# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello everyone. Today I’ll be presenting the Green Pace Security Policy, which formalizes the best practices our team already uses and ensures consistency as we grow. My name is Evann Hopkins, and this policy represents our commitment to building secure systems from the ground up. |
| **2** | Our security policy adopts a defense in depth strategy, meaning we protect systems at multiple layers. This is critical because no single solution can stop every threat. For example, even if a hacker bypasses the firewall, encryption and input validation still safeguard our data. The threat matrix on the next slide breaks down the risks we’ve prioritized. |
| **3** | Our matrix categorizes vulnerabilities by likelihood and impact. SQL injection is marked critical because it’s common and devastating. Broken authentication is high priority since compromised accounts threaten all user data. Lower risks, like outdated libraries, still matter but get scheduled fixes. Automation tools like SonarQube will flag these issues early. |
| **4** | These principles guide every decision. Principle one mandates input validation to block attacks, which ties to Standard one. Principle two enforces multi factor authentication, aligning with Standard two. Each principle maps to specific standards, ensuring we address both code and architecture risks. |
| **5** | Standards are ranked by impact. Input validation tops the list because flaws here cause data breaches. Secure authentication follows since weak logins compromise entire systems. Lower priority standards, like session timeouts, still matter but won’t cripple us if delayed. This ranking reflects real world exploit data from OWASP. |
| **6** | Data is encrypted three ways. In transit, TLS 1.3 secures all communications. At rest, AES 256 protects databases. In use, memory encryption prevents leaks during processing. This layered approach means even if one layer fails, others keep data safe. |
| **7** | Authentication requires MFA for all admins. Authorization uses role based access so users only see what they need. Accounting logs every action for audits. Together, these ensure only the right people access the right data, and we can trace any issues. |
| **8** | Let’s test Standard one. This C++ test verifies input validation blocks SQLi. The payload ' OR 1=1-- gets rejected, which passes. But Test three failed when users accessed admin endpoints. We fixed this by adding role checks. Each test mirrors our principles. |
| **9** | Security tools are embedded in our workflow. Pre production, we model threats. During builds, SonarQube scans code. Testing includes OWASP ZAP for live attacks. Post deployment, Splunk monitors logs. Automation catches what manual reviews miss. |
| **10** | Fixing critical bugs like SQLi now prevents breaches but slows features. Delaying risks hacks but keeps deadlines. Our policy balances this by patching critical flaws immediately and scheduling others in sprints. |
| **11** | Current gaps include third party risks. We should adopt software bills of materials to track dependencies. Future focus? Threat modeling workshops every quarter to stay ahead of new attack methods. |
| **12** | Our policy draws from OWASP Top 10 and NIST guidelines, shown here. These are industry standards for a reason they work. |
| **13** | Thanks for your time. Let’s discuss how to implement these steps in your workflows. Together, we’ll build systems that are secure by design. |