# Maridelle Anne Gonzales

June 20, 2025

Project Two: Security Policy Presentation

<https://youtu.be/-5z5hluc3BM>

# CS 405 Project Two Script

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, everyone. My name is Maridelle Anne Gonzales, and I’m excited to present the Green Pace Security Policy. Today, I’ll walk you through our approach to creating a robust, standardized, and enforceable set of security policies that guide the team as we design, build, and maintain secure software. This presentation captures the core ideas behind Green Pace’s policies, including coding standards, encryption strategies, and access controls, to foster a culture of security within our team. |
| **2** | Green Pace operates in an environment where security is vital for trust, continuity, and compliance. Our security policy embraces a ‘defense in depth’ approach, which means we don’t rely on a single measure to protect our code and data. Instead, we apply multiple layers of protection from coding standards and code reviews, to automated scans, to access controls and encryption. This approach significantly reduces the risk of a successful attack. The goal is to make sure that even if one layer fails, other layers remain in place to protect the integrity, confidentiality, and availability of our systems. This approach allows our developers to build robust, secure code that benefits both the company and its customers. |
| **3** | Here you can see our Threats Matrix, which maps coding and design vulnerabilities by their likelihood and impact. We’ve identified critical threats like SQL Injection and Improper Input Validation as high-priority issues because of their frequency and potential for catastrophic outcomes. Meanwhile, low-priority concerns, such as the use of ‘goto’ statements, have minimal risk but still deserve attention for maintainability. By prioritizing these threats, we focus our efforts and resources where they matter most, addressing the biggest risks first, ensuring our codebase is both secure and resilient. This matrix also allows us to justify our remediation timeline and stay aligned with best practices for software security. |
| **4** | The ten core security principles form the foundation for every coding standard we implement. These ten principles range from input validation and using the ‘least privilege’ design approach, to practicing ‘defense in depth’ and ‘keeping it simple.’ Together, these principles guide how we design, review, and maintain code across the team. They also serve as a lens for identifying risk and making sound engineering decisions. By aligning every coding standard with one or more of these ten principles, we create a roadmap for developers that is actionable, concrete, and rooted in established best practices. In doing so, we reduce security risk from the very first line of code. |
| **5** | Here we have ten coding standards prioritized from highest to lowest risk. At the top are issues that can result in serious breaches, like data and SQL Injection vulnerabilities. These are rated as critical due to their potential for significant data loss and service downtime. Mid-priority standards like assertions, exceptions, and resource management help us write robust, maintainable, and error-free code. At the lower end are stylistic and maintainability concerns, like avoiding ‘goto’ statements. Together, this prioritization provides developers with a roadmap for focusing their efforts. The approach ensures that the highest-impact issues are addressed first, allowing us to balance security, reliability, and maintainability across the codebase. |
| **6** | Green Pace’s encryption policies are tailored to cover data across its complete lifecycle. At rest, sensitive data stored in files, databases, or backups must be protected with AES‑256 encryption, making it unreadable to unauthorized parties. In flight, we use TLS to secure data traveling across networks, ensuring that user information is protected from interception. In use, data within application memory is kept secure by leveraging trusted libraries that encrypt sensitive information while it’s being processed. Together, these policies create a strong encryption posture, making sure that every piece of sensitive data is protected from the moment it’s captured, until it’s deleted from the system. |
| **7** | The Triple‑A Framework, Authentication, Authorization, and Accounting, is central to controlling access and accountability within the Green Pace system. Authentication verifies user identities using strong methods such as multi‑factor authentication. Authorization applies role‑based access controls, making sure that only authorized personnel can access sensitive data or make critical changes. Meanwhile, Accounting maintains a record of all user activity and significant events within the system, providing an auditable trail for security review. Together, these three ‘A’s create a comprehensive approach that supports accountability and minimizes risk across the organization. |
| **8** | Unit testing is a cornerstone of our approach to writing secure, resilient code. For example, I chose to focus on buffer overflow vulnerabilities, one of the most common in C++, to demonstrate the power of automated unit testing. By implementing the Google Test framework, I created test cases that cover both expected and edge case inputs. These test scenarios include inputs that would normally cause an overflow, verifying that our input constraints and error handling work as intended. In doing so, unit testing gives us a measure of confidence that our safeguards are functioning correctly, making our software more robust and trustworthy. |
| **9** | In this example, we tested and resolved a buffer overflow vulnerability caused by accepting too many user input characters. We replaced the unsafe method with one that limits input size and added checks for overflows. These precautions help maintain data integrity and protect sensitive information. |
| **10** | Through thorough unit testing using the Google Test framework, we identified and resolved boundary condition and exception handling errors. Negative test cases confirmed that invalid operations trigger the correct responses. The results demonstrated that the code behaves reliably in both normal and edge-case scenarios. |
| **11** | In this slide, I’m showing how we use automation across the whole software lifecycle. In the early stages, like planning, assessing, and designing, we use tools like SonarQube, Cppcheck, and Clang‑Tidy to spot coding issues right when we write the code. During the build and test phases, tools like Coverity and Fortify help us find security vulnerabilities before the code goes live. After deployment, when the system is in production, we use monitoring tools like the ELK Stack and Splunk to watch for any strange behavior and alert us quickly. Basically, this approach makes security a part of every step of development, making it much harder for threats to slip through. |
| **12** | The risk of inaction is significant. Without addressing critical issues like SQL Injection or Buffer Overflows, we expose ourselves to data breaches, downtime, and loss of trust. The longer we wait, the higher the remediation cost becomes and the greater the risk of damage to the brand. The benefits of acting now are profound, improved compliance, stronger customer trust, and a more resilient infrastructure. Our approach allows us to fix critical vulnerabilities first, applying policies and tooling that have long‑term benefits for maintainability, security, and scalability. In short, acting early is an investment in quality, security, and the ongoing reliability of our code and services. |
| **13** | Looking to the future, our focus must be on expanding these policies to cover mobile platforms and expanding AI‑driven threat detection. This includes making sure that every piece of code that enters our repositories is checked against our policies, extending these policies to mobile app deployments, and leveraging AI to spot threats before they can be exploited. Real‑world incidents, like the Capital One breach caused by a misconfiguration, highlight why vigilance and early enforcement matter. By doubling down on training, tooling, and automated enforcement, we can fill the gaps in our current policies and build a stronger, more secure future for Green Pace and its stakeholders. |
| **14** | In conclusion, this security policy is more than a set of rules, it’s a roadmap for making security an integral part of how we design, build, and maintain software. By aligning coding standards, encryption policies, and access controls with the ten core principles, and by embedding automated detection within our DevSecOps pipelines, we can reduce risk and build trust with every line of code. As we evolve, focusing on mobile security and AI‑enhanced threat detection will help us stay ahead of the curve. Together, this approach ensures that Green Pace delivers safe, secure, and resilient products for ourselves, our stakeholders, and our customers long into the future. Thank you. |
| **15** | These are the references that shaped the policies and best practices presented today. By relying on proven standards and leveraging trusted external tools, we ensure that our approach is grounded in the best available knowledge. |