# CS 405 Project Two Script

Jose Munoz

8/24/25

<https://youtu.be/2A9IhdJkG4U>

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Welcome to my presentation on Green Pace’s Defense-in-Depth Security Policy. I’ll walk you through our core security principles, coding standards, and strategies for encryption, authentication, automation, and risk management. |
| **2** | Our policy applies layered security across all stages of development and operations. It was created to reduce vulnerabilities like SQL injection, buffer overflows, and weak authentication, while supporting compliance and secure software delivery. |
| **3** | Key risks include unchecked input, buffer overflows, SQL injection, and hardcoded keys. Automated tools like Cppcheck and SonarQube detect these issues early in our CI/CD pipeline, reducing both frequency and severity. |
| **4** | Our ten security principles include input validation, least privilege, defense in depth, secure coding, and more. Each principle aligns to specific coding standards to ensure consistent implementation across the system. |
| **5** | Our coding standards are prioritized by risk. SQL injection prevention and removing hardcoded credentials rank highest, followed by input validation, safe buffer usage, thread safety, memory management, exception handling, assertions, RAII, and compiler warnings |
| **6** | Encryption works in three main stages: when data is moving, stored, or actively being used.  encryption at rest means we keep stored data, like files or databases, encrypted so that if someone steals the drive or gets unauthorized access, they still can’t read it without the proper keys.  encryption in flight means we protect data while it’s traveling between systems or over the internet using secure protocols. This helps stop anyone from intercepting it along the way.  Finally, encryption in use is about protecting data while it’s actively being processed in memory, using special software or hardware techniques to make sure it can’t be exposed even while the system is working with it. |
| **7** | The Triple-A framework covers authentication, authorization, and accounting.  Authentication is about verifying who you are. Our policies here include things like requiring strong passwords, multi-factor authentication, and secure login methods to make sure users are who they say they are.  Authorization decides what you’re allowed to do after you’re logged in. We use role-based access controls and least-privilege policies so people only have access to what they need.  Finally, accounting tracks user actions. This means logging activity, monitoring access, and reviewing reports so we can spot suspicious behavior and maintain accountability. |
| **8** | We use Google Test to automate unit testing for vulnerabilities. Tests include positive and negative cases for SQL injection, input validation, and memory safety. We also integrate sanitizers and fuzzing to catch hidden bugs. |
| **9** | Assess and Plan (Pre-production): Security automation begins with threat modeling tools and compliance checkers. These tools continuously monitor regulatory changes and backlog priorities.  Design: We apply OWASP and security test-driven design using automated code linters and static analysis (SAST) during development.  Build: The compiler runs here with strict flags and integrated static analyzers like Cppcheck and Clang-Tidy. Dependency scans and open-source license checkers are automated to detect vulnerabilities early.  Verify and Test: Automated unit tests, vulnerability scans, and dynamic application security tests. Security test frameworks validate functional and compliance requirements automatically.  Transition and Health Check: Security automation includes configuration scanners and penetration testing tools before deployment.  Monitor and Detect (Production): Log monitoring, and intrusion detection systems automatically detect anomalies and trigger alerts.  Respond: Automated workflows block malicious traffic, roll back unsafe changes, and disable compromised services.  Maintain and Stabilize: Continuous scanning against security baselines and automated regression tests ensure we remain in a secure state after updates or incidents. |
| **10** | Parasoft C/C++test runs unit tests and static checks on our C/C++ code, enforcing CERT/MISRA in CI.  SAST scans source in IDE/PRs to catch insecure patterns early.  DAST probes the running app in staging to find runtime weaknesses.  SIEM aggregates logs in prod for detection and response.  code is hardened early, behavior is validated pre-release, and production is continuously monitored. |
| **11** | Encryption is inconsistent, some secrets are still in code or documents, releases have few automated checks, and some libraries and logging are outdated.  We plan to standardize encryption, move secrets into a secure vault, add automated build checks, sign releases, and improve logging and alerts.  If we act now, we reduce breach risk, ship faster and safer, make audits easier, and detect and recover from issues more quickly.  The trade-offs are some tool and setup costs, a brief slowdown while we tune things, and some training and change management.  If we wait, there’s no disruption or spend today, but the risks include higher chances of an incident, bigger cleanup costs, possible fines, and growing security debt that will be more expensive to fix later. |
| **12** | Theres always a chance for gaps in security. There could be little encryption while data runs, training isn’t consistent, and having full zero-trust controls.  To fix that, we should add runtime encryption with clear key rules, give developers regular security training, and move toward zero-trust with stronger access checks.  This Equifax breach shows how missing these basics can lead to big problems, so tackling them now will make us much safer. |
| **13** | Looking ahead, we should adopt well-known security standards and Benchmarks to keep our defenses strong.  We also need to prepare for future issues like clouds or even potential misuse of AI.  Staying secure means constant improvement of regular audits, threat modeling, automated scans, and red team testing.  Security isn’t a one-time project. It’s ongoing, and we need defense-in-depth and automation that can adapt as new threats appear. |
| **14** | References |