# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hi, I’m Scott Cain. This presentation outlines the Green Pace Security Policy, built to support our developers with clear, repeatable standards that protect our systems and code. I’ll walk through our threat matrix, coding principles, testing methods, and future recommendations. |
| **2** | As our team grows, it’s important to stay aligned on secure development practices. This policy formalizes what we already do well and adds structure to support defense-in-depth. That means we’re layering protections across every part of our workflow—from code to infrastructure |
| **3** | This threat matrix shows the ten vulnerabilities we’ve identified, ranked by severity and likelihood. Issues like hardcoded credentials and missing input validation are high-risk and need immediate attention. The matrix helps us focus on what matters most and guides our priorities. |
| **4** | To catch vulnerabilities early, we use automated tools like Cppcheck and Visual Studio’s static analysis. These tools scan for buffer overflows, unsafe memory access, and insecure file handling. Automation lets us catch problems before they reach production. |
| **5** | Our policy is built on ten guiding principles, including least privilege, fail securely, and defense in depth. Each principle connects directly to specific coding standards, making it easier for developers to apply them consistently across projects. |
| **6** | Here are our ten coding standards, listed in priority order. We start with hardcoded credentials and input validation because they pose the greatest risk. From there, we address buffer overflows, secure file handling, and error management. Prioritization is based on exploitability and impact |
| **7** | Our encryption strategy covers data in flight, at rest, and in use. We use TLS 1.3 for secure transmission, AES-256 for stored data, and secure memory handling to protect data during processing. These layers ensure confidentiality and integrity throughout the system |
| **8** | The Triple-A framework—Authentication, Authorization, and Accounting—helps us manage access securely. We use multi-factor authentication, role-based access control, and detailed audit logs to track user activity and support accountability. |
| **9** | This unit test checks for buffer overflow vulnerabilities. We validate input boundaries and use safe memory functions. Visual Studio’s unit testing framework lets us simulate edge cases and confirm that our code handles them securely. |
| **10** | Here’s a unit test for input validation. We test against SQL injection and cross-site scripting by feeding malicious input and verifying that our sanitization logic blocks it. These tests are part of our CI pipeline and run automatically with each commit. |
| **11** | This DevSecOps diagram shows where security tools fit into our automation flow. Static analysis runs during coding, unit tests run during integration, and dynamic scanning happens before deployment. The compiler flags unsafe code early, and automated tests catch regressions. |
| **12** | Let’s look at risks and benefits. Fixing hardcoded credentials and input validation now reduces the chance of a breach. Waiting increases exposure and technical debt. Lower-risk items like logging format can be deferred, but only with a clear plan in place. |
| **13** | Looking ahead, we need centralized secrets management and consistent logging practices. We recommend adopting OWASP Secure Coding Guidelines and integrating SAST and DAST tools. Real-world breaches like Equifax show how costly delays can be. |
| **14** | Thanks for reviewing the Green Pace Security Policy. This presentation outlines our commitment to secure coding, layered defense, and continuous improvement. By following these standards, we protect our users, our systems, and our future. |