**Project Two: Security Policy Presentation** **Script**

Duane Wegner

Southern New Hampshire University

CS-405 Secure Coding

Alan Spencer, Adjunct Instructor (M.S)

August 11, 2025

YouTube Link to Video: https://youtu.be/iTDZzXTp0nc

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello everyone, my name is Duane Wegner Welcome to a presentation on Green Pace’s security policy. |
| **2** | Green Pace’s Security Policy sets the standard for secure coding and system practices. It focuses on key areas like input validation, safe memory handling, exception management, secure logging, and encryption. By following these standards, developers create multiple layers of protection. This defense-in-depth approach ensures that if one control fails, others continue to keep the system secure. |
| **3** | The following threat matrix shows the threat level and priority for vulnerabilities covered under the Green Pace security policy. High priority threat should be addressed first however do not ignore low level threats plan for them as soon as higher priority threat coverage is placed into action. |
| **4** | |  |  | | --- | --- | | Validate Input Data | Has been applied to 5 coding standards. | | Heed Compiler Warnings | Has been applied to 1 coding standards. | | Architect and Design for Security Policies | Has been applied to 1 coding standards. | | Keep It Simple | Has been applied to 5 coding standards. | | Default Deny | Has been applied to 5 coding standards. | | Adhere to the Principle of Least Privilege | Has been applied to 1 coding standards. | | Sanitize Data Sent to Other Systems | Has been applied to 2 coding standards. | | Practice Defense in Depth | Has been applied to 5 coding standards. | | Secure and Privacy-Conscious Logging | Has been applied to 6 coding standards. | | Consistent and Correct Error Code Handling | Has been applied to 4 coding standards. | |
| **5** | These coding standards are identified by priority and threat level as seen in the earlier threat matrix. Coding standards were placed into a threat level category based on several factors such as severity, cost, priority, and likelihood. |
| **6** | Encryption is a key part of our security policy because it protects sensitive data at every stage. Encryption at rest secures stored data, like credentials or personal information, using strong methods such as AES-256. Encryption in flight protects data moving across networks with protocols like TLS 1.3, preventing interception or tampering. Encryption in use safeguards data while it is actively being processed in memory, reducing risks from attacks on live data. Together, these policies ensure data stays protected whether stored, transmitted, or in use. |
| **7** | The Triple-A Framework, Authentication, Authorization, and Accounting, ensures secure access and accountability across Green Pace systems. Authentication verifies user identity, Authorization enforces least-privilege access, and Accounting logs all actions for compliance and audits. Together, they create layered security that controls entry, limits permissions, and tracks activity. |
| **8** | Unit testing is important because it helps catch security flaws early in development. By testing small pieces of code, we make sure functions like input validation, encryption, and error handling work as expected. This reduces vulnerabilities before software goes live, supporting our security policy and preventing costly fixes later. |
| **9** | **What it does:** Confirms that the collection object is properly created and starts empty.  **Why:** Prevents null pointer dereference errors, which could crash the program or cause undefined behavior. |
| **10** | **What it does:** Checks that a newly created collection starts empty with size 0.  **Why:** Prevents logic errors that could occur if the collection incorrectly starts with elements. |
| **11** | **What it does:** Intentionally fails the test to confirm that Google Test properly detects and reports failed conditions.  **Why:** Ensures that the testing framework’s error reporting works correctly, and prepares the ground for real negative tests with invalid inputs or out-of-range access. |
| **12** | **What it does:** This test verifies that adding multiple values to a collection works correctly and that the collection’s size matches the number of added elements.  **Why:** It ensures data integrity by preventing lost or mishandled elements, confirming that the collection can correctly store and manage multiple items as expected. |
| **13** | **What it does:** This test checks that accessing an invalid index in the collection triggers an exception, preventing out-of-bounds access.  **Why:** It enforces safe data access, protecting against crashes, undefined behavior, and potential security vulnerabilities from improper memory access. |
| **14** | Automation should be implemented at multiple stages. First, it must be integrated during pre-production verification and testing using static and dynamic analysis tools. Second, automation should run whenever the codebase requires analysis, such as during transitions or health checks. Additional automation can be applied at other stages as needed. |
| **15** | The Green Pace development pipeline uses a continuous DevSecOps cycle to embed security at every stage. External tools like CPPCheck, Clang-Tidy, and SonarQube perform static analysis to catch unsafe code and type issues early. Dynamic analysis and memory checks with Valgrind detect leaks and buffer overflows during testing. Security scanners such as Fortify and Veracode identify vulnerabilities like SQL injection and improper logging. These tools run throughout builds, unit testing, code transitions, and periodic checks to ensure compliance with coding standards and maintain secure software. |
| **16** | The main risks involve code and system vulnerabilities, including buffer overflows, SQL injection, unsafe memory usage, and inconsistent developer practices. Solutions include automated static and dynamic analysis tools, strict coding standards, encryption policies, and the Triple-A framework, all integrated into a continuous DevSecOps pipeline. Acting now reduces breaches, enforces defense-in-depth, and improves code quality, while delaying increases the risk of exposure or system compromise. Future steps should focus on runtime monitoring, automated remediation, ongoing staff training, maintaining unit tests, and expanding security coverage for concurrency, exceptions, and emerging threats. |
| **17** | To maintain an effective security policy, teams must follow and update it as new threats emerge. Current gaps include lack of automated runtime monitoring, limited guidance for emerging threats like zero-day or AI-driven attacks, incomplete implementation of encryption in use, inconsistent developer training, and insufficient automated remediation. Following the principle of “Don’t leave security to the end,” security should be integrated at every development stage through input validation, compiler checks, secure architecture design, least-privilege enforcement, and automated unit testing, ensuring early vulnerability prevention and strong defense-in-depth. |
| **18** | Green Pace has built a strong security policy foundation, but continuous improvement is essential to maintain resilience against evolving threats. Future steps should include expanding automated unit testing to cover new attack vectors, integrating continuous runtime monitoring to detect and respond to threats in real time, and regularly updating policies to reflect emerging technologies such as AI-driven systems. Ongoing developer training will reinforce secure coding practices, while adopting additional encryption standards and access controls will safeguard sensitive data during processing. These measures create a proactive, defense-in-depth strategy that embeds security into every stage of development, ensuring that vulnerabilities are addressed early, system integrity is maintained, and Green Pace remains prepared for future security challenges. |
| **19** | Be sure to look over the following references at your leisure. |