# CS 405 Project Two Script

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

Andres Sanchez

10/20/2024

Project Two: Security Policy Presentation

[**https://youtu.be/9Q5\_sACXhks**](https://youtu.be/9Q5_sACXhks)

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Welcome everyone, my name is Andres, and today I will be presenting the Green Pace Security Policy Guide. I’ll be walking you through some of the policy’s core elements and its importance, in order to offer guidelines and recommendations for maintaining it in the future.  Before we begin, I want to mention that this policy offers a pretty comprehensive framework to secure all aspects of software development. This ensures security principles are embedded into our suggested best practices through designing, coding and deployment.    By the end, my primary goal is to ensure Green Pace can proactively safeguard data, address security risks, and comply with industry regulations by encouraging the adoption of best practices throughout the organization. |
| **2** | Let's start by discussing Defense-in-Depth, a critical security best practice. Defense-in-Depth means applying multiple layers of security controls across the entire system to ensure that if one layer fails, the others will still protect the organization. This layered approach doesn’t just affect one piece of security but instead extends across Green Pace in its network security, application security, and data protection. Our security policy integrates this principle by embedding secure coding practices with encryption, access controls, and monitoring throughout the software development lifecycle. By using this strategy, we ensure that any potential breach or attack is contained & mitigated quickly to minimize damage. Some of those coding practices that we will be talking about today is authorization, authentication, auditing, encryption, and other measures to ensure we are building those best practices. |
| **3** | Now, let's walk through the threat’s matrix. A table in which we categorize some of our standards into Likely, Priority, Low Priority, and Unlikely.  Likely are those most probable for vulnerabilities that can significantly impact security if not addressed. Examples such are standard 2 and standard 3 for Integer Overflow & String Memory can both be common and create large exploits in our systems. That said, its recommended that these are focused on first due their potential likelihood and impactfulness.  Priority Risks are those that are less frequent but can still have a high impact when they do occur. For example, standard four, SQL Injections and standard ten, System Function Misuse can cause severe data breaches if exploited.  Low Priority Risks on the other hand are those such as standard seven which is Exception Handling, that may be less impactful but should still be resolved once higher-priority vulnerabilities are mitigated.  Unlikely like standard six, Assertions for Debugging, are rare and have minimal impact. While they are lower priority, it’s still important to maintain best practices by addressing them. |
| **4** | Next we move to our 10 core security principles:  We first start with Validate Input Data. This ensures the input data entered by external sources, such as users, meets the predefined criteria of the program. This involves checking data for the expected format, type and constraint before allowing it to interact with the rest of the application. The intent is to prevent attackers from exploiting the system or accessing sensitive information. As such it also extends to APIs & involves notifying users if their input does not meet the expected requirements. This ensures data entered by users is reliable and secure, preventing issues like data corruption, injection attacks, and buffer overflows. Under this principle, we apply the Data Value, String Correctness, and SQL Injection Coding Standards.  Next is Heed Compiler Warning. This alerts developers to potential issues in code that may not result in compiling errors but could lead to risk in unexpected behavior, performance or security vulnerability. It’s important to equally consider these warning for preventing further risks. It includes standards such as Data Type and Memory Protection.  Architect & Design for Security Policies means integrating security considerations to every step of the software development process, including but not limited to planning, developing, testing, and etc. This approach allows security to be integrated proactively rather than reactively.  We then want to make sure we are keeping It simple. This emphasizes the importance to minimize complexity to reduce potential vulnerabilities and make security easier to manage. Lower complexity means less time required for testing, reducing encountering errors and speeding up development. This includes standard ten for environment.  Default Deny & Adhere to the Principle of Least Privilege are both principles that minimize risks by ensuring processes operate with the minimum required privileges. In which standard ten for environment also fits under this category.  Sanitize Data sent to other System involves cleaning, filtering and transforming data that may be harmful or used to exploit vulnerabilities in set systems. This may include removing special characters or validating format of inputs. The goal is to prevent injection attacks by external systems. In this case, the standard for SQL injection fits.  Defense in Depth as we briefly talked about is the practice of including multiple defensive strategies to protect data & systems and mitigate risk of exploitable vulnerability. Additional layers of defense can limit the damage in successful exploits. This includes standards for both error handling and memory protection.  Then is using Effective Quality Assurance Techniques which ensures software is reliable, secure and meets any required standard before production. This process includes various testing such as penetration testing, security audits and other vulnerability assessment. The standard of data value fitting under this principle  Finally be not least is Adopt a Secure Coding Standard. This principle ensures that software is developed with security in mind and is tailored to the specific language and platform in use. This approach provides guidelines to best enforce best practices. Many standards which fit under here are such as Data Type, Memory Protection, Assertions, Exceptions and Error Handling. |
| **5** | Now, let’s talk about my personal ranking of coding standards based on their importance. This is based on general practices and my personal considerations.  My number one spot is guaranteeing storage for strings has sufficient space for character data and null terminator. Ensuring enough memory is allocated for storing strings can prevent buffer overflow which can be highly impactful and highly likely. This can allow attackers to exploit this vulnerability to execute arbitrary code.  Up next is preventing SQL injections which is another one that's highly likely and can lead to severe consequences. This focuses on the importance of sanitizing and validating input before passing it to the system. Failure to do so can lead to malicious data running through the program which can result in an injection attack.  My 3rd standard is not accessing free memory. This is important as memory access issues can cause system crashes and critical security vulnerabilities. This is seen as a very common way to cause a denial-of-service attack.  4th is detecting and handling standard library errors which can prevent system failures and vulnerabilities. It is important to detect and handle these errors as failure to do so can lead to unpredictable results including abnormal program termination, denial of service attacks or in some cases allow attackers to run arbitrary code.  5th ensuring the operations of signed integers do not result in an overflow. We emphasize the importance of checking that these operations do not cause overflow as signed integers can lead to unexpected behavior. In C++, since signed integer overflow is considered undefined this means the compiler does not always handle it as predicted. This can lead to incorrect data processing.  6 is properly deallocating dynamically allocated resources. This standard focuses on ensuring allocated memory and resources are properly deallocated to avoid memory leaks and maintain efficient resource utilization. This is important as memory management issues can lead to system instability which can be exploited.  #7 is not the defining C-style variadic functions. This introduces potential type safety issues leading to crashes or even exploitations. Instead, we recommend the use of initializer lists or other function types that provide type checking  Next and #8 is not calling the system function. The system function allows execution of commands in the operating system which introduce security risks and compromises the local system integrity. This can cause security vulnerabilities as it can introduce command injection risks.  #9 is handling all exceptions. Proper exception handling is important for maintaining stability and ensuring all exceptions thrown by the program are caught by the exception handler. Allowing the application to terminate abnormally can result to resources not being freed, closed and so on. It is often a vector for denial-of-service attacks.  Finally, #10 is using assert only for debugging purposes. I find this to be slightly less impactful when properly handled in production. This focuses on the importance of using assert only for debugging purposes and should be turned off before the code is deployed to production. Failing to do so can leak internal logic which can then be exploited by attackers to discover vulnerabilities. |
| **6** | Now time we have some fun, with Encryption.  Encryption is an essential part of protecting sensitive data at every stage of its lifecycle. Our policy covers three critical areas that enforces encryption.  First is Encryption at Rest which helps protect data stored in databases, disks, or backups. Advanced Encryption Standards (AES) ensures that data remains unreadable even if its stolen. The reason this is crucial, is it allows our teams to focus our strategy on encryption more than trying to protect all the data.  Encryption in Transit (also known as Encryption in Flight) is the practice of securing data as it moves between systems or networks. This is key as data is most risk of being exposed during this cycle. Using TLS (Transport Layer Security) or VPNs can be especially important for securing sensitive data in emails and/or web traffic.  Finally, we have Encryption in Use which safeguards data being actively processed by applications. Using techniques like homomorphic encryption ensures sensitive data is protected even in memory. |
| **7** | Next is our Triple-A framework, which ensures that only authorized individuals have access to our data, systems and resources.  Authentication is the process of verifying user identities through passwords, multi-factor authentication, or biometric methods to ensure only trusted users gain access. The goal is to prevent unauthorized access our sensitive information.  Authorization is determining what actions authenticated users can perform based on the controls we have set in place. This ensures users only have access to resources need and is key to enforcing our ‘Default Deny’ security principle.  Accounting then maintains detailed logs of user activity, such as login attempts and data changes, to detect and respond to suspicious behavior quickly. This is key as it help organizations prevent & respond to incidents. |
| **8** | In this activity, we had to perform unit testing on some of the code that was provided. Unit testing is a software testing technique that tests individual components of software in isolations to make sure they are functioning correctly.  This is important as it helps detect bugs early and improves the quality of our codebase. Furthermore, it helps us feel confident that the code behaves as expected. The first example, tests if using the clear function erases our collection stored in a data structure. You can see we first test if the collection is empty, add a few entries, verify the collection size is equal to the size we added, use the clear function and verify the collection is empty. Then underneath we see the test run successfully. |
| **9** | In the next example, we use a negative test which means we purposely want the program to fail. This example tries to reserve more than the max size of the collection. As you can see from the unit testing, its throws an exception successfully. |
| **10** | The third test is checking if resizing the collection also increases it. Again, we check if the collection is empty, resize the collection and then check if the size was increased. |
| **11** | Our final test is then verifying that the max size is greater than the number of entries we put into the collection. We did this for 0, 1, 5 & 10 entries which all ran successfully. |
| **12** | Next let’s take a deep dive into Automation Summary. Taking a look at the diagram in the right, this represents the process of code creation and implementation in a figure eight pattern with the left side being pre-production and the right-side showing production. We use this diagram to visualize how we can apply the security standards we discussed across multiple areas within the existing DevOps Framework. That said, lets break it down by starting with assessing and planning. This portion helps us visualize threats that may encounter by including a threat landscape. We then move on to designing, includes a test-driven design which follow best practices. Its then important while building, you ensure you develop secure code and then you verify it by testing vulnerability with security testing. We then move on to production, where you transition and perform health checks. At this point you configure and deploy security settings and conduct penetration testing. After is monitoring and detecting where we look for logs and alerts that may indicate an intrusion. If one is found, you respond by blocking the attack and then revert to maintaining and stabilizing. As such this diagram can also help think on how using automation techniques can streamline some of the security tasks defined in this policy. Enhancing the efficiency and security. |
| **13** | DevSecOps, short for Development Security Operations, refers to the concept of making software security a core part of the overall software development process. This is what we explained in the last slide. In the left, we listed some tools that can be used by Green Pace in order to automate some of the security processes in the pipeline. |
| **14** | Now that we have gone through our policies and frameworks, lets summarize the risks of waiting and the benefits of acting now.  The risks of waiting include increased exposure to common attacks and leaving systems vulnerable. Delayed responses to breaches can also result in data loss and reputational harm. Additionally, non-compliance with regulations may lead to fines and other penalties.  On the other hand, acting now allows us to proactively mitigate risks by securing systems before threats arise. Faster responses help reduce financial losses and operational disruptions. We also ensure compliance with industry standards, minimizing insider threats. Early detection of vulnerabilities saves time, resources, and effort in the long run, making our security practices more efficient and effective. |
| **15** | While the existing policy provides a solid foundation, there are still a few gaps that could expose Green Pace to potential risks. One of these, is the lack of detailed implementation for a Defense in Depth strategy. Improvements should outline each layer of Defense in Depth across the entire software development process.  Another gap is the absence of a comprehensive Incident Response Plan (IRP). Clear guidance for managing breaches or attacks is essential, and improvements should include an up-to-date disaster recovery plan to mitigate potential damage effectively.  The policy also lacks standards for consistent enforcement. Improvements are needed to provide more detail on how policies and standards are applied in day-to-day operations and how compliance is monitored and enforced.  Lastly, the policy could be strengthened by adding more detail on access and audit controls. Future iterations should clarify how company data is categorized and be more specific on who has authority over that data.  Looking at history, two notable examples highlight the consequences of security gaps. Equifax experienced a major data breach, and due to a delayed incident response, the company incurred fines and damages totaling 700 million dollars in settlements. In the case of LifeLabs, the absence of a Defense in Depth strategy and missing security policies led to a breach of sensitive customer information, resulting in a 1.2 billion dollar lawsuit. These examples demonstrate the importance of proactive security measures and the need for robust policies to protect against such outcomes. |
| **16** | To conclude, adopting a security-first mindset, aligning with ISO standards, and following NIST guidelines will help Green Pace develop a strong security framework. These are what we believe are the focal point of implementing a secure foundation. These measures will ensure that we can detect, respond to, and recover from security threats effectively. Thank you for your time and will now be open for any questions. |