# CS 405 Project Two Script

**Name**: Brayden Cook

**Date**: 8/12/2025

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

**Video Link:** [**https://youtu.be/VB\_8YdrXja0**](https://youtu.be/VB_8YdrXja0)

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Welcome to the Green Pace Secure Development Policy presentation. This presentation outlines the key standards, principles, and practices that guide our software security posture. It also aims to formalize our existing best practices into a consistent and enforceable policy for all developers across Green Pace. |
| **2** | The security policy serves to unify and standardize secure practices across our development teams. By embedding security into our development lifecycle, we ensure consistent risk mitigation and compliance. Furthermore, this policy supports our defense-in-depth approach, aligning secure development with systems architecture to reduce vulnerabilities and improve resilience. |
| **3** | This threat matrix provides a prioritized view of the top vulnerabilities addressed in our coding standards. Each risk is assessed by severity, likelihood, and remediation cost, giving us a calculated priority score. These values guide our mitigation strategy and help focus automated enforcement efforts where they are most needed. |
| **4** | These ten core principles form the foundation of Green Pace’s security philosophy. Each coding standard is directly aligned with one or more principles to ensure a holistic and layered approach to security. Additionally, this mapping ensures our policy isn’t just prescriptive, it’s principled and strategically designed. |
| **5** | These ten coding standards were prioritized by combining risk level, ease of detection, and remediation cost. SQL injection and buffer overflows rank highest due to their frequent exploitation and high impact. Automating detection of these issues is a key part of our secure DevSecOps strategy. |
| **6** | Green Pace mandates strong encryption in all states of data. AES-256 secures data at rest, TLS 1.3 ensures secure transmission, and encryption in use protects data being processed in memory. These layers defend against theft, eavesdropping, and memory scraping. |
| **7** | Authentication, Authorization, and Accounting form the Triple-A Framework. We authenticate users and systems via tokens and multi-factor methods. Access is restricted by role and reviewed frequently. All activity is logged and retained securely for at least 12 months for compliance and accountability. |
| **8** | This unit test ensures that unsigned integer values are used correctly when indexing arrays. It verifies the value is within bounds before access, avoiding logic bugs due to negative indices. Google Test is used here to assert proper type usage. |
| **9** | This test validates that speed inputs do not exceed safe operational limits. We assert that a value greater than the maximum is properly flagged, ensuring developers enforce range validation. It’s critical for preventing logic errors and runtime crashes. |
| **10** | This test uses strncpy and confirms the result is null-terminated. It helps detect risks like buffer overflows and undefined behavior. Google Test makes it easy to isolate unsafe behavior in string manipulation. |
| **11** | This test simulates a typical SQL injection payload being built into a query string. We intentionally let it pass to show how it can be caught and prevented through safe query APIs. It reinforces why input sanitization is essential. |
| **12** | This test allocates and frees heap memory, ensuring that no leaks remain. Tools like Valgrind run alongside unit tests to detect issues not visible to the test framework alone. It confirms secure memory management practices. |
| **13** | This test ensures we’re not using assertions as substitutes for input validation. Assertions should not control application flow in production. Using unit tests enforces the correct use of runtime checks instead. |
| **14** | This test checks that exception-prone code paths are caught and controlled. We simulate a divide-by-zero operation and verify that it doesn’t crash the system. It helps enforce structured exception handling policies. |
| **15** | This unit test verifies that integer overflow occurs and is detectable. While the behavior might be platform-dependent, testing it highlights where overflow could lead to logic failure. It supports secure arithmetic handling. |
| **16** | This test ensures that code safely handles file access failures. It checks that the input file stream does not silently fail. Such handling is crucial for both usability and security. |
| **17** | This test validates the use of RAII for automatic resource cleanup. It sets a flag in the destructor to ensure the resource was cleaned up correctly. RAII prevents leaks and enforces deterministic cleanup. |
| **18** | Automation is the backbone of policy enforcement. From static analysis tools in coding phases to SIEM and IDS in operations, every stage of the pipeline includes security validation. Furthermore, these integrations reduce human error and ensure continuous compliance with our secure development policy. |
| **19** | [See slide 19 of presentation] |
| **20** | Delaying these improvements increases the risk of breaches and untraceable errors. By acting now, we gain confidence in our code quality, regulatory posture, and developer coordination. Overall, these policies make us faster and more secure at the same time. |
| **21** | Our current gaps include runtime detection, outdated secure coding resources, and unmonitored dependencies. These areas increase our exposure and require strategic focus for continued policy maturity. |
| **22** | Looking ahead, we aim to integrate runtime defenses, automate dependency analysis, and modernize our standards. These efforts will enhance our resilience, prevent regressions, and prepare us for future threats. Ultimately, internal training will further embed security into our culture. |
| **23** | As no outside references were used, this concludes our secure development policy presentation. Thank you for your time. |