potential legal and financial repercussions. If an attack occurs, it's crucial to analyze the incident and update the security policy to cover the exploited vulnerability. This reactive measure helps prevent similar attacks in the future and strengthens our overall security framework. |
| **17** | In conclusion, to ensure ongoing compliance with our security policy, both now and in the future, we should always adhere to the following practices:  Keep the 10 core security principles in mind and use layers of defense to mitigate vulnerabilities.  Adhere to the coding standards that are set in place to ensure consistency, security, and quality in our codebase.  Implement encryption at rest, in flight, and in use to protect sensitive data at all stages of its lifecycle.  Keep Authentication, Authorization, and Accounting in mind. These fundamental principles ensure that users are who they claim to be, have access to only the resources they are permitted to, and their actions are tracked and logged.  Use unit testing to catch vulnerabilities early in the development process and leverage automation tools to streamline security practices and ensure consistent enforcement of security policies.  Regularly review and update the security policy to address new vulnerabilities, changes in technology, and evolving regulatory requirements. Continuous improvement and vigilance are key to maintaining robust security. |
| **18** | References. |