# CS 405 Project Two Script

Douglas Rowland

CS405

2/22/2025

<https://youtu.be/xPrWjYmLCJ8>

https://www.youtube.com/watch?v=xPrWjYmLCJ8

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello everyone, my name is Douglas Rowland, and today I will be presenting the Green Pace Security Policy. This presentation will cover the importance of secure coding, the threats we face, and the policies and tools we use to protect our systems |
| **2** | The Green Pace Security Policy standardizes secure coding practices to mitigate risks and ensure compliance. By integrating security into our DevSecOps pipeline, we create a secure, multi-layered defense that enforces security policies through automation and systematic protection. |
| **3** | This matrix categorizes potential threats based on likelihood and priority. High-priority threats include SQL injection and memory corruption, which can cause significant damage. Medium-priority risks involve input validation and random number generation. Lower-priority risks, such as assertions and error handling, still enhance security but are not immediate threats |
| **4** | We adhere to 10 key security principles, such as validating input data to prevent injection attacks, keeping systems simple to minimize vulnerabilities, enforcing the principle of least privilege, and adopting secure coding standards. Each principle aligns with industry best practices to fortify our software development lifecycle |
| **5** | Our coding standards enforce best practices to prevent vulnerabilities. For example, validating input length avoids buffer overflows, using parameterized queries prevents SQL injection, and banning unsafe functions like strcpy() enhances memory safety. These standards are prioritized based on their impact, with high-priority rules addressing the most critical security concerns. |
| **6** | Data security is critical at every stage. Encryption in flight protects data during transmission using TLS, encryption at rest secures stored data using AES-256, and encryption in use safeguards active data through advanced memory encryption techniques like Intel SGX |
| **7** | Authentication, authorization, and accounting form the core of access control. Multi-factor authentication ensures secure logins, role-based access control limits privileges, and activity logging helps with auditing and compliance |
| **8** | To verify our security measures, we conducted unit tests. For example, we tested whether input validation prevents buffer overflows, whether hardcoded credentials exist in the code, and whether SQL injection protection is effective. Each test provides insight into the strengths and weaknesses of our security controls. |
| **9** | One of the most critical security measures in software development is validating input length to prevent buffer overflows. In this test, we provided an input string longer than the allowed buffer size to check whether the system properly restricts it. Ideally, the system should either truncate the input safely or reject it altogether. However, as we can see from the test result, this validation failed, meaning there is a risk of buffer overflow, which could lead to memory corruption or system crashes. To fix this, we need to implement strict input validation to limit the number of characters accepted and enforce safer string handling practices |
| **10** | Hardcoded credentials, such as usernames and passwords stored directly in the source code, pose a major security risk. Attackers who gain access to the source code can easily extract these credentials and use them for unauthorized access. This test was designed to detect any hardcoded sensitive data in the codebase. The result confirms that hardcoded credentials were found, meaning a security risk is present. To mitigate this, credentials should be stored securely using environment variables or a secrets management tool, ensuring they are not exposed in the code |
| **11** | SQL injection is one of the most common vulnerabilities in web applications. This occurs when user input is not properly sanitized, allowing attackers to manipulate database queries. In this test, we attempted to inject a malicious SQL command to see if the system properly filters and sanitizes input. The test passed, confirming that the implemented security measures successfully prevent SQL injection. This means that user input is being properly sanitized, and parameterized queries are being used, which is a critical best practice in database security. |
| **12** | Memory management issues, such as accessing freed memory or using uninitialized pointers, can lead to program crashes or exploitable vulnerabilities. In this test, we simulated a potential use-after-free scenario to verify if the system properly handles memory allocation and deallocation. The test passed, indicating that the system correctly enforces memory safety, reducing the risk of crashes or security exploits. Proper memory management is essential in preventing vulnerabilities that attackers could leverage to execute arbitrary code or cause system failures. |
| **13** | This test verifies that unsafe C functions, such as strcpy(), are avoided in favor of safer alternatives. Functions like strcpy() do not perform bounds checking, which can lead to buffer overflows and security vulnerabilities. Instead, we used strcpy\_s(), which includes built-in safeguards to prevent overruns by requiring a buffer size parameter. Since our test implemented the secure function correctly, it passed, confirming that the safer approach was used. This ensures better memory safety and helps prevent common vulnerabilities related to unsafe string handling |
| **14** | Automation plays a crucial role in security by integrating security testing and monitoring into the development process. Security is enforced continuously, reducing the risk of vulnerabilities slipping into production. |
| **15** | We use various security tools throughout our DevSecOps pipeline. Static Analysis tools (SAST) scan source code for vulnerabilities, while Dynamic Analysis (DAST) tests running applications. Software Composition Analysis (SCA) checks dependencies for security flaws, and Intrusion Detection Systems (IDS) monitor production environments for anomalies. |
| **16** | Delaying security measures increases exposure to threats and raises recovery costs. A proactive approach minimizes attack surfaces and ensures compliance. Our strategy must emphasize continuous monitoring, penetration testing, and ongoing training to maintain a resilient security posture. |
| **17** | To further strengthen our security policy, we must improve real-time threat detection, enhance user training, enforce secure coding through automated tools, and develop a structured incident response plan to mitigate potential breaches |
| **18** | To ensure long-term security, we must adopt OWASP Secure Coding Practices, the NIST Cybersecurity Framework, and CWE/SANS Top 25 security guidelines. Implementing Zero Trust Architecture and continuous security testing will reinforce our security posture against evolving threats. |
| **19** | These references provide further insights into secure coding and DevSecOps best practices. They include resources from AWS, Microsoft Azure, and the Department of Defense DevSecOps guidebook  Thank you for your time. Security is an ongoing process, and by implementing these measures, we can safeguard our systems against emerging threats. I am happy to take any questions you may have. |