# 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 Oleksiy Koshlyak, and today I will present the brand-new security policy designed to prevent security vulnerabilities in both code development and systems architecture. This policy will play a critical role in maintaining a robust defense-in-depth strategy to safeguard our software and systems. |
| **2** | Our new security policy addresses the need for more comprehensive protection against coding vulnerabilities and system threats. By adhering to these policies, we can significantly reduce potential risks. This policy is designed to be repeatable, clear, and ready for immediate implementation. It aligns with industry best practices and supports our overarching defense-in-depth approach, ensuring that security is embedded at every layer of our development process. |
| **3** | This slide presents the Threats Matrix, which identifies and categorizes the most critical coding vulnerabilities that we've identified. Each threat is mapped to its corresponding coding standard to ensure that we're addressing every area of concern. We've also included an explanation of how automated tools can be utilized to detect and manage these coding vulnerabilities efficiently. |
| **4** | Our security policy is built on 10 fundamental principles that guide our coding practices and ensure alignment with our security goals. These principles include the enforcement of least privilege, input validation, secure session handling, and more. Each principle is tied to specific coding standards that ensure consistency and thoroughness throughout our development process. |
| **5** | The 10 coding standards in this slide are listed in order of priority. We've prioritized them based on the severity of the vulnerabilities they address and their potential impact on the system. For example, input validation and sanitization have been given the highest priority due to the critical role they play in preventing injection attacks. By following these standards, we can ensure that our code is secure from the outset. |
| **6** | Our encryption strategy focuses on three key areas: data in flight, data at rest, and data in use. We ensure that sensitive data is always encrypted during transmission, storage, and usage, using industry-standard encryption algorithms. This layered approach to encryption is essential for preventing unauthorized access and maintaining data confidentiality. |
| **7** | This slide outlines our policies for authentication, authorization, and accounting. Strong authentication mechanisms ensure that only authorized users gain access to our systems, while fine-grained authorization controls determine the actions that users can perform. Accounting ensures that we have a comprehensive audit trail of all system activities, enhancing transparency and accountability. |
| **8** | Unit testing is a vital component of our security strategy. Each test case is designed to validate that individual components of our code are functioning as expected and are free from vulnerabilities. This slide highlights key unit testing strategies and shows how they can be extended to include security-related test cases. By using automated testing frameworks, we can ensure thorough, continuous validation of our code. |
| **9** | Here, we demonstrate how to apply unit testing frameworks, such as JUnit and NUnit, to our security policy. These frameworks can be integrated into our continuous integration pipeline to catch vulnerabilities early in the development process. Automated security testing ensures that we identify and address potential issues before they make it into production. |
| **10** | This slide explains where our security tools fit into the DevSecOps flow. From static code analysis during development to automated vulnerability scanning in the build stage, and finally, penetration testing in the deployment stage, each security tool plays a crucial role in protecting our software. The diagram also illustrates when the compiler will be used in conjunction with these tools. |
| **11** | If we act now and implement these security measures, the immediate benefit is a significant reduction in our exposure to vulnerabilities. However, if we delay, the risks include higher costs associated with patching security issues later and potential damage to our brand if vulnerabilities are exploited. Implementing these measures now will also allow us to stay ahead of regulatory requirements. |
| **12** | Our current security policy has gaps in areas such as secure code reviews and automated testing for all types of vulnerabilities. Moving forward, we recommend adopting industry standards like OWASP and integrating these into our development workflows. This will help prevent future issues, as threats continue to evolve and our software needs to stay resilient. |
| **13** | In conclusion, this security policy provides a solid foundation for protecting our software and systems from potential vulnerabilities. By following these best practices, standards, and principles, we can ensure the continued safety and reliability of our codebase. It's crucial to act now to prevent future risks and ensure compliance with security regulations. |
| **14** | OWASP Foundation. (2021). OWASP Top Ten Web Application Security Risks. https://owasp.org/www-project-top-ten/    National Institute of Standards and Technology (NIST). (2020). Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1. https://doi.org/10.6028/NIST.CSWP.04162018 |