# CS 405 Project Two Script

Link to the YouTube project: <https://www.youtube.com/watch?v=w7Rl_-fzK_M>

Student: Irina Borozan

Date: August 17, 2024

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** |  |
| **2** | The Green Pace Security Policy is a comprehensive guide designed to protect our software development process against a wide range of security threats. As our team grows, maintaining consistent and secure coding practices is critical to safeguarding our applications and data.  This policy was developed to address the increasing complexity of modern security challenges, ensuring that our development practices are aligned with industry standards and best practices. By integrating this policy into our daily operations, we reinforce a defense-in-depth strategy, which layers multiple security measures to provide robust protection against vulnerabilities at every stage of the software development lifecycle. |
| **3** | The threat matrix provides a visual representation of security levels based on the likelihood and severity of potential security issues. It categorizes threats into different levels, starting with **Level 1**, which is the most critical. These high-severity threats require immediate attention due to their significant impact. On the other end of the spectrum, **Level 3** threats are considered low priority. These are less exploitable, with minimal potential loss, and thus do not demand urgent action. |
| **4** | The 10 principles of secure coding are essential for building robust and secure software. **Validating input data** (STD-001-CPP, STD-002-CPP, STD-003-CPP) ensures that all data entering the system is clean and safe from injection attacks. **Heeding compiler warnings** (STD-005-CPP, STD-008-CPP, STD-010-CPP) is crucial for catching potential issues early in the development process. **Architecting and designing for security policies** (STD-007-CPP) helps integrate security considerations from the ground up. **Keeping it simple** reduces complexity, making the code easier to secure and maintain. **Default deny** enforces a security-first approach by restricting access unless explicitly allowed, while **adhering to the principle of least privilege** limits user and system access to only what is necessary. **Sanitizing data sent to other systems** (STD-004-CPP) prevents data contamination and security breaches. **Practicing defense in depth** adds multiple layers of security to safeguard against threats. **Using effective quality assurance techniques** (STD-001-CPP to STD-010-CPP) ensures the software is rigorously tested for vulnerabilities. Finally, **adopting a secure coding standard** (STD-001-CPP, STD-004-CPP to STD-010-CPP) ensures consistency and adherence to best practices across the development team. |
| **5** | * **Critical Threat Mitigation:** The top priorities (STD-001-CPP to STD-005-CPP) are standards that directly address the most critical security vulnerabilities. These standards focus on preventing the most common and severe threats, such as input validation issues and privilege escalation, which are often the primary targets of attackers. * **Foundation for Secure Architecture:** Standards ranked in the middle (STD-003-CPP to STD-007-CPP) lay the groundwork for a secure system by ensuring that security is built into the architecture, data is sanitized, and access is appropriately controlled. These standards are vital for maintaining a secure environment but are dependent on the foundational safeguards being in place first. * **Quality Assurance and Ongoing Security:** The final standards (STD-008-CPP to STD-010-CPP) focus on maintaining and continuously improving security through defense-in-depth, comprehensive QA practices, and adherence to secure coding guidelines. These are essential for the long-term stability and security of the system, ensuring that all other measures are consistently upheld and refined. |
| **6** | * To safeguard sensitive information throughout its lifecycle, we implement comprehensive encryption policies that address data in three key states: **in flight**, **at rest**, and **in use**.  1. **Encryption in Flight** ensures that data being transmitted between systems or networks is secured to prevent unauthorized access or interception. This policy preserves the confidentiality and integrity of data during transmission by using protocols like TLS/SSL. 2. **Encryption at Rest** protects stored data, whether it resides in databases, files, or backups. This includes encrypting entire storage volumes or specific files, ensuring that even if the storage is compromised, sensitive information remains protected. 3. **Encryption in Use** applies to data that is actively being processed. It maintains data confidentiality during operations by securing it in memory or within applications, ensuring protection while the data is being accessed or manipulated by authorized processes.  * These encryption policies form the backbone of our data security strategy, providing layered protection across all stages of data handling. |
| **7** | * Introducing the **Triple-A (Authentication, Authorization, Accounting) Security Policies**, which are designed to ensure robust security across all user interactions within our systems:  1. **Authentication:** To verify user identities, we enforce multi-factor authentication (MFA) for all users. This approach requires multiple forms of verification, adding an extra layer of security beyond just passwords. Strong password policies are also mandated, requiring complex combinations and regular updates to minimize the risk of unauthorized access (NIST, 2020). 2. **Authorization:** Access to resources is controlled through the principle of least privilege, meaning users are granted only the permissions necessary for their specific roles. Role-based access control (RBAC) is implemented to enforce this, reducing the potential for security breaches by limiting access to sensitive information (NIST, 2020). 3. **Accounting:** To maintain accountability, comprehensive logging is employed to track all user activities. These logs create audit trails that can be reviewed regularly to detect and respond to any unauthorized or suspicious behavior, ensuring the integrity of the system (NIST, 2020).  * Together, these Triple-A policies create a secure environment that rigorously controls access, enforces accountability, and monitors user interactions to protect sensitive data and resources. |
| **8** | Unit testing is used to verify that individual components or units of code function correctly in isolation. Each unit typically represents a small, specific piece of functionality, such as a function, method, or class. The primary goals of unit testing are to ensure code reliability, detect bugs early, facilitate refactoring, support documentation, and promote code quality.  This unit test was used to check if resizing a collection that starts as empty(or 0), increases. The test asserts true wjem the collection size has been increased to 5. |
| **9** | This next Unit test verifies that push\_back function adds an element to the end of the collection. When we push\_back 42, the last item in the collection equals t42, which is true and the unit test passed. |
| **10** | This unit test verifies if the out\_of\_range exception is throuns hen calling an index out of bound. We can see that we started with an empty collection. Then we resized is at 10. And than check to see if the unit test will throw an out\_of\_range exception when we’re trying to access index at 11 (which is not available since the size of the collection is 10). |
| **11** | * The image depicts a DevSecOps workflow, which integrates security practices into the traditional DevOps process. The workflow is divided into two main phases: **Pre-production** and **Production**. * In the **Pre-production** phase, the process starts with assessing and planning for security by evaluating the threat landscape and regulatory changes. The design phase focuses on security-driven design practices, followed by secure building practices. Verification and testing are conducted through vulnerability scanning and functional testing. Transition and health checks involve configuring and deploying security settings. * In the **Production** phase, monitoring and detection mechanisms are employed to identify potential threats. If a security issue is detected, the response phase involves blocking attacks and rolling back to a secure state. Finally, the system is maintained and stabilized by assessing against security baselines and returning to a stable state after any compromise. * The diagram illustrates how security is embedded throughout the entire software development lifecycle, emphasizing continuous security assessment, testing, and monitoring. |
| **12** | In the DevSecOps pipeline diagram, each stage represents critical steps in secure software development, with specific tools enhancing security and automation throughout the process. Let’s see the tools used in each stage:  these tools exemplify how DevSecOps integrates security into every phase, ensuring continuous protection throughout the development lifecycle. |
| **13** | It is very important to understand when to act and when to wait while practicing security. Acting now is better than waiting because it allows for proactive mitigation of security risks, reducing the potential for costly breaches and data loss. Early action helps build stronger defenses, ensuring that vulnerabilities are addressed before they can be exploited. This not only saves money and resources in the long run but also protects the company’s reputation. Delaying action increases the likelihood of encountering more complex threats, making future interventions more difficult and expensive. |
| **14** | * + - The shortcomings in the policy involve unidentified risks. Green Pace must regularly assess emerging threats and revise the security policy accordingly.     - The policy should be treated as an evolving document, undergoing regular reviews and updates.     - Additionally, it is crucial to implement a training program for the updated security policy.     - Establish a system to audit compliance with the policy in code, and integrate the security policy into the code review process. |