# CS 405 Project Two Script Template

Waeil Mikhaeil

February 22, 2025

Project Two: Security Policy Presentation

YouTube Link: <https://youtu.be/ZJX9N3uloco>

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Welcome to the Green Pace Security Policy Presentation, authored by Waeil Mikhaeil. This is Career Face transitioning to Green Face, setting the stage for a secure future in software development. |
| **2** | This security policy standardizes the best practices already in use by the Green Pace team. It’s essential as the team grows, ensuring software and data remain protected. The policy supports the Defense in Depth strategy by layering protections—coding standards, encryption, authentication, and testing—to form a robust barrier against threats. This approach ensures compliance and addresses vulnerabilities across the application surface area |
| **3** | Here’s the threat assessment in a matrix format, categorizing vulnerabilities by likelihood and severity. Unlikely low-severity issues include STD-001-CPP and STD-006-CPP—minor but manageable. Unlikely high-severity risks like STD-009-CPP and STD-010-CPP are rare but impactful if ignored. Likely low-severity concerns, such as STD-002-CPP and STD-007-CPP, occur often but are less critical. Likely high-severity threats—STD-003-CPP, STD-004-CPP, STD-005-CPP, and STD-008-CPP—are top priorities due to their frequency and potential damage. Automation, including unit tests and static analysis, detects these early in development. |
| **4** | These 10 security principles guide the policy. Validate Input Data ties to STD-004-CPP for sanitization. Heed Compiler Warnings aligns with STD-002-CPP for return values. Design for Security connects to STD-003-CPP for range checks. Keep It Simple links to STD-008-CPP to avoid namespace changes. Default Deny uses STD-007-CPP for exception handling. Least Privilege pairs with STD-001-CPP for reference qualifiers. Sanitize External Data reinforces STD-004-CPP. Defense in Depth supports STD-005-CPP for memory management. Quality Assurance matches STD-006-CPP for assert behavior. Finally, Secure Coding Standard covers STD-009-CPP and STD-010-CPP for files and constructors |
| **5** | These coding standards are prioritized by impact and frequency. STD-004-CPP, sanitizing data, tops the list to prevent injection attacks. STD-005-CPP, deallocating resources, follows to avoid memory leaks. STD-003-CPP ensures range checks to stop overflows. STD-008-CPP prohibits namespace mods for integrity. STD-002-CPP guarantees return values for reliability. STD-007-CPP mandates exception handling to prevent crashes. STD-001-CPP avoids const/volatile refs for clarity. STD-006-CPP covers assert and abort behavior for debugging. STD-009-CPP closes files to free resources. STD-010-CPP ensures constructor order for proper initialization. This ranking targets attack surface reduction and stability. |
| **6** | Encryption policies secure data in all states. In flight, TLS 1.3 protects data during transmission against interception. At rest, AES-256 encrypts stored data to safeguard it if storage is compromised. In use, secure enclaves like Intel SGX shield data during processing, minimizing exposure. These measures reinforce Defense in Depth across the data lifecycle |
| **7** | The Triple-A framework—Authentication, Authorization, Accounting—ensures secure access. Authentication requires multi-factor authentication with strong, 16-plus-character passwords to verify identities. Authorization uses role-based access control, tied to STD-001-CPP’s least privilege, to limit permissions. Accounting logs all access and changes with timestamps, stored securely, for tracking and auditing. This framework keeps resources locked down and monitored |
| **8** | The Triple-A framework—Authentication, Authorization, Accounting—ensures secure access. Authentication requires multi-factor authentication with strong, 16-plus-character passwords to verify identities. Authorization uses role-based access control, tied to STD-001-CPP’s least privilege, to limit permissions. Accounting logs all access and changes with timestamps, stored securely, for tracking and auditing. This framework keeps resources locked down and monitored |
| **9** | Unit Test 2 - Can Reserve Exceed Max Size?  "This test addresses STD-005-CPP. It verifies if reserving space beyond the collection’s max size throws an std::length\_error. The test confirms the collection is empty, size is zero, and the exception occurs. Result: Pass. This prevents memory over-allocation. Next step: Test incremental reserve increases to ensure gradual limits hold |
| **10** | Unit Test 3 - Does Every Function Return a Value?  "Here’s a test for STD-002-CPP. It checks if computeValue returns a valid result for input 5, expecting it not to equal INT\_MIN, a sentinel for missing returns. Result: Pass. This ensures reliability across functions. Next step: Test multiple exit points to catch all return paths |
| **11** | Unit Test 4 - Are Exceptions Handled?  "This test supports STD-007-CPP. It confirms safeDivide handles division by zero without throwing, returning zero as a default. It expects no throw and validates the result. Result: Pass. This prevents crashes. Next step: Test nested exceptions for deeper coverage." |
| **12** | The DevSecOps pipeline outlines the steps of the development process from beginning to end. Pre-production focuses on designing and testing software to identify issues early. Production involves deploying, monitoring, and maintaining software for smooth operation. In the design phase, OWASP guidelines or the security policy shape a secure foundation. During the build stage, an IDE and compiler handle code compilation and execution. The verify and test phase uses static analysis and unit testing tools to uncover vulnerabilities. In the monitor and detect stage, logging and application monitoring tools track issues in real time. |
| **13** | Should security be addressed now or delayed? The answer is now—without question! Protecting applications and data is a critical priority that demands immediate attention. Security needs to be considered right from the planning stage. Benefits include simpler implementation when embedded early, more robust and reliable code, and improved consistency across the codebase. Risks involve lack of full team buy-in, challenges in retroactively adding security effectively, and potential breaches exposing data if action is postponed |
| **14** | A security policy is useless unless it’s understood and followed. DevSecOps training for developers ensures everyone stays aligned. Automated monitoring of software dependencies detects vulnerabilities and applies patches to prevent supply chain attacks. This resource offers valuable guidance: https://learn.microsoft.com/en-us/nuget/concepts/security-best-practices. |