**Assignment:** Project Two: Security Policy Presentation

**Name:** Jon Wickerd  
**Date:** February 19, 2025

**Video presentation:** https://www.youtube.com/watch?v=YUwatX2mX0Q

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is Jon Wickerd, and today I will be presenting the security policy for Green Pace. This policy outlines the guiding principles, standards, and best practices for developers to secure our software and systems. It’s an essential step in ensuring our defense-in-depth strategy is robust and effective in mitigating potential threats. |
| **2** | Welcome to the Green Pace Security Policy presentation. Our goal is to protect Green Pace’s infrastructure using a layered security approach. This policy establishes core security principles, coding standards for C and C++, and essential guidelines for authorization, authentication, auditing, and data encryption. These standards are designed to support a defense-in-depth strategy, ensuring that our development process leads to secure, resilient, and non-vulnerable solutions. By following these guidelines, we strengthen our security posture and reduce risks across our systems. |
| **3** | This threats matrix categorizes vulnerabilities based on their likelihood and priority. High-priority threats require immediate attention due to their potential high impact on security. These vulnerabilities pose significant risks and could lead to severe breaches if left unaddressed.  Low-priority threats, while still important, have a lower impact and can be addressed later as part of ongoing security improvements. By organizing threats in this way, we can efficiently allocate resources and focus on mitigating the most critical risks first, ensuring a proactive approach to securing our systems. |
| **4** | These ten guiding security principles form the foundation of our secure development practices.  Validate Input Data – Prevents injection attacks by ensuring all inputs are properly checked.  Heed Compiler Warnings – Helps detect potential vulnerabilities early in the development process.  Architect and Design for Security Policies – Security should be integrated from the start, not as an afterthought.  Keep It Simple – Complexity increases risk; simple designs are easier to secure.  Default Deny – Access should be restricted by default and only granted when explicitly needed.  Adhere to the Principle of Least Privilege – Limits user and process permissions to reduce attack surfaces.  Sanitize Data Sent to Other Systems – Ensures data integrity and prevents malicious payloads from spreading.  Practice Defense in Depth – Multiple layers of security provide stronger protection.  Use Effective Quality Assurance Techniques – Regular testing and reviews help detect vulnerabilities early.  Adopt a Secure Coding Standard – Consistency in secure coding minimizes errors and security flaws.  By following these principles, we reinforce a security-first mindset in our development process. |
| **5** | Our coding standards ensure secure development by categorizing rules like STD-001-CPP based on severity, likelihood, remediation cost, priority, and enforcement level.  Severity & Likelihood: High-risk issues need immediate attention.  Remediation Cost: Assesses effort needed for fixes.  Priority: Ensures critical issues are addressed first.  Enforcement Level: Defines mandatory vs. recommended practices.  Following these standards helps prevent vulnerabilities and maintain security consistency. |
| **6** | Our encryption strategy is vital for safeguarding data in various stages of its lifecycle. We will ensure that data is encrypted:  In Transit – Using SSL/TLS to secure communication channels between clients and servers.  At Rest – Encrypt sensitive data stored in databases or file systems using AES or similar encryption algorithms.  In Use – Protect data during processing through secure memory management and techniques like homomorphic encryption, when applicable. |
| **7** | The Triple-A framework — Authentication, Authorization, and Accounting — is fundamental to our security strategy.  Authentication – Verifying the identity of users or systems to ensure access is granted only to authorized entities.  Authorization – Determining the level of access users or systems have based on their roles or attributes.  Accounting – Tracking user activities to ensure accountability and detect suspicious behavior. |
| **8** | Unit testing is essential for identifying vulnerabilities early in development. These tests validate the security and reliability of our code by checking for common issues like buffer overflows, input validation errors, and improper memory handling.  By using a structured unit testing framework in C++ with Visual Studio, we can systematically detect and fix security flaws before deployment. This approach ensures our code adheres to best practices and remains resilient against potential attacks. |
| **9** | This diagram outlines the DevSecOps lifecycle, focusing on integrating security at every stage of development and deployment.  Pre-Production emphasizes secure development and testing. This includes assessing threats, designing with security in mind, using secure builds, and conducting thorough verification and testing.  Production ensures secure deployment, continuous monitoring, and fast incident response. This phase includes configuring systems securely, monitoring for threats, and responding to security incidents quickly.  At the heart of the lifecycle, we integrate Development (DEV), Security (SEC), and Operations (OPS) to ensure security is a constant focus throughout both development and operational phases. |
| **10** | DevSecOps integrates security throughout the software development lifecycle. We utilize tools such as CPPCheck for static code analysis, Clang as a front-end compiler, and Parasoft C/C++test for verification and testing to ensure robust security. |
| **11** | Benefits:  Implementing security early creates a proactive defense strategy, helping to prevent security breaches before they happen. It establishes structure and consistency across the development process, reducing overhead during testing and providing a stronger defense against attackers.  Risks:  There are some trade-offs to consider, such as financial and time costs associated with setting up security measures. Implementing security can also add complexity to managing the system and may require additional resources. Additionally, while security measures reduce risks, breaches could still harm customer trust and data integrity if they occur. |
| **12** | To improve our security, we need to focus on a few key areas:  First, address gaps in threat detection and response time. We also need to expand coverage for new attack methods.  It’s essential to integrate more code analysis tools and update security frameworks regularly. We should also invest in secure development training for our team.  Automating testing and CI pipelines will ensure ongoing security checks, and we must strengthen encryption in all phases.  Incorporating Threat Modeling early will help us identify risks sooner. We should also move toward a zero-trust model and constantly monitor our systems for incidents.  These actions will help us stay ahead of threats and improve our overall security. |
| **13** | Finally, to strengthen our security, we need to develop a clear incident response plan and focus on system hardening before deployment. Regular security audits and testing will help us stay on top of emerging threats. By fostering a culture of security awareness and updating policies based on feedback, we can continuously improve our approach and keep our systems secure. |