# CS 405 Project Two Script

Jeremy Snow

December 13, 2024

Project Two: Security Policy Presentation

YouTube Video Link: <https://youtu.be/hu8O2_w_1z8>

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, and welcome to the Green Pace Security Policy Presentation. My name is Jeremy Snow, and today we will explore the key principles, secure coding standards, automation tools, and best practices that form the foundation of this policy. |
| **2** | The Green Pace Security Policy was established to address increasing cybersecurity threats, ensure regulatory compliance, and promote consistency across development processes. It defines secure coding standards, data encryption policies, and the Triple-A framework (Authentication, Authorization, and Accounting) to mitigate risks and prevent vulnerabilities. This policy supports the Defense-in-Depth best practice by implementing layered security measures, including input validation, memory handling, error detection, and encryption for data at rest, in transit, and in use. Automated enforcement through tools like static code analyzers and CI/CD pipelines ensures compliance while enhancing system integrity and resilience. The accompanying graphic illustrates the layered approach of Defense-in-Depth, highlighting areas such as physical, cloud, network, and application security. |
| **3** | Next, we’ll take a look at the threat matrix that categorizes the coding standards within the Green Pace Security Policy. This matrix organizes threats based on their priority and likelihood. Low-priority threats, which have minimal impact, are typically addressed during regular maintenance. In contrast, high-priority threats demand immediate attention due to their significant risks. The matrix also distinguishes between likely threats, which have a high probability of occurring, and unlikely threats, which are less probable but still require consideration in our security planning. |
| **4** | Let’s now focus on the 10 key principles for secure development within the Green Pace Security Policy, which serve as the foundation for our coding standards. Each principle is designed to promote security at every stage of development. Let’s go through them briefly:   1. Validate Input Data – Ensuring that all user inputs are properly sanitized prevents malicious data from entering the system. 2. Heed Compiler Warnings – Paying attention to compiler warnings helps identify potential vulnerabilities early. 3. Architect and Design for Security Policies – Security must be built into the system’s architecture and design from the very beginning. 4. Keep It Simple – Simplicity in coding reduces errors and makes security flaws easier to detect. 5. Default Deny – Denying access by default and granting permissions only when necessary minimizes exposure. 6. Adhere to the Principle of Least Privilege – Limiting user permissions reduces the risk of unauthorized access. 7. Sanitize Data Sent to Other Systems – Transmitting sanitized data prevents exploitation. 8. Practice Defense in Depth – Layered security measures ensure multiple safeguards if one layer fails. 9. Use Effective Quality Assurance Techniques – Regular testing, code reviews, and static analysis identify and address vulnerabilities early. 10. Adopt a Secure Coding Standard – Standardized security practices ensure consistency and mitigate common vulnerabilities.   These principles collectively ensure that security is integrated into our systems at every stage of development. |
| **5** | Now, let’s examine the 10 coding standards outlined in the Green Pace Security Policy. These standards are prioritized based on the threat matrix, which evaluates each by severity, likelihood, and remediation cost.  At the highest priority, we address critical issues that pose severe risks, such as STD-004-CPP (memory allocation errors) and STD-001-JAV (SQL injection). These require immediate attention to prevent significant vulnerabilities.  Next, we focus on standards that protect sensitive information and prevent common vulnerabilities. Examples include STD-008-CPP (hard-coding sensitive information) and STD-002-CPP (uninitialized memory). While these risks are less frequent than critical issues, addressing them is key to maintaining secure practices.  Following that, we have standards that support resource management and stability, such as STD-007-CPP (random number generation) and STD-001-CPP (out-of-range enumeration). These ensure the system remains reliable and secure over time.  At a lower priority, we address standards related to error handling and system stability, like STD-003-CPP (string-to-number conversion errors), STD-009-CPP (file handling), and STD-006-CPP (exception handling). These standards play a vital role in maintaining the system's integrity and preventing security risks from errors.  Finally, STD-005-CPP (static assertion) ranks lowest, as it addresses compile-time concerns with minimal impact on runtime security, but still contributes to overall code quality.  By following this prioritized approach, the Green Pace Security Policy ensures we tackle the most pressing vulnerabilities first while maintaining a strong and stable security framework. |
| **6** | On this slide, we summarize the three key encryption policies designed to protect sensitive data:  Encryption at Rest protects data stored on physical devices like hard drives, SSDs, and cloud storage. It ensures that data remains unreadable if unauthorized parties gain access to the storage, providing security against theft and breaches. This policy ensures compliance with data security regulations and safeguards sensitive information when it's not actively in use.  Encryption in Flight secures data as it moves across networks. By using protocols like HTTPS (SSL/TLS) and VPNs, it prevents interception or tampering during transmission, ensuring sensitive data such as login credentials and financial information remains protected from eavesdropping and man-in-the-middle attacks.  Encryption in Use protects data while it is being processed or actively used in applications, memory, or CPU operations. Techniques like homomorphic encryption and trusted execution environments ensure that data remains encrypted even during computations, mitigating risks from memory scraping or insider threats during active data use.  Together, these policies provide a comprehensive approach to ensuring that sensitive data is securely protected at all stages: when stored, during transmission, and while in use. |
| **7** | Next, we’ll explore the Triple-A Policies - Authentication, Authorization, and Accounting - which are essential for securing access to systems and ensuring accountability:  Authentication verifies the identity of users, devices, or systems attempting to access a resource. By using methods like passwords, biometrics, security tokens, or multi-factor authentication, we ensure that only authorized parties can access sensitive data or services. This step is critical for preventing unauthorized access.  Authorization determines what authenticated users are allowed to do within the system. Once users are authenticated, their roles or permissions are checked to ensure they can only access the data or perform the actions necessary for their tasks. This policy minimizes the risk of privilege escalation or unauthorized actions by limiting access rights.  Accounting involves tracking user activity within the system. This includes logging and monitoring actions such as file access, changes made to databases, or the creation of new user accounts. Accounting helps ensure compliance with security policies, provides visibility into potential security breaches, and supports audits by creating a traceable record of user actions. |
| **8** | The next four slides demonstrate the usage of unit tests which are essential to ensuring that a program is performing as expected. The first unit test verifies that using the resize function increases the collection. On the left is the code for the unit test and on the right is the result of the unit test. |
| **9** | This next unit test verifies that using the resize function is also able to decrease the collection. |
| **10** | In this unit test, it is verified that indexing beyond the bounds of the array throws an exception. |
| **11** | This final unit test verifies that using the push\_back function adds the correct value to the array. These examples help demonstrate proper usage of unit tests and should be followed in creating similar tests of program functionality. |
| **12** | Here, we see a depiction of the DevSecOps pipeline. |
| **13** | The DevSecOps pipeline integrates security into every stage of the software development lifecycle, ensuring a seamless collaboration between development, security, and operations teams (Microsoft, n.d.). In the pre-production phase, the "Assess and Plan" stage focuses on identifying potential threats, regulatory requirements, and prioritizing security tasks using tools like threat modeling and change impact analysis. During the "Design" phase, secure design principles, such as OWASP best practices and test-driven design, are applied with the help of static code analysis tools to validate security requirements early. In the "Build" phase, security tools like dependency checkers and secure open-source usage scanners ensure secure code integration, and compilers may be utilized to flag issues like buffer overflows or other vulnerabilities. The "Verify and Test" stage employs vulnerability scanning, functional testing, and compliance testing tools to identify and mitigate issues before deployment.  In the production phase, security automation continues through various stages. During "Transition and Health Check," tools for configuration validation, deployment testing, and penetration testing are implemented to ensure secure environments. The "Monitor and Detect" stage relies on automation tools like SIEM (Security Information and Event Management), intrusion detection systems, and event alerting to monitor live environments. The "Respond" phase uses automation to block attacks, roll back changes, and ensure systems recover from disruptions efficiently. Finally, in the "Maintain and Stabilize" phase, tools assess systems against the established security baseline and automate processes to restore stability post-incident. Security automation is an ongoing and essential component of the DevSecOps pipeline, ensuring strong defenses are maintained at every stage of the software development and deployment lifecycle. |
| **14** | Acting immediately on security issues can be effective in addressing vulnerabilities quickly, reducing the risk of breaches, and enhancing system integrity. However, immediate action can also result in rushed decisions, which may lead to resource strain or incomplete fixes. On the other hand, waiting to address security issues allows for careful planning and better resource allocation, but it increases exposure to potential attacks, which could lead to significant losses or compromised data.  Without proper planning, the strategy may lack comprehensive threat assessments or fail to prioritize the most critical vulnerabilities, leaving key areas exposed. Additionally, insufficient monitoring or reliance on outdated tools can weaken the strategy, leading to delayed detection of threats, ineffective remediation, or over-reliance on tools that may no longer be effective against evolving attack vectors.  To strengthen and improve our security approach, it is essential to prioritize vulnerabilities based on their severity and likelihood and implement continuous monitoring. This includes investing in up-to-date tools and providing ongoing training for our teams. Establishing a clear remediation timeline, conducting regular assessments, and fostering collaboration across development, security, and operations teams will ensure we maintain a balanced, proactive, and resilient security strategy. |
| **15** | Let’s go over a few key recommendations that can really help strengthen our security efforts:  First, we need to establish detailed incident response procedures. It’s essential to have clear, actionable steps in place for how we identify, contain, and recover from security incidents. Having a solid plan will help minimize the impact of any security breaches and allow us to respond much faster.  Next, we should implement regular security training and awareness programs for all employees. Security is everyone’s responsibility, and ongoing training will help make sure everyone is up to date on best practices, phishing awareness, and how important it is to follow security policies. This will help reduce human error, which is often a weak point in security.  Another important recommendation is to conduct routine risk assessments. Our security policy should be a living document that we update regularly to keep pace with new threats, technologies, and any changes within the organization. This ensures we’re always on top of what could potentially put us at risk.  We should also enhance our monitoring and auditing capabilities. Investing in automated tools like intrusion detection systems and conducting regular security audits will help us catch vulnerabilities and breaches earlier. The more visibility we have into our systems, the faster we can respond to any issues that arise.  Finally, we need to promote proactive vulnerability management. Regularly scanning for vulnerabilities and staying on top of patch management is key to addressing any weaknesses before they can be exploited. Being proactive helps us avoid costly breaches and ensures our systems are always secure.  By focusing on these areas, we’ll create a more resilient and responsive security environment. |
| **16** | Now, let’s wrap up with some key conclusions and suggested standards for strengthening our security approach:  First, we should adopt ISO 27001. This standard helps us establish a formal Information Security Management System, which is crucial for proactively addressing our security needs as they evolve. The benefit here is that it gives us a structured approach to managing risk and reducing vulnerabilities across the organization.  Next, we recommend integrating the NIST Cybersecurity Framework. This framework provides a five-step process - Identify, Protect, Detect, Respond, Recover - that helps us make better, risk-based decisions. By using this framework, we can improve our ability to prepare for and respond to potential threats, boosting our overall readiness and response capabilities.  Another best practice is to implement OWASP Best Practices. These are guidelines for secure software development, helping us avoid common vulnerabilities like injection flaws and cross-site scripting. By following these practices, we can better protect ourselves from widespread web-based attacks that are all too common today.  We should also consider Zero Trust Architecture, which is all about continuous verification for every access request. With Zero Trust, we’re not just assuming trust based on network location or previous access - we verify every request, every time, ensuring tighter security.  Lastly, we need to establish Cloud Security Standards. With so many services and data moving to the cloud, having clear guidelines for securing cloud-based services is essential. A real-world example of why this is so critical is the Equifax breach in 2017, which could have been prevented with the right Zero Trust and cloud security protocols in place.  By implementing these standards, we can build a stronger, more proactive security framework that better protects the organization and its assets. |
| **17** | This slide lists the resources referenced in creating this presentation. Thank you for your time and attention today. I hope the information shared has been valuable and will contribute to your understanding of the Green Pace Security Policy. |