# CS 405 Project Two Script Template

Complete this template by replacing the bracketed text with the relevant information.

https://www.youtube.com/watch?v=CJjgFlqpHr4

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is Liam McAveney. This presentation is for Project Two in CS 405 and explains the Green Pace Security Policy I created to guide secure development. |
| **2** | The Green Pace Security Policy was made to help all development teams follow clear and consistent security steps. It follows a defense in depth approach, adding layers of protection at every phase of development. With rules for input checks, memory handling, encryption, and testing, the policy helps reduce risk and improve system safety. |
| **3** | This matrix shows major threats found in the coding rules. Problems like unsafe data types, SQL injection, and log injection are high risk. Others, like poor string handling or weak exception checks, still matter but are less urgent. Tools like SonarQube, Coverity, and clang tidy help spot and fix these with automated scans. |
| **4** | The policy follows ten security principles. Each one connects to one or more coding rules. For example, 'Validate Input Data' relates to rules like STD-001 and STD-004. 'Manage Resources Securely' focuses on memory and exception use. The last principle, 'Adopt a Secure Coding Standard,' supports the entire list. |
| **5** | The coding rules are ranked by risk, based on how likely and harmful the problems are. Unsafe data types and SQL injection are the biggest risks. Other rules deal with memory safety, thread safety, and better error handling. Ranking helps the team fix the most serious problems first. |
| **6** | The encryption plan protects data through all stages. Stored data uses AES 256 and key changes every 90 days. Data in motion uses TLS 1.3, with checks between services. Data being used stays in protected memory, and if possible, uses secure hardware like Intel SGX. |
| **7** | Authentication is done with single sign on and multi factor login. Services use short term tokens or certificates. Authorization uses role based access and is reviewed often. Accounting means recording user actions and system changes, with logs stored for at least one year. |
| **8** | To test the rules, I built six unit tests focused on array index limits. Each test uses a question like "Is this input safe?" and shows if the system accepts or blocks it. This proves how the code handles unsafe inputs. |
| **9** | This test uses index 3 on a 10-size array. The system stores the value and passes the test. |
| **10** | Here, the index is -1. The system throws an out of range error, which is correct. The test passes. |
| **11** | This test uses index 10 on an array with 10 spots. That is too high, so it fails as expected. The test passes. |
| **12** | The input is 15, which is beyond the array. The system blocks it and throws an error. The test passes. |
| **13** | Index 9 is the last valid spot. The value is saved, and the test passes. |
| **14** | A letter is used as the index. The system rejects it or shows a compile error. The test passes. |
| **15** | The DevSecOps pipeline adds security at every stage of development. It begins with planning, where security rules are included in system templates. During builds, tools scan code for unsafe patterns. In testing, unit tests and security scans help find any issues. After release, the system watches logs and reacts to threats.Tools include Coverity Scan, clang tidy, SonarQube, CodeQL, OWASP ZAP, and a SIEM system to track logs and alert the team if needed. |
| **16** | This slide highlights the key tools used across the DevSecOps pipeline. During development, tools like Coverity Scan and clang tidy help catch unsafe code early. When code is committed, SonarQube checks for quality issues and security flaws. For deeper scans, CodeQL analyzes the codebase for logic errors and missing input checks. In the staging phase, OWASP ZAP is used to test web interfaces for security holes. After deployment, a SIEM system tracks logs, looks for unusual activity, and alerts the team. These tools work together to protect the system from start to finish. |
| **17** | Delaying fixes raises the risk of attacks. Problems like SQL injection or memory issues can cause serious damage. Fixing them early lowers costs and keeps users safe. Logging and third party checks still need work and should be improved soon. |
| **18** | Check outside libraries more often. Give developers regular training. Improve log coverage across all parts of the system. Review old code. Update the policy once a year. |
| **19** | Next, the policy should cover third party use, cloud protection, and secure key storage. Also, remove unused code when possible. Keeping the policy current will help stop new threats and improve long term security. |