**Project Two: Security Policy Presentation**

Winnie Kwong

Southern New Hampshire University

CS-405: Secure Coding

Professor Tam

August 18th, 2024

<https://youtu.be/F_qNLixJnLc>

# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Did you know according to Security Magazine, a recent study in 2023 discovered that at least 70% of applications fail to meet basic security standards? This means roughly 3.5 vulnerabilities per application are affected by some security weakness. (Security Staff, 2023) We need better security coding practices to address these exposures and ensure that applications are functional and secure. Hi everyone, I am Winnie Kwong, a senior at Southern New Hampshire University. I am here on behalf of the Green Pace Organization to demonstrate what components make up a good security policy. Let’s get started. |
| **2** | Through the influence of defense in depth, the security policy uses specific standards and best practices to maintain safety and security. With the image provided, defense in depth illustrates potential areas and layers needed to be secure. Even when the threat landscape evolves, defense in depth is essential because its multi-layered protection mitigates risks, enhances security, and increases complexity for attackers. Using its comprehensive coverage of security controls, we can increase the range of protection covering anywhere from physical access controls to advanced threat protections. Although the image is just a baseline for assessments and improvements, it emphasizes the practice of continuous improvement to stay ahead of potential exploits. |
| **3** | Threats matrix. The threat matrix is used to analyze and manage a vulnerability's severity. A threat matrix can offer many benefits to improving security in software development and system design, such as improving resource allocation, proactive security management, and enhancing security planning. Starting at the top left, we have Likely risks. These are risks that are expected to occur with medium to low severity. Moving to the right, in the Priority area, we find risks of high severity that are anticipated to occur. At the bottom left, Low Priority, are risks with medium to low severity that are unlikely to happen. Finally, in the Unlikely quadrant, we encounter risks that, despite their high severity, are not expected to occur. Automated tools such as CPPcheck and SonarQube can assist with identifying risks and properly addressing the issues. |
| **4** | Here are the ten core security principles, and next to them are the corresponding coding standards where the principles apply.  Number one. Validate input data checks the source's data accuracy and quality before processing it into the application's function. This prevents malicious attacks and confirms the data entered is the same pattern as the expected formation.  Number two. Heed compiler warnings generate warnings when the code acts differently than it intended, meaning there's some security issue in the code. This principle is used to advise early bug detection and security flaws to mitigate attacks.  Number Three. Architect and design for security policies help to ensure the design aligns with security requirements.  Number Four. Keep it simple emphasizes codes must be more efficient and user-friendly to eliminate redundant steps for maintainability and clarity for all developers.  Number five. Default deny assumes all access or actions are denied unless they are granted permission. The smaller the circle of granted authorization, the smaller the risk of vulnerabilities to occur.  Number six. Adhere to the principle of least privilege is the minimum access rights to perform a given task. This principle helps to contain and minimize attacks on the surface.  Number seven. Sanitize data sent to other systems means to clean and validate data before transferring it to another system. If the data isn't clean, it can corrupt other systems and spread like wildfire.  Number eight. Practice defense in depth means to create multiple layers to protect the system, data, and users. Referring back to the second slide, defense in depth enhances security and reduces the risk of a successful attack.  Number nine. Use effective quality assurance techniques to maintain the quality of a product or service throughout the development lifecycle. Ways to approach this include continuous testing, unit testing, and test automation.  Lastly, number ten. Adopt a secure coding standard means to promote best practices to reduce the risk of vulnerabilities. In doing so, it displays security awareness and comprehensiveness to industry standards. |
| **5** | There are ten coding standards that are labeled and organized based on the threat matrix. Detecting and handling memory allocation errors is at the top priority because if the system does not have enough resources to fulfill the request, it can lead to unexpected behaviors. Systems should never be abruptly terminated due to memory allocation failures, or else it can cause immediate system impact, data loss and corruption, error handling and recovery challenges, and security implications. |
| **6** | Encryption policies. Encryption plays a crucial role in secure coding, and there are three main types of encryptions: at rest, in flight, and in use.  When encryption is at rest, data is stored on a device, regardless of if its lost or stolen, it can prevent unauthorized users from reading data. For example, hackers accessing customer details and banking records could have severe consequences. Encryption prevents these situations by making the data unreadable without the decryption key.  Next, encryption in flight secures data that is being transmitted from one network to another. The importance of the policy is to protect sensitive information without being intercepted. Pretend that a hospital sends client information to another facility. That data travels through a network, and once the data reaches the facility, only authorized staff can use a decryption key to review it.  Lastly, encryption in use protects data while a device is actively running. This is the highest level of security to ensure that sensitive information is safe from internal threats, even if the device is compromised. For example, government and military officials frequently work with classified information. Encryption in use ensures that this data stays confidential during processing and effectively mitigates the risk of exposure. |
| **7** | Triple-A policies are the framework for authentication, authorization, and accounting. Each component protects unauthorized users from accessing and misusing the software systems. Authentication verifies user identity, authorization determines what users can do, and accounting tracks user activity and resources.  Authentication applies to user logins, database changes, and new users through various methods of passwords, security tokens, biometrics, and multi-factors. For example, students are verified through the university portal through their username and password to ensure they are the right person.  Next, authorization verifies access levels, file access permissions, and database changes through ACLs, RBAC, and PBAC. Carrying on with the last example, students can view grades while the teacher can edit the grade, and admins have both access and more.  Lastly, the accounting policy applies to user logins, database changes, new users, and file access through recorded logs and KPIS. An example is viewing a log of when a user is accessing a file. |
| **8** | The following slides cover examples of mixed tests for positive and negative unit testing results. The first positive case is to add five values to the collection. The test involves asserting that the collection is empty with a size of zero, then calling the function to add five entries, and lastly, testing the size of the collection is accurate to see if the behavior is true. Knowing the exact amount of data is fundamental for effective security risk management toward stringent protection. |
| **9** | The following positive test is to verify assigning a specific number of elements or values to the collection. The test starts by asserting the collection is empty. Once it is confirmed the collection is empty, the system will add five entries and call the assign method to replace the content with three elements. Once completed, the Google test will confirm that the assigned method functions appropriately. This test's importance helps identify potential issues with the assign method and prevent bugs that suggest incorrect data manipulation or unexpected behavior. |
| **10** | The first negative testing inspection of the collection can be resized and decreased. The first step is to assert if the collection is empty with a size of zero, then resize and confirm the collection is at five. The collection will then again decrease to three and verify that the test is growing smaller. Creating a function designated to resize by reducing its capacity benefits code reusability, encapsulation, and flexibility. |
| **11** | The last negative testing slide ensures the function, pop-back, can decrease the collection size by removing the last element in the collection. The code starts with adding five elements to the collection and quickly calls the pop-back function and tests to confirm the collection is now at four. The benefit of using the function is for memory management and readability, making it a valuable tool for developers. |
| **12** | The illustration demonstrates the DevSecOps model and its approach to integrating security practices throughout the software development lifecycle. Incorporating DevSecOps with the policies and standards throughout the development process can build a more resilient security practice. Key benefits of the model include improved security, faster remediation of vulnerabilities, and reduced security breaches. |
| **13** | The DevSecOps pipeline is divided into two sections: the pre-production and production phases. The pre-production phase lays the foundation for secure development and deployment. It consists of four stages: assess and plan, design, build, and verify and test. The production phase is when the application is deployed and used to monitor and respond continuously. This phase also consists of four stages: transition and health check, monitor and detect, respond, and maintain and stabilize.  Plenty of tools can be used for automation in the DevSecOps lifecycle. To name a few tools that can be integrated include GitLab, SonarQube, and Splunk. GitLab is a tool used throughout the DevSecOps lifecycle, including CI/CD pipelines, security scanning, and code quality management. SonarQube can be used as a static application security testing in the build stage to offer code quality and security analysis. An example of a tool in the production phase is Splunk. This automation tool can search, analyze, and visualize data to extract valuable insight. |
| **14** | The example case of tesla described by Pryimenko (2023), highlights the risks and benefits of handling security weaknesses. Telsa forgot to revoke the access rights of two former employees, resulting in a massive exposure of leaked data. Although the company acted by conducting user access and monitoring user activity, they waited until it was too late, risking data breaches, financial loss, and insider threats. If the company had acted sooner, it could have mitigated the damages that occurred and preserved its reputation. The incident underscores the importance of insider threat management. However, one risk that can be associated is over-reliance on external security controls, where sensitive information can fall into the wrong hands and cause reputable damage. To prevent similar incidents, companies should implement training employees about data security, stricter access controls, categorizing data to protection levels, developing a response plan for insider threat, and utilizing tools to monitor abnormal behavior. By adopting these strategies, organizations can enhance their security and mitigate risks of similar scenarios. |
| **15** | Recommendations. Testing software throughout SDLC can identify gaps in the security policy. Outdated information can occur when policies do not reflect current threats, technology, or regulatory changes. Data becoming obsolete can increase vulnerabilities to threats, operational errors, and ineffective problem-solving. Lack of security awareness can occur when employees need help understanding the policy or its importance. When ignored, it can lead to an increased risk of security breaches and compromised data protection, which causes organizations to have system downtime and loss of productivity.  Luckily, there are plenty of methods to address these issues. The first is ensuring developers have proper training. Developers can increase their skills, improve the identification of vulnerabilities and code quality, and reduce risk mitigation. All developers should continuously review and update. With the increase in developer knowledge, developers can reduce their rework in the development lifecycle while ensuring the code aligns with the latest security standards and best practices. It showcases proactive identification and rapid response when continuously testing for exploiting vulnerabilities. Continuous testing helps protect valuable data from unauthorized users. Utilizing automation tools can assist with making codes become more robust and resilient. Lastly, implementing test incident response plans helps outline procedures for handling security incidents during the testing phase. Benefits include enhanced coordination, faster response times for quicker resolutions, and reduced impact from the incident. |
| **16** | Conclusion. This policy outlines the principles of secure coding within our organization. It's more than writing code, it's writing a secure code. Modeling best practices and secure coding shouldn't be a chore, but a responsibility for software developers. I strongly suggest adopting defense in depth, encryption, triple-A, and automation to develop a more robust foundation. Fortifying systems and safeguarding data showcase the efficiency and effectiveness of a compelling application. I also want to emphasize security should never be an afterthought. It's a fundamental part of the development lifecycle.  Remember, secure coding is not just a technical requirement; it's a strategic priority. By embracing secure coding practices and staying ahead of the curve, we can build a more resilient culture for the next generation of security. Thank you for your time. |