**Project Two: Security Policy Presentation – Narration Script**

**Name: Marissa Lanza  
Date: August 14, 2025  
Course: CS 405 – Secure Coding (CS-405-15275-M01)  
Assignment: Project Two – Security Policy Presentation  
Organization: Green Pace  
Submission Method: MP4 submitted directly via Brightspace (**[**https://youtu.be/xokUEyZ6mEo**](https://youtu.be/xokUEyZ6mEo)**)**

**Slide 1 — Title**

Hi everyone, I’m Marissa Lanza. This presentation introduces the Green Pace Security Policy Guide—our playbook for building resilient code and systems architecture across teams as we scale.

**Slide 2 — Overview: Defense in Depth**

We’re standardizing secure-by-default practices so what we already do well becomes consistent and auditable. We apply defense-in-depth and an assume-breach mindset: shrink attack surface, verify every request, and contain failures so one defect doesn’t become a crisis.

**Slide 3 — Automation to Catch Vulnerabilities**

Security must be continuous. Our pipeline integrates pre-commit checks (secrets, style), compiler hardening, SAST, SCA/SBOM, and unit + fuzz tests. In staging, DAST validates live endpoints; releases are scanned and signed; production has centralized logging and anomaly alerts. You’ll see unit-test results and code examples shortly.

**Slide 4 — Threats Matrix (by Coding Standard)**

This matrix ranks risks by likelihood × impact. Top items we tackle first are STD-001 Injection/Input Validation, STD-008 Secrets Management, and STD-009 Supply Chain. We also manage auth/crypto weaknesses and memory safety defects. This risk-driven order ensures the biggest blast-radius issues are addressed first.

**Slide 5 — Ten Principles → Ten Standards**

Our ten principles map directly to ten coding standards, so design intent is enforced in code and in the pipeline. For example, Least Privilege informs authentication and deployment hardening; Validate All Inputs maps to STD-001; Log and Monitor Securely maps to STD-004/005. The mapping keeps policy actionable.

**Slide 6 — Coding Standards (Priority & Rationale)**

Priority reflects exploitability, blast radius, detectability, and compliance pressure:

STD-001 Input Validation/Output Encoding;

STD-002 Auth & Session;

STD-003 Cryptography;

STD-004 Error Handling;

STD-005 Logging/Monitoring;

STD-006 Memory/Resource Safety;

STD-007 Concurrency/TOCTOU;

STD-008 Secrets/Configuration;

STD-009 Supply Chain Security;

STD-010 Secure Build/Deploy with CI/CD gates.

**Slide 7 — Encryption Strategy**

In transit: enforce TLS 1.3 with strong cipher suites and HSTS.  
At rest: AES-GCM via vetted libraries; keys managed and rotated in KMS/HSM.  
In use: minimize plaintext in memory, zeroize sensitive buffers, and use enclaves/TEEs where appropriate. No home-grown crypto; approved libraries only.

**Slide 8 — Triple-A Framework**

Authentication: MFA; phishing-resistant methods where possible; passwords hashed with Argon2id.  
Authorization: least-privilege RBAC/ABAC, scoped short-lived tokens, deny-by-default.  
Accounting: centralized, immutable logs with redaction, retention, and alerts for sensitive events (role changes, token misuse, policy edits).

**Slide 9 — Unit Testing Framework (Visual Studio + GoogleTest)**

We created a dedicated Google Test (gtest) C++ project, referenced the app project, and ran tests via Test Explorer. Tests run in milliseconds, making them ideal CI gates. Failing tests block merges and releases, proving the policy is repeatable and enforceable.

**Slide 10 — Unit Test UT-002: Path Traversal**

This test validates safe file-path handling. We reject .., URL-encoded traversal, and absolute paths; we allow only safe relative paths. The Test Explorer screenshot shows the suite passing. This directly enforces STD-001 (Input Validation) and reduces file disclosure risk.

**Slide 11 — Unit Test UT-003: Bounds / Length Checks**

These tests guard against out-of-bounds access. A valid index returns the expected value; an invalid index is flagged as unsafe. Passing results in Test Explorer demonstrate memory-safety controls aligned to STD-006 and prevent crashes/UB before release.

**Slide 12 — Automation Summary (DevSecOps Flow)**

Pre-commit: secrets scan, style/format, quick unit tests.  
Build: MSVC hardening flags; SAST.  
Test: gtest unit tests, coverage, and targeted fuzzing.  
Stage/DAST: scan running endpoints in QA.  
Release: signed artifacts, SBOM + provenance.  
Runtime: centralized logs/SIEM with alerts.  
Gates: critical findings fail the pipeline—evidence is published with each build.

**Slide 13 — Risks and Benefits**

Act now: reduces breach likelihood and impact, improves audit readiness, and lowers long-term cost.  
Wait: increases security debt and incident severity. We phase rollouts, start with “fail on High/Critical,” and tune rules to manage noise while raising our baseline.

**Slide 14 — Recommendations & Roadmap**

Address STD-001, STD-008, and STD-009 first. Codify AAA policies as CI gates. Add fuzzing for parsers and auth flows; expand SBOM and signed builds; consider reproducible builds. Strengthen training, threat modeling, and tabletop exercises to keep people, process, and tech in sync.

**Slide 15 — Conclusion**

Combining standards with automation makes secure-by-default the path of least resistance. A threat-driven approach ensures we fix the most dangerous issues first and keep Green Pace audit-ready as we grow.

**Slide 16 — References (APA)**

NIST. (2020). SP 800-53 Rev. 5: Security and Privacy Controls for Information Systems and Organizations.

NIST. (2023). SP 800-63B: Digital Identity Guidelines – Authentication and Lifecycle Management.

OWASP Foundation. (2021). OWASP Top Ten Web Application Security Risks.

OWASP Foundation. (2021). Application Security Verification Standard (ASVS) 4.x.

SEI CERT. (2016). SEI CERT C Coding Standard.

MITRE. (n.d.). Common Weakness Enumeration (CWE).

Microsoft. (n.d.). Security Development Lifecycle (SDL).

FIRST. (2023). Common Vulnerability Scoring System (CVSS) v4.0 Specification.

Google. (n.d.). GoogleTest — Google C++ Testing Framework.

OWASP ZAP. (n.d.). Zed Attack Proxy (ZAP) Documentation.