# CS 405 Project Two Script

Giovanni Oddo

April 20, 2025

Link: <https://youtu.be/zwsbhNv09nc>

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Welcome to the Green Pace Security Policy Presentation. I’m Giovanni, and in this presentation I’ll go over topics in our secure coding policy, explain our defense-in-depth strategy, and outline the improvements made to the policy. |
| **2** | Our secure coding policy at Green Pace uses a defense-in-depth approach, implementing layered protections at every stage of the SDLC. This strategy helps us minimize vulnerabilities while maximizing automation, lowering remediation costs, and ensuring secure and stable application releases through DevSecOps practices. |
| **3** | This slide introduces our threat matrix, which categorizes security threats by likelihood and severity. It helps prioritize our remediation efforts by identifying which threats are probable and impactful, allowing developers and stakeholders to identify and address the most dangerous risks first. |
| **4** | The foundation of our policy is based on ten core principles, including  input validation, standards that apply are 2 3 4 9  heed compiler warnings, standards that apply are 1 2  architect and design for security policies, standards that apply are 6 7 10  keeping it simple, standards that apply are 3 4 6 19  default deny, standards that apply are 8  Adhere to the principle of least privilege, applies to AAA policy  Sanitize data sent to other systems, standard 4  Practice defense-in-depth, standard 6 7 10  Use effective QA techniques, standards 5 9  And Adopt a secure coding standard standards 1 2 5 8  These principles guide security and development decisions made throughout the software lifecycle. |
| **5** | The standards are ordered by the highest priority first, then by highest severity, and finally by highest remediation cost. I chose this order to allow teams to focus on the highest risk and remediation cost items first. |
| **6** | We define three types of encryption: at rest, in flight, and in use.  At rest – encrypts stored data to prevent data breaches  In flight – uses secure transfer protocols such as HTTPS, TLS/SSL and SNI to safely transfer data.  In use – ensures that authorized users can perform actions on data. |
| **7** | We also use the Triple-A framework: Authentication to confirm identity, Authorization to enforce access rights, and Accounting to log actions and support audits. This ensures data confidentiality and system accountability. |
| **8** | CanAddFiveValuesToVector verifies that elements can be safely appended and stored in a collection. We assert that the collection size is empty, add 5 entries into the container and then assert that the new size is equal to the number of entries that were added. |
| **9** | CanDecreaseCollectionSizeToZero ensures that operations like pop\_back() don’t lead to underflows. We start with our empty collection and add 5 entries to it. Next store the collection size before we resize it. Resize the collection to 0 and then assert that the size is equal to zero. |
| **10** | IsOutOfRangeIndex checks for exception handling when accessing elements outside valid bounds to prevent undefined behavior. First we add entries to the collection and expect that the size is greater than 0 but less than 15. Then assert throw an out of range error by checking for an index way outside the collection size. |
| **11** | IsReserveBeyondMaxSize tests how the application responds when trying to allocate more memory than the container supports. We expect the capacity of the newly created collection to be less than the max size of the collection since it is empty. Then try to reserve capacity by 1 beyond the max size allowed which will throw a length error. |
| **12** | The DevSecOps pipeline integrates security practices at each stage of the software development lifecycle to maximize software security and data integrity. This approach to development allows security to be a focal point of discussion and integration as a project progresses with the end goal of mitigating as many vulnerabilities as possible by deployment. |
| **13** | * + Asses and Plan – OWASP and Threat Modeler can identify early threats and provide solutions for mitigation.   + Design – Google Test and OWASP Top 10 can be used to create scaffolding for tests specifying methods to be used in production.   + Build – CMake and Clang can provide trusted repositories and architecture for secure coding.   + Verify and Test – Parasoft C++ Test and CodeSonar tools can be used to automate unit and static testing.   + Transition and Health Check – Docker can be used for penetration testing in a secure container to check for any other vulnerabilities before deployment   + Monitor and Detect – Tools such as Splunk and SIEM monitor for abnormal activities and log all activity to trace where a vulnerability may be.   + Respond – Custom scripts can be used to stop or turn services off during an attack   + Maintain and Stabilize – constantly test and update after an incident to ensure system stability |
| **14** | We compare two approaches: waiting to address security or acting now. While prevention has upfront costs, it reduces vulnerabilities and saves time later. Conversely, delaying security introduces higher remediation costs and delays product delivery. Prevention supports long-term efficiency and system resilience. |
| **15** | Gaps in the current security policy   * + - Add standards for threat modeling in the assess and plan phase     - Specify which events to be logged and add specific tools to be used     - Develop security training schedule for developers     - Add platform specific standards. |
| **16** | Given our gaps in the security policy, we will adopt the following standards to prevent future problems:   * Initial and Annual refresher for security training using OWASP Top 10 vulnerabilities and project specific vulnerabilities. * Log all pertinent events to catch active attacks or outlier behavior * Highlight key practices with cloud, web, and mobile applications |