# CS 405 Project Two Security Policy Presentation

Mark Hall

Oct 15, 2024

[**https://youtu.be/gxBWjQHubjQ**](https://youtu.be/gxBWjQHubjQ)

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Welcome to this security policy presentation. Presented by: Mark Hall |
| **2** | This policy integrates security throughout all phases of software development. Our defense-in-depth approach adds multiple protective layers—covering physical, cloud, and network systems. The goal is to enforce secure coding, risk assessments, and encryption policies to maintain a resilient security posture for Green Pace. |
| **3** | This matrix summarizes potential threats. We assess SQL injections and weak passwords as likely risks, making them a top focus. Prioritization also includes encryption gaps and improper access control. Though unlikely, we monitor for rare buffer overflows and race conditions as well. |
| **4** | Our policy follows 10 core security principles that map directly to coding standards. For example, ‘Validate Input Data’ aligns with preventing SQL injections and buffer overflows, while ‘Heed Compiler Warnings’ ensures early detection of potential bugs. This alignment ensures a secure development process. |
| **5** | We’ve prioritized our standards based on the severity of vulnerabilities they address. Input validation, encryption, and SQL injection prevention rank highest due to their potential impact. Medium-priority standards include safe memory practices and exception handling, critical for system stability. |
| **6** | We apply encryption policies at three stages—data in flight, at rest, and in use. This ensures protection throughout the data lifecycle, minimizing exposure to unauthorized access or leaks. |
| **7** | Our policy enforces authentication, authorization, and accounting. This ensures only authorized users access the system, their actions are logged, and we have complete visibility into system activity. |
| **8** | Unit testing plays a critical role in maintaining code quality and catching bugs early. By focusing on individual components, unit testing ensures that each piece of the application performs as expected before integration with other modules. This allows us to identify potential issues before they escalate. |
| **9** | This test checks for valid input. We expect the system to process the input without errors. The test verifies that input handling functions correctly under normal conditions. |
| **10** | This test introduces an SQL injection attempt to see if the input validation mechanisms can prevent it. The expected result is that the system rejects the input or neutralizes any malicious code, ensuring the database remains secure and unaffected. |
| **11** | In this test, we provide an empty input to verify how the system responds. The expected behavior is either rejecting the input with a proper error message or accepting it if empty input is valid for that context. This ensures the system gracefully handles such cases. |
| **12** | This test uses special characters as input to test the application’s ability to sanitize and properly handle unexpected symbols. The system should either accept the input if it's valid or flag it as invalid, preventing unintended behavior or vulnerabilities. |
| **13** | Here, we provide a very long string to test the system's input handling limits. The goal is to ensure the application can either process or reject excessively long input without crashing or becoming unresponsive. This helps prevent buffer overflow vulnerabilities. |
| **14** | In this test, we simulate a buffer overflow attempt by providing input larger than the system can handle. The expected behavior is for the system to block the input and handle it safely without causing any memory corruption or crashes, ensuring stability. |
| **15** | Automation is built into our DevSecOps pipeline. Tools like SonarQube handle static code analysis during development, while Splunk provides continuous monitoring after deployment. |
| **16** | Our DevSecOps pipeline leverages automation tools at different stages to maintain security. SonarQube is integrated during the build phase for static code analysis to catch vulnerabilities early in development. Splunk or the ELK stack provides real-time monitoring and log management during production, allowing us to detect anomalies and respond to potential security incidents promptly. |
| **17** | There are several challenges we need to address. Delaying security measures increases the risk of exploits, which could lead to breaches, financial losses, and damage to our reputation. Additionally, without continuous monitoring, vulnerabilities may go unnoticed, increasing system exposure over time.  To address these issues, we propose the immediate integration of encryption, authentication, and authorization policies. Tools like SonarQube for static code analysis and Splunk for monitoring will ensure system integrity. However, adding these security measures will increase development time and setup complexity. |
| **18** | The benefits of acting now are clear. Early integration of security ensures compliance and provides long-term protection against potential attacks. It’s also more cost-effective to prevent issues now than to address breaches later.  However, there are still gaps we need to address, such as ensuring comprehensive testing coverage at every stage of development. If we wait, the risks will grow, as vulnerabilities will remain unaddressed. To avoid this, we recommend integrating security tools immediately, conducting risk assessments to prioritize vulnerabilities, and providing team training in secure coding and tool usage. |
| **19** | We recommend regular security training, improved incident response plans, and enhanced version control. Reviewing legacy code and updating encryption protocols will further strengthen our security posture. |
| **20** | In conclusion, secure coding standards, continuous integration of security tools, and strong access controls will help us stay ahead of potential vulnerabilities. Regular risk assessments will ensure ongoing compliance and readiness. |