# CS 405 Project Two Script - Joseph Dengler

| **Slide Number** | **Narrative -** [**https://youtu.be/r1R7pjJBipw**](https://youtu.be/r1R7pjJBipw) |
| --- | --- |
| **1** | "Welcome to the Green Pace Security Policy Presentation. I am Joseph Dengler, and today I will guide you through our security standards and best practices that are essential in maintaining a secure coding and architecture framework within Green Pace." |
| **2** | "The Green Pace security policy was created to support the defense-in-depth strategy, ensuring that we have multiple layers of defense to address the various security risks. This policy not only covers technical vulnerabilities but also helps establish standards and procedures that enhance our coding and architectural best practices. It was necessary to standardize our existing practices as the team grows and new developers join." |
| **3** | “In this threat matrix, we categorize various security risks based on both their likelihood of occurring and the severity of the threat they pose.  In the Priority quadrant, we place vulnerabilities that require immediate attention, such as unencrypted sensitive data in transit, which poses a high risk to the security of user information.  The Likely quadrant includes vulnerabilities that are probable but may not be critical, such as code injection from unsanitized inputs, which is common but manageable with proper safeguards.  In the Low Priority section, we classify risks that are not immediate concerns, such as missing comments in the code, which might slow down development but don’t pose a direct security risk.  Finally, in the Unlikely category, we place vulnerabilities that are rare or less likely to occur, such as the use of obsolete code libraries without known vulnerabilities.” |
| **4** | On this slide, we are showing the 10 key security principles and aligning them with the coding standards that correspond to each principle.  Validate Input Data: Ensures all input is sanitized and validated before processing. Related coding standards: STD-009-VAL and STD-003-STR.  Heed Compiler Warnings: Addressing compiler warnings early helps avoid vulnerabilities. Related coding standards: STD-001-CPP and STD-002-INT.  Architect and Design for Security Policies: Designing for security from the start, implementing policies like encryption and access control. Related coding standards: STD-005-MEM and STD-006-ASR.  Keep it Simple: Simplicity reduces the attack surface by limiting complexity. Related coding standard: STD-004-SQL.  Default Deny: Restrict access to resources unless explicitly allowed. Related coding standard: STD-008-RSC.  Adhere to the Principle of Least Privilege: Minimize access to only necessary resources. Related coding standard: STD-010-ERR.  Sanitize Data Sent to Other Systems: Ensuring outgoing data is secure. Related coding standards: STD-004-SQL and STD-003-STR.  Practice Defense in Depth: Layering multiple security controls. Related coding standards: STD-005-MEM and STD-006-ASR.  Use Effective Quality Assurance Techniques: Continuous testing and validation for potential vulnerabilities. Related coding standards: STD-007-EXC and STD-001-CPP.  Adopt a Secure Coding Standard: Secure coding guidelines followed across all development efforts. Related coding standards: All standards apply. |
| **5** | On this slide, we show the 10 coding standards ranked by priority. These are organized based on the severity and likelihood of exploitation.  STD-001-CPP: Focuses on C++ vulnerabilities, especially in memory management and object handling.  STD-002-INT: Prevents integer overflows and underflows which could lead to unpredictable behavior.  STD-003-STR: Handles string-related vulnerabilities, especially buffer overflows.  STD-004-SQL: Mitigates SQL injection vulnerabilities through query sanitization and validation.  STD-005-MEM: Manages memory allocations to prevent buffer overflows and memory leaks.  STD-006-ASR: Implements Address Space Layout Randomization to reduce exploit predictability.  STD-007-EXC: Ensures proper exception handling to prevent program crashes and vulnerabilities.  STD-008-RSC: Ensures secure resource management to prevent unauthorized access and leaks.  STD-009-VAL: Validates all external input to ensure it is correctly processed.  STD-010-ERR: Manages error and logging to avoid exposing sensitive information.  We prioritize these standards based on their impact on security and their potential for exploitation. Memory, input validation, and string handling take higher priority due to their common exploitation in attacks. Error management and logging are essential but rank lower in terms of immediate security risks. |
| **6** | "In this slide, we cover the Green Pace encryption policies, ensuring data protection throughout its lifecycle:  Encryption in Flight refers to protecting data while it is being transmitted over networks. This prevents attackers from intercepting sensitive information during communication. We utilize industry-standard protocols like TLS (Transport Layer Security) to safeguard data in transit.  Encryption at Rest ensures that stored data, whether on hard drives, cloud storage, or databases, remains secure from unauthorized access. Even if physical storage devices are compromised, the encrypted data cannot be read without the proper decryption keys. We employ AES (Advanced Encryption Standard) as our encryption method of choice for securing data at rest.  Encryption in Use is critical for protecting data while it's actively being processed. During this phase, we leverage techniques like homomorphic encryption and secure enclaves to ensure that sensitive data remains secure even as it is being used in operations or computations.  Together, these encryption practices ensure comprehensive protection of data throughout its lifecycle, reinforcing our defense-in-depth strategy." |
| **7** | "At Green Pace, our security policy follows the Triple-A framework to ensure that users are properly identified, their actions are authorized, and their activity is monitored.  Authentication: This process verifies the identity of users before they are granted access to systems. We employ multi-factor authentication (MFA) as a best practice, combining something the user knows (password), something the user has (security token), and something the user is (biometrics). This helps to secure access to sensitive resources and prevent unauthorized logins.  Authorization: Once authenticated, authorization policies determine what a user is allowed to do. At Green Pace, we implement Role-Based Access Control (RBAC) to ensure that users have access to only the resources necessary for their role. This reduces the risk of privilege escalation and data breaches.  Accounting: To maintain accountability, we keep detailed logs of all user actions within the system. These logs track who accessed which resources, when, and what changes were made. Our logging systems are tamper-resistant and regularly audited to ensure compliance and identify potential security incidents early.  Together, these policies form the backbone of our access control strategy, helping Green Pace maintain a secure environment that limits unauthorized access while tracking and responding to security incidents efficiently." |
| **8 / 8A / 8B / 8C** | **Unit Testing Overview**  "In this section, we focus on the purpose and application of unit testing within our security framework. Unit testing ensures that individual components of the code function as expected and helps detect potential vulnerabilities early in the development process. By leveraging frameworks such as JUnit, PyTest, or NUnit, we automate our testing process, focusing on edge cases and expected inputs. The vulnerabilities we target through unit testing include buffer overflows, integer overflows, and input validation errors—areas that are critical for maintaining secure code."  **Test 1: Buffer Overflow**  "For the first test, we address buffer overflow prevention. Buffer overflows occur when data exceeds the allocated memory buffer size, leading to unintended behavior and potential security risks. In this test, we validate that the input is correctly checked to prevent such overflows. A positive test involves input within the allowed limits, which should return valid results. A negative test involves exceeding the buffer limits, which should trigger an error or prevent the overflow. We use JUnit to test boundary conditions and ensure that our code handles buffer allocations safely."  **Test 2: Input Validation**  "The second test focuses on input validation handling. Input validation is crucial to prevent malicious input from being processed by the system. For the positive test, we ensure that correct input formats are accepted, while the negative test involves injecting malicious input that is properly rejected with an error message. By using PyTest for testing, we can simulate different edge cases to ensure robust input validation throughout the system."  **Test 3: Integer Overflow**  "The third test addresses integer overflow detection, a vulnerability that arises when integer values exceed their maximum limit, causing unpredictable behavior. In this test, valid inputs within the expected range should behave as expected, while inputs exceeding the limit should be rejected with appropriate safeguards. We employ NUnit to automate the detection of integer overflow vulnerabilities and ensure the system remains secure under various input scenarios." |
| **9** | "This slide provides an overview of the automation process at Green Pace, highlighting the key stages involved in securing our development lifecycle through automation.  The automation process is divided into Pre-production and Production phases, ensuring that security is integrated from the design phase all the way through to deployment and monitoring.  In the Pre-production phase, we begin with Assess and Plan, where we evaluate the current threat landscape, regulatory changes, and conduct impact analysis to prioritize our security focus areas. This is followed by the Design phase, which incorporates security-driven test design and best practices, such as OWASP guidelines.  Next is the Build phase, where we ensure the use of secure build practices, trusted repositories, and open-source security compliance. Then comes Verify and Test, where vulnerability scanning and functional testing are conducted to ensure the system meets security requirements.  As we transition to the Production phase, the Health Check phase is critical for configuring and deploying security settings, followed by Monitor and Detect, where we actively collect logs, perform SIEM analysis, and detect intrusions.  Respond and Maintain and Stabilize form the final stages. In the event of an attack or issue, we respond by blocking attacks, rolling back services, and then maintaining stability by assessing the security baseline and returning the system to a secure state.  Together, this automation process ensures that security is continuously integrated and monitored throughout the system's lifecycle, enabling rapid responses to emerging threats." |
| **10** | "This slide provides an overview of the tools we use throughout the DevSecOps pipeline at Green Pace to ensure seamless integration of security into every stage of the development process.  The DevSecOps pipeline is designed to integrate security practices into the traditional DevOps workflow, ensuring that security is considered at every step, from development to deployment.  In the design and build phases, we utilize tools like OWASP Dependency-Check and SonarQube for static code analysis, ensuring that vulnerabilities are identified and addressed early in the software development lifecycle.  During the verification and testing phase, we implement tools like Jenkins for continuous integration and JUnit/PyTest for unit testing automation. These tools ensure that any vulnerabilities introduced during development are caught and fixed before deployment.  In production, we use SIEM (Security Information and Event Management) tools such as Splunk to monitor system logs for suspicious activities. Combined with Intrusion Detection Systems (IDS), they allow us to detect and respond to potential threats in real time.  Throughout the process, tools like Docker and Kubernetes help manage secure containerized deployments, ensuring that application environments remain consistent and secure, even as they scale.  By integrating these tools into our DevSecOps pipeline, we maintain a proactive security posture, ensuring that vulnerabilities are detected and addressed at the earliest possible stage, while continuously monitoring for threats." |
| **11** | "In this slide, we assess the risks and benefits of implementing our security strategy immediately versus delaying action.  If we take immediate action, the clear benefit is a stronger security posture. By addressing vulnerabilities upfront, we can reduce potential attack surfaces and ensure better compliance with industry standards. However, the risks of acting now include potential disruptions to existing systems and higher initial costs for implementation.  On the other hand, if we delay action, there are some short-term benefits, such as having more time to refine the strategy and potentially saving on upfront costs. But the risks of waiting are far more significant. Every day we delay, we increase our exposure to potential attacks and miss out on early mitigation opportunities, which could prove costly in the long run.  This strategy is not without its gaps. We need to continuously evaluate our monitoring and automation tools to ensure that they remain effective as our systems evolve.  In terms of next steps, it's crucial that we identify the highest-priority vulnerabilities and address them immediately. This approach allows us to balance the need for rapid action with the practical realities of cost and system stability." |
| **12** | "In this slide, we'll focus on identifying gaps in the existing security policy and suggest improvements to enhance Green Pace's security posture.  Gap 1: Insufficient Incident Response Plan  Currently, our security policy lacks a comprehensive incident response plan. While we have monitoring tools in place, we need a detailed strategy for responding to security breaches, including roles, timelines, and communication plans for addressing incidents as they arise.  Gap 2: Lack of Employee Training  Security awareness training for employees has not been emphasized. This leaves a significant gap, as human error often contributes to security vulnerabilities. Regular training sessions on phishing, password management, and security best practices should be mandated.  Gap 3: Minimal Third-Party Risk Management  Our policy does not currently cover third-party vendors or contractors. Considering many breaches come from third-party connections, we need to ensure that any third-party partner meets the same security standards as Green Pace.  Gap 4: No Regular Penetration Testing  While automated tests are part of the pipeline, there is no emphasis on regular penetration testing. Integrating this into the policy would allow us to identify vulnerabilities that may not be detectable through automated tests alone.  Recommendations  We recommend addressing these gaps by introducing an incident response plan, implementing mandatory employee security training, establishing vendor risk assessments, and scheduling regular penetration tests. By doing so, we strengthen Green Pace’s overall security strategy and mitigate potential threats." |
| **13** | "In conclusion, based on our assessment, there are several key standards that Green Pace should adopt to prevent future security problems and strengthen our overall posture.  Adopt a Comprehensive Incident Response Standard  Formalizing our incident response procedures using standards like NIST 800-61 will ensure that we can quickly and effectively respond to security breaches, minimizing damage and recovery time.  Regular Employee Security Training  Implementing regular training programs for all staff based on security awareness standards like ISO/IEC 27001 can help reduce human errors and increase our ability to identify and respond to potential security threats.  Enhanced Third-Party Risk Management  To better manage risks associated with vendors and contractors, adopting a third-party risk management framework, such as SOC 2 compliance, will ensure that external partners meet our security requirements.  Scheduled Penetration Testing  Adopting regular penetration testing in line with OWASP standards can help identify vulnerabilities that may not be detected by automated scans, further securing our systems against potential exploits.  By implementing these standards, Green Pace can continue to improve its security posture and ensure long-term protection against evolving threats." |
| **14** | “References include Microsoft, Peerbits, and Adobe Stock for images and clipart” |