# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Welcome to the Green Pace Security Policy presentation. In this presentation, we will cover our security strategy, including coding standards, risk assessment, and automation within DevSecOps to ensure compliance and long-term security. |
| **2** | Green Pace is committed to a Defense in Depth strategy, where security is implemented at multiple layers throughout the development lifecycle. By enforcing secure coding standards, using automation for compliance, and applying industry best practices, we create a strong security posture. |
| **3** | Our threat assessment categorizes vulnerabilities by likelihood and priority. Critical risks, such as SQL injection and memory safety issues, require immediate attention, while lower-priority concerns, like assertions and random number generation, are still addressed systematically. |
| **4** | Our security framework is built upon 10 core principles, such as validate input data, enforce least privilege, and architect for security. These principles align with our coding standards to mitigate vulnerabilities in development. |
| **5** | We have prioritized 10 coding standards based on risk level. SQL injection and buffer overflow protections are top priorities, while other areas, such as exception handling and assertions, ensure stability and best practices. |
| **6** | Encryption is applied at rest, in transit, and in use to ensure data confidentiality. We implement AES-256 for stored data, TLS for transmission security, and memory encryption for data in use. |
| **7** | Authentication, Authorization, and Accounting (AAA) ensure user access is properly managed. Multi-factor authentication (MFA), role-based access control (RBAC), and logging mechanisms prevent unauthorized access and track system activity. |
| **8** | SQL injection is a common and dangerous vulnerability that can allow attackers to manipulate queries and gain unauthorized access to data. Our unit test ensures that parameterized queries are used to prevent malicious input from altering database commands. By testing various inputs—including valid credentials and common SQL injection attempts—we verify that injection attacks are successfully blocked. This test is an essential safeguard for any system handling user input and database queries. |
| **9** | Improper string handling can lead to buffer overflows, potentially allowing attackers to execute arbitrary code. This unit test ensures that all string operations use bounded functions, such as strncpy instead of strcpy, to prevent overflows and memory corruption. The test verifies that input strings remain within the allocated buffer size, reducing the risk of security breaches due to unbounded data manipulation. |
| **10** | Strong authentication controls are vital to securing user access. Our unit test simulates login attempts with valid and invalid credentials to verify that only authorized users can access the system. It also checks for brute-force protection, ensuring that repeated failed login attempts trigger appropriate security responses, such as temporary account locks or CAPTCHA challenges. Proper authentication validation is crucial for preventing unauthorized system access. |
| **11** | Memory management vulnerabilities, such as use-after-free and memory leaks, can lead to crashes and exploitability. This unit test ensures that dynamically allocated memory is properly allocated and freed without causing dangling pointers or memory corruption. By integrating these tests into our CI/CD pipeline, we can detect and remediate memory-related issues before they impact production. |
| **12** | Unauthorized file access can lead to data leaks and privilege escalation. Our unit test verifies that only authorized users can read, write, or modify restricted files. It checks for proper role-based access controls (RBAC) and ensures that users without permission are denied access. This security measure prevents data breaches and unauthorized modifications to critical system files. |
| **13** | Automation plays a crucial role in securing the DevSecOps pipeline. As shown in the diagram, security measures are integrated at multiple stages, from planning and design to deployment and monitoring. Automated tools like SAST, DAST, SIEM, and SOAR enable real-time vulnerability detection and mitigation. By embedding security automation throughout the pipeline, Green Pace ensures continuous compliance and proactive defense against cyber threats. |
| **14** | Security is embedded at every stage of DevSecOps to detect and mitigate vulnerabilities early. We use SAST, DAST, SIEM tools, IDS/IPS, and automated compliance frameworks to enforce security policies throughout development and production. |
| **15** | While our security strategy is robust, delaying implementation increases breach risks, and over-reliance on automation can miss complex threats. Our approach balances automation with manual oversight to address these concerns. |
| **16** | To strengthen security, we recommend ongoing training, improved real-time threat intelligence, stricter access control, and enhanced incident response testing. These measures ensure continuous improvement and resilience against evolving threats. |
| **17** | Green Pace’s security policy provides a proactive approach to software security by integrating security throughout the DevSecOps pipeline. By continuously refining our strategy and leveraging automation, we ensure a strong, future-proof security posture while supporting innovation.  This concludes our presentation. Thank you for your time. |