# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, and welcome. My name is Juan Juvera and today we will be discussing our new security policy. |
| **2** | Our network security policy is designed to protect the integrity, confidentiality, and availability of our company's information assets. As cyber threats continue to evolve, it’s important that we establish a comprehensive defensive posture to safeguard against unauthorized access, data breaches, and other security incidents. This is needed more than ever due to the increasing reliance on digital systems and interconnected networks makes our organization vulnerable to cyber attacks. Without a formalized network security policy, we risk exposing sensitive data, compromising customer trust, and incurring financial losses. Therefore, this policy is necessary to mitigate these risks and ensure the continuous operation of our business. Our new network security policy aligns with the defense-in-depth best practice by implementing multiple layers of security controls at various points within our network infrastructure. Each layer complements the others, creating a comprehensive security framework that provides overlapping protection against potential threats.  The first is Perimeter Security: We implement firewalls, intrusion detection and prevention systems (IDS/IPS), and secure gateways to monitor and control traffic entering and leaving our network.  Next, we intend to implement Endpoint Protection across the board: All endpoints, including desktops, laptops, and mobile devices, are equipped with antivirus software, host-based firewalls, and endpoint detection and response (EDR) tools to defend against malware and other malicious activities.  Access Controls: Access to network resources and sensitive data is restricted based on the principle of least privilege. Role-based access controls (RBAC), multi-factor authentication (MFA), and strong password policies are enforced to ensure only authorized users can access critical assets.  Data Encryption: We will employ encryption protocols, such as SSL/TLS for data in transit and AES for data at rest, to protect the confidentiality and integrity of sensitive information.  Employee Awareness Training: Regular training sessions will be conducted to educate employees about cybersecurity best practices, phishing awareness, and social engineering tactics to reduce the risk of human error. |
| **3** |  |
| **4** | 1. Must validate all inputs from untrusted sources to prevent software vulnerabilities. Be vigilant for warning signs like environmental variables and command line arguments. 2. Use caution when compiling code, eliminating warnings by making code changes. Employ dynamic and static analysis tools to detect security issues and flaws. 3. Design software around existing security policies to ensure adherence. 4. Simplicity and smallness offer safer protection compared to complex systems. 5. Base access on the concept of permissions needed. 6. Grant permissions only for the duration and level necessary to complete a task. 7. Ensure data sent to other subsystems is sanitized to deter attacks and prevent further complications. 8. Employ multiple layers of security to create fallbacks if the primary defense is compromised. 9. Employ quality assurance techniques to identify and eliminate weaknesses in the defense. Security reviews and multiple testing phases are beneficial. 10. Follow a secure coding standard for the language and platform in use. |
| **5** | 1. Do not delete a polymorphic object without a virtual destructor\*\* - Justification: Deleting polymorphic objects without a virtual destructor can lead to undefined behavior and memory leaks, making it critical to prioritize.  2. Use static assertions to test the value of a constant expression\*\* - Justification: Static assertions ensure compile-time checks, improving code reliability and catching errors early in the development processs  3. Do not let exceptions escape from destructors or deallocation functions\*\* - Justification: Preventing exceptions from propagating ensures predictable program behavior and avoids resource leaks, enhancing code robustness.  4. Close files when they are no longer needed\*\* - Justification: Failing to close files can lead to resource exhaustion and potential security vulnerabilities, making it essential for proper resource management.  5. Preserve thread safety and liveness using condition variables\*\* - Justification: Ensuring thread safety and liveness with condition variables prevents race conditions and deadlocks, improving the reliability of concurrent programs.  6. Expressions used in assertions must not produce side effects\*\* - Justification: Side effects in assertions can lead to unexpected behavior and make debugging more challenging, emphasizing the importance of clarity and predictability in assertions.  7. Do not cast to an out-of-range enumeration value\*\* - Justification: Casting to out-of-range enumeration values can lead to undefined behavior and unexpected program states, prioritizing type safety and avoiding potential bugs.  8. Do not use pointer-to-member operators to access nonexistent members\*\* - Justification: Using pointer-to-member operators with nonexistent members results in undefined behavior, highlighting the need for code clarity and avoiding unnecessary risks.  9. Do not attempt to create a std::string from a null pointer\*\* - Justification: Attempting to create a string from a null pointer can lead to runtime errors and undefined behavior, underscoring the importance of defensive programming and avoiding potential crashes.  10. Honor replacement dynamic storage management requirements\*\* - Justification: While important for memory management, this guideline ranks lower as it's specific to certain contexts and may not apply universally. |
| **6** | 1. This will be done by hashing the information, rendering it useless to anyone without the key. 2. This encryption goal will be achieved by using common protocols on our web servers. 3. While this is traditionally a bit more difficult than the others, we will be implementing a novel approach. For this, will be using a proprietary tool that encrypts data within a computer's memory while it's being processed, ensuring that it remains protected even during processing or querying operations. |
| **7** | * The Triple A security framework, also known as AAA, stands for Authentication, Authorization, and Accounting. * Together, these three components form a comprehensive security framework that helps ensure only authorized users have access to resources, while also providing accountability and monitoring of user activities. |
| **8** | [Insert text.] |
| **9** | Security tools will be integrated into various stages of automation to ensure comprehensive protection throughout the development and deployment process.  They typically reside in stages such as code development, continuous integration/continuous deployment (CI/CD), and production.  In pre-prod, static analysis tools may be used to identify security vulnerabilities early in the development process.  During CI/CD (or prod), security automation tools like vulnerability scanners and code analysis tools can be integrated to automatically check for security issues in the codebase before deployment.  Additionally, security measures like encryption and access control are implemented in the production stage to protect data and systems from threats. The compiler is used during the code development stage to translate source code into machine code and can be augmented with security-focused plugins or extensions for additional security checks. |
| **10** | The DevSecOps pipeline integrates security practices into the DevOps workflow, ensuring that security is prioritized throughout the software development lifecycle. This pipeline typically consists of several stages, including development, testing, deployment, and monitoring. In each stage, security checks and controls are implemented to identify and mitigate vulnerabilities early in the process. For instance, in the development stage, static code analysis tools can scan code for security flaws, while in the testing stage, dynamic application security testing (DAST) tools can simulate attacks to uncover vulnerabilities.  External tools play a crucial role in various stages of the DevSecOps pipeline. For instance, in the development stage, version control systems like Git are used to manage source code, while static code analysis tools such as SonarQube or Veracode are employed to identify security issues in the codebase. In the testing phase, tools like OWASP ZAP or Burp Suite are used for dynamic application security testing to detect vulnerabilities in the running application. Additionally, in the deployment stage, infrastructure as code (IaC) tools such as Terraform or Ansible help maintain consistent and secure environments, while container scanning tools like Docker Security Scanning or Clair ensure that containerized applications are free from known vulnerabilities. These tools collectively enable teams to build, deploy, and maintain secure software efficiently within the DevSecOps pipeline. |
| **11** | [Insert text.] |
| **12** | Hypothetical Gaps in the Network Security Policy: Lack of Regular Security Audits and Assessments: While the proposed policy outlines various security measures, it may not include provisions for regular security audits and assessments to evaluate the effectiveness of implemented controls. Without periodic assessments, we may fail to identify vulnerabilities and weaknesses in our network infrastructure, leaving us susceptible to undetected threats.  2. Insufficient Incident Response Plan: At this time, the policy may lack a detailed incident response plan that outlines the steps to be taken in the event of a security breach. Without a clear plan in place, our ability to respond promptly and effectively to security incidents may be compromised, leading to prolonged downtime, data loss, and damage to our reputation.  3. Limited Integration of Threat Intelligence: There may be limited integration of threat intelligence sources into our network security strategy. Without real-time threat intelligence feeds and analysis, we may miss early indicators of potential threats and fail to adapt our defenses accordingly. This gap could result in delayed or inadequate responses to emerging cyber threats, putting our organization at risk. |
| **13** | [Insert text.] |
| **14** | [Insert text.] |