# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is Nathanial Lowe, and this presentation covers the Green Pace Security Policy. This project demonstrates how we translate secure coding principles and standards into consistent, enforceable practices that strengthen system resilience across our development lifecycle. |
| **2** | This presentation outlines the Green Pace security policy and how it supports a defense-in-depth approach. By converting daily best practices into standardized policy, we ensure that our developers operate with consistent awareness of security vulnerabilities and mitigation strategies. The policy aligns with secure software architecture principles to reduce our attack surface and assume potential vulnerabilities throughout the system. |
| **3** | This threat matrix summarizes the security risks identified through our 10 secure coding standards. High-priority threats like SQL injection and memory misuse are both likely and severe, warranting immediate attention. Medium-priority items include pointer misuse, unchecked input, and improper exception handling. Lower-priority but still significant risks include assertion misuse and integer overflow. This matrix helps prioritize remediation and allocate testing resources. |
| **4** | Our secure coding policy is grounded in 10 key principles, such as validating input data, keeping code simple, practicing defense in depth, and using effective QA techniques. Each of our coding standards aligns with one or more of these principles to ensure a holistic, principle-driven approach to writing secure software. |
| **5** | These are the 10 coding standards in priority order, based on threat level, likelihood, and remediation cost. Top priorities include avoiding SQL injection, securing file access, and preventing use-after-free and memory corruption. The prioritization ensures we address the most critical risks first while still maintaining a comprehensive posture. |
| **6** | Our encryption policy covers three domains: in flight, at rest, and in use. Encryption in flight protects data transmitted over networks using protocols like TLS 1.3. Encryption at rest secures data stored on disk using AES-256 or similar standards. Encryption in use protects sensitive data while it's actively being processed, often by isolating processes or using encrypted memory. These policies ensure confidentiality, integrity, and compliance across the system lifecycle. |
| **7** | The Triple-A framework—Authentication, Authorization, and Accounting—is vital to secure user and system interactions. Authentication confirms a user's identity using strong credentials. Authorization defines what authenticated users can access or modify, enforcing least privilege principles. Accounting logs user activity, including logins, data access, and permission changes. Together, these controls support secure auditability and reduce the risk of insider threats or accidental misuse. |
| **8** | This unit test focuses on integer overflow, which occurs when values exceed the maximum representable range. The test checks whether inputs like INT\_MAX + 1 are correctly flagged and whether valid inputs compute correctly. By covering both expected and edge case scenarios, these tests help prevent logic errors and potential vulnerabilities in financial, buffer, or array-bound calculations. |
| **9** | Automation is critical to enforcing the policy. Within the DevSecOps pipeline, tools like Cppcheck, SonarQube, Clang-Tidy, and Visual Studio Code Analysis are integrated during code commit, build, and test stages. These tools detect violations of our coding standards early, helping ensure that secure code is the default, not an afterthought |
| **10** | In the DevSecOps pipeline, security tools are integrated throughout the development lifecycle. During the coding and build stages, static analysis tools like Cppcheck, Clang-Tidy, and SonarQube scan the codebase for compliance with our secure coding standards. In the testing stage, Visual Studio’s unit test framework validates function behavior, while GitHub Actions or Jenkins can automate these tests on every code push. Integrating tools early and often allows us to catch vulnerabilities before they reach production and ensures that security is treated as a shared responsibility |
| **11** | Addressing vulnerabilities proactively offers significant benefits: reduced breach risk, improved code reliability, and better alignment with regulatory compliance. If we act now, we can eliminate high-risk flaws like SQL injection and memory corruption before they’re exploited. Many vulnerabilities such as unchecked input, memory mismanagement, and improper resource handling exist in legacy or hastily written code. Inconsistent adherence to secure coding standards across developers increases the attack surface. Lack of automation in security enforcement delays detection of issues until late in the SDLC. In order to solve these issues, we can start by implementing strict coding standards using SEI CERT C++ guidelines. Integrate static analysis tools (e.g., SonarQube, Cppcheck, Clang-Tidy) into the DevSecOps pipeline and train developers on secure design principles and enforce code reviews and unit testing. |
| **12** | The risks if you act now are not drastic but still should be noted: Developers need time to adopt and apply secure practices. CI/CD workflows must be updated to include automation and policy enforcement. Acting early has significant benefits, like lowering remediation costs and strengthening software integrity. Reduce the likelihood of a breach, protecting user data and the company reputation. Improves maintainability and team alignment around common security practices. |
| **13** | The risk if you wait is that vulnerabilities may go undetected and exploited in production. Technical debt will increase, making future fixes more costly and complex. Regulatory or customer trust issues may arise after a security incident. While waiting, strategy gaps show themselves, like not all developers currently use static analysis or test coverage tools. Security practices are not consistently enforced across all project stages. What is recommended to solve these issues is to immediately enforce secure coding standards. Automate policy compliance with DevSecOps tools at every stage: build, test, deploy. Conduct regular code audits and update policies as new threats emerge. |
| **14** | While our current security policy addresses coding vulnerabilities effectively, there are still several gaps. For instance, we lack continuous runtime monitoring and behavioral analysis in production. Additionally, third-party dependencies and open-source packages have not been fully audited for risks. To enhance coverage, we should adopt a zero-trust policy for external libraries, apply Software Composition Analysis tools, and set up logging alerts for privilege escalation or unexpected file access. Another gap is the limited developer training on security-focused code reviews. Establishing training and checklist-driven peer reviews would help enforce standards across the team. |
| **15** | Moving forward, we must adopt additional standards that anticipate future threats. Implementing secure memory management through smart pointers, integrating formal threat modeling, and mandating SEI CERT C++ compliance are crucial. By applying the principle of least privilege, validating all input, and automating security tests, we move toward a mature and proactive security posture. Our policies must evolve alongside new threats, making continuous improvement and automation central to our strategy. |
| **16** | The references used throughout this presentation include official SEI CERT secure coding standards, documentation from SonarSource, Cppcheck, and Clang-Tidy, as well as supporting articles from trusted sources like the OWASP Foundation and Microsoft DevSecOps guidance. These sources validate our approach and help ensure alignment with industry-recognized best practices for secure software development. |