# 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 **Selva**, and today I’ll be presenting the **Green Pace Security Policy and Implementation Guide**, developed to support secure, standardized, and scalable software development practices across our growing team. |
| **2** | As Green Pace expands rapidly, so do the security challenges. Without a well-defined security policy, inconsistencies in coding practices can lead to vulnerabilities, breaches, or compliance failures. Our objective is to introduce a policy that embeds security from the ground up—starting in design and flowing through development, testing, deployment, and monitoring. |
| **3** | We assessed our system and categorized potential threats using a matrix of severity and likelihood.   * **Priority threats** like inadequate encryption and lack of authentication controls are both likely and highly severe. * **Likely threats** such as unhandled exceptions and insecure memory access are common but still manageable with safeguards. * **Low priority issues** include items like missing automation tools. * **Unlikely but severe risks**, like infinite recursion, are harder to predict but must still be acknowledged. Each threat is mapped to coding standards and mitigation practices to reduce exposure. |
| **4** | Our policy is built on ten core security principles, including input validation, secure defaults, auditability, and least privilege. For instance, we validate all inputs to guard against injection attacks and enforce role-based access to maintain secure separation of duties. These principles not only improve security—they improve quality, too. |
| **5** | We’ve defined and prioritized ten coding standards based on exploitability and risk history. For example, **bounds checking** and **secure memory handling** are top priorities, helping us prevent issues like buffer overflows and memory corruption. We enforce these standards through peer review, training, and automated analysis tools. |
| **6** | Encryption is enforced across three fronts:   * **In transit**, all communication uses TLS 1.3 with HSTS and Perfect Forward Secrecy. * **At rest**, we secure data with AES-256 encryption and tightly controlled access policies. * **In use**, memory-level protection and field-level encryption reduce exposure even while data is being processed. This layered approach ensures data is protected at all times. |
| **7** | Authentication, Authorization, and Accounting form our AAA framework.   * **Authentication** includes SSO and MFA. * **Authorization** follows least-privilege principles using RBAC. * **Accounting** uses centralized logging, audit trails, and SIEM tools. This framework ensures only the right people access the right data—and that we know exactly when and how they do. |
| **8** | We’ve written unit tests to detect critical vulnerabilities:   * **Test 1:** Buffer overflow boundaries * **Test 2:** Null pointer dereferencing * **Test 3:** Noexcept safety * **Test 4:** Memory leaks using Valgrind or AddressSanitizer   These tests are run in every CI/CD cycle to ensure no regressions or new vulnerabilities. |
| **9** | Security is fully integrated into our DevSecOps lifecycle.   * We **plan** using threat modeling. * We **code** with static analysis tools like Clang-Tidy. * We **build and test** using GitHub Actions and SonarQube. * We **deploy and operate** using Infrastructure as Code and continuous monitoring. Automation ensures nothing slips through the cracks. |
| **10** | [Insert text.] |
| **11** | Without enforcement, coding standards become suggestions—not safeguards.   * The risk of delaying security implementation is high: breaches, legal fines, and customer loss. * Acting now means early detection, better alignment across teams, and reduced future cost. Our policies turn security into a proactive discipline—not a reactive fix. |
| **12** | Based on our assessment and current implementation:   * We **recommend enforcing security checks** as part of mandatory code reviews and automated CI pipelines. * **Legacy code** should be refactored to meet current standards and retested with updated tooling. * **Mandatory training** on secure C++ development practices should be provided to all engineers. * We should **expand unit test coverage** and make threat modeling a regular activity during planning. * Finally, **investing in secure-by-default frameworks** will minimize human error and reduce recurring issues. These steps will close gaps and future-proof our secure development process. |
| **13** | In conclusion, implementing and automating a security policy isn’t just a checkbox—it’s a strategic necessity. With well-defined coding standards, consistent testing, and DevSecOps automation, we proactively prevent security incidents. A real-world case is the **Equifax breach**, where a known vulnerability in Apache Struts led to the exposure of over 140 million records. Had automated static analysis and patching been enforced, this could have been avoided.  Green Pace’s strategy is to never be caught off-guard. Security is not a phase—it’s a mindset embedded in every commit. |
| **14** | We’ve built our policies on trusted, industry-recognized standards and tools:   * OWASP Top 10 * SEI CERT C++ Secure Coding Guidelines * CppCheck and Clang-Tidy Documentation * Google C++ Style Guide * NIST Cybersecurity Framework * CVE-2017-5638 (Apache Struts Equifax breach) These references ensure that Green Pace’s security strategy aligns with the latest best practices. |