# CS 405 Project Two Script Template

Project 2: Security Presentation

04/21/2025

Video Link: <https://www.youtube.com/watch?v=XP2Lihixyz8>

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is Binaya Rimal. I work at Green Pace. I have been asked to present the Green Pace Security Policy Guide and provide implementation guidelines and future maintenance recommendations. |
| **2** | Our Security Policy serves as a framework and guiding documents to protect Green Pace’s assets from potential cyber threats. Cyberattacks target and exploit vulnerabilities across different areas of a system to gain unauthorized access and cause harm.  We implement a multi-layer security system to cover a wide range of risks. In this presentation, I will talk about some core policies that create a robust security system, which include:   * 10 Coding Principles * 10 Coding Standards * Encryption Policy * Tripple A * Encryption Policies |
| **3** | Here is a Threat Matrix, which outlines the likelihood or unlikelihood of security threats associated with each coding standard. It also highlights the priority level for addressing each one, helping us focus on the most critical vulnerabilities first.  Issues that fall under high likelihood and high priority should be given the highest importance during security triage and resolution. |
| **4** | Here are our coding principles with a brief description:   1. **Validate Input Data** Always validate untrusted input to prevent vulnerabilities like injection and buffer overflows. 2. **Heed Compiler Warnings** Use the highest compiler warnings and analysis tools to catch potential flaws early. 3. **Architect and Design for Security Policies** Integrate security features like encryption and authentication into the system design phase. 4. **Keep It Simple** Simpler designs reduce the chance of errors and are easier to secure. 5. **Default Deny** Deny access by default and only grant it when explicitly authorized. 6. **Least Privilege** Give users and processes the minimum access needed to perform their tasks. 7. **Sanitize Data Sent to Other Systems** Clean all outbound data to prevent injection and misuse by downstream systems. 8. **Defense in Depth** Layer multiple security controls so one can back up another if it fails. 9. **Effective Quality Assurance** Use testing and reviews to catch vulnerabilities before deployment. 10. **Secure Coding Standards** Follow language-specific secure coding practices to reduce known risks.   The policies are created and ranked based on the C++ official coding guidelines. |
| **5** | These are our 10 coding standard:   1. **Data Type** Use appropriate data types to ensure safe storage, interpretation, and operation on variables. 2. **Data Value** Always validate that data values fall within expected and secure ranges before use. 3. **String Correctness** Handle strings carefully to avoid buffer overflows and injection vulnerabilities. 4. **SQL Injections** Prevent SQL injection by using parameterized queries and proper input sanitization. 5. **Memory Protection** Manage memory securely to prevent leaks, corruption, and unauthorized access. 6. **Assertions** Use assertions to catch programming errors during development—not in production code. 7. **Exceptions** Handle exceptions properly to avoid exposing sensitive information or crashing the system. 8. **Input/Output** Validate all I/O operations to ensure data integrity and prevent file-based attacks. 9. **Overflow** Guard against integer and buffer overflows by checking limits and using safe functions. 10. **Object-Oriented Programming** Apply secure OOP practices like encapsulation and input validation to protect class behavior.   In terms of determining what is more or less important, we can refer to the Threat Matrix to identify which threats have the highest likelihood and priority. For instance, SQL Injection is categorized as both high likelihood and high priority, indicating that it is likely to be of high importance. |
| **6** | To protect data, we must encrypt it. Encryption is the process of converting information or data into a code that needs to be decoded to be understandable.   * Encryption at rest refers to securing data when it is stored in a database or a server. Stored data must be encrypted using secure and widely accepted methods such as Advanced Encryption Standard (AES) or Rivest–Shamir–Adleman (RSA) encryption to ensure confidentiality and prevent unauthorized access. * Encryption in flight refers to the practice of securing data while it is being transmitted between systems, such as between a client and a server. * Encryption in use refers to the protection of data while it is being processed, such as when it is being accessed, read, updated, etc. |
| **7** | The Triple A policy accounts for user access and activity:  The Three A stand for:   * Authentication the process of verifying a user’s identity using credentials such as a username and password, biometrics, or other forms of identification. * Authorization is the process of determining what actions a user is permitted to perform within a system after their identity has been authenticated. * Accounting involves monitoring and recording user activity within a system. It captures details such as login times, accessed URLs, IP-based location data, and the services used by each user. |
| **8** | Unit Test Help Us Detect vulnerabilities and errors in code early in the development process. Here are some example positive and negative Tests:  Positive Test: ResizeDecreaseCollection:  Tests that resizing a collection of size 10 down to 5 correctly reduces its size. |
| **9** | Positive Test: ReserveIncreasesCapacityNotSize  Tests that reserving more capacity increases capacity() but does not increase size(). |
| **10** | Negative Test: AtThrowsOutOfRange  Tests that calling at() with an out-of-bounds index throws an std::out\_of\_range exception. |
| **11** | Negative Test: ResizeWithInvalidNumber  Tests if resizing with a negative number throws an error. |
| **12** | DevSecOps is a workflow methodology that integrates security practices into every phase of the software development lifecycle, ensuring that security is considered from the initial design stages through development, deployment, and maintenance. |
| **13** | Automation is a critical component of any effective DevSecOps process. The current infrastructure could significantly benefit from implementing various automation techniques. A well-designed DevSecOps automation strategy should aim to fully automate most testing procedures, including code analysis, configuration management, and patching and vulnerability management. Several frameworks support automated testing. For instance, BeEF (Browser Exploitation Framework) can be used for browser-based security testing, while SQLMap is a powerful tool for detecting SQL injection vulnerabilities. These tools help identify and mitigate risks early in the development cycle. |
| **14** | There are many tools that help with automation in a DevSecOp pipeline, here are some popular ones:  Jenkins: An open-source automation server used to build, test, and deploy software continuously.  GitLab CI/CD: A built-in GitLab toolchain for automating the software delivery process using pipelines.  SonarQube: A platform that continuously inspects code quality and detects bugs, vulnerabilities, and code smells.  OWASP ZAP: An open-source security tool for finding vulnerabilities in web applications through automated scanning.  Terraform: An open-source Infrastructure as Code (IaC) tool used to provision and manage cloud infrastructure. |
| **15** | When it comes to risk, we recommend acting as quickly as possible.  The longer it takes to detect and mitigate vulnerabilities, the more time attackers have to exploit them. Delayed action increases the spread of unauthorized access and the overall damage done to systems, data, and reputation.  Acting **immediately** through automated threat detection and mitigation systems allows organizations to:   * Identify and isolate vulnerabilities early. * Reduce the attack surface. * Limit the impact of security breaches. * Maintain trust with stakeholders and users.   Risks:   * Overwhelming Alerts * False Positives * Insufficient Planning |
| **16** | Our policies do not include everything, here are some further recommendations:  **Employee Training and Physical Asset Protection**  There is no mention of employee training and how employees should act against threat such email phasing attacks.  **Mult Factor Authentication**  While authentication is mentioned, The security policy does not go into details on how important, MFA is for all critical systems or accounts.  **Cloud Infrastructure**  The policies do not properly discuss how to protect cloud infrastructure  **Artificial Intelligence**  The current policies do not address how artificial intelligence (AI) can be exploited by hackers, nor do they outline how AI can be leveraged by the company to enhance cybersecurity and defend against threats. |
| **17** | To conclude I would highly advocate for these two policies:  **For Future Improvements:**  Cloud Protection:  Cloud environments store vast amounts of data, including personal information, financial records, and intellectual property. Without strong cloud security, this data is vulnerable to breaches, leaks, or theft.  **Artificial Intelligence:**  AI can analyze massive volumes of network traffic and user behavior in real-time. It is able to learn hacker’s attack patterns, it can understand what a suspicious behavior is from normal behavior in a way that is intuitive and complex. It can detect suspicious activities such as:   * Unusual login patterns * Data exfiltration attempts * Access anomalies across time zones or locations * This means faster detection and response than traditional methods. |