# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, everyone. My name is Mohamed, and today I will be presenting the security policy for Greenpeace. |
| **2** | The Green pace Security Policy ensures that all developers follow consistent secure coding and system practices. As the team grows, it keeps everyone aligned. The policy supports a defense-in-depth strategy by implementing multiple layers of security, such as coding standards, encryption, testing, and access control, to reduce risks and protect against attacks |
| **3** | The THREATS MATRIX is divided into four categories: Likely, High Priority, Low Priority, and Unlikely. The likelihood of a threat should determine how frequently it is monitored and whether detection can be automated. High-priority threats must be addressed immediately, as they pose the greatest risks to security and product stability. By prioritizing the most common and damaging threats, we can reduce vulnerabilities and strengthen our overall security posture. |
| **4** | These are the ten core secure coding principles we follow. First, we validate input data to prevent injection attacks. We heed compiler warnings to catch issues early. We architect systems with security in mind and keep designs simple to reduce complexity. Our default is to deny access unless explicitly allowed. We follow the principle of least privilege, ensuring users and systems only have access to what they need. We sanitize all data shared with external systems. We also practice defense in depth, using multiple layers of security. Quality assurance is built into our process, and finally, we adopt a secure coding standard to maintain consistency and best practices across the team |
| **5** | These ten coding standards help us write secure, stable code. They focus on safe data handling, preventing overflows and injections, protecting memory, handling errors properly, and making code easier to understand and maintain. |
| **6** | Encryption at rest protects stored data—even if someone gets physical access to the system. Encryption in transit secures data as it moves between client and server, making it unreadable if intercepted. Encryption in use protects data while it's being processed, ensuring it's secure at every stage |
| **7** | Authentication means verifying user identity with strong credentials and MFA. Authorization ensures users only access what they need, using least privilege and regular permission reviews. Accounting involves logging activity, monitoring for unusual behavior, and using tools to track and report actions. |
| **8** | "Unit testing catches issues early by checking each part of the code for errors and vulnerabilities. It ensures safe input handling, proper memory use, and expected function behavior—stopping bugs before they spread. |
| **9** | Automation speeds up security checks and ensures consistency. Static and dynamic analysis tools automatically scan for issues during coding and testing. CI/CD pipelines integrate these tools to catch vulnerabilities early, reducing manual effort and human error. |
| **10** | DevSecOps adds security at every stage—coding, building, testing, and deploying. It catches issues early using tools like static analysis and automated tests.  Tools like Astrée check code during development. Jenkins or GitLab CI automate testing and deployment with built-in security. OWASP ZAP tests live apps for vulnerabilities |
| **11** | Early integration of security in development helps reduce vulnerabilities before they can be exploited. It also improves code quality and system stability by embedding secure practices throughout the process. This proactive approach minimizes the risk of data breaches and other threats. However, there are some drawbacks, such as the initial cost of tools and training. Teams also need time to adapt to new workflows, which can slow progress at first but pays off in long-term security. |
| **12** | Integrating security early and continuously throughout the development lifecycle helps reduce risks from the start. Automating security testing can minimize human error and ensure consistent protection. Standardized code improves readability, consistency, and maintainability, making it easier to spot vulnerabilities. Ongoing secure coding training for developers ensures the team stays up to date on best practices. A feedback loop should be established to update policies based on real incidents and audits. As threats evolve, policies must adapt to stay effective. This guide is a starting point, and additional standards will be introduced as needed to address future challenges |
| **13** | Security is something we always need to work on as new threats appear. Using trusted standards like OWASP and NIST makes our system stronger. Writing secure code and using layers of protection helps stop problems early. DevSecOps means we think about security at every step of development. By checking often and updating our rules, we stay ahead of threats. This way, we build safer apps and earn users’ trust. |
| **14** | Here are the key sources I used to support the information throughout this presentation. |