Project Two: Security Policy Presentation Script

Charlotte Armistead

Southern New Hampshire University

CS 405: Secure Coding

April 21th, 2024

**Video Link:** [**https://youtu.be/P33ItNIbvKU**](https://youtu.be/P33ItNIbvKU)

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Welcome, Green Pace, to this Security Policy Presentation. My name is Charlotte Armistead. Let’s get started. |
| **2** | This security policy for Green Pace will ensure compliance with security principles in software development. It establishes core security principles, coding standards, authentication, authorization, accounting standards, and data encryption protocols. This will also support defense-in-depth best practices by implementing various security measures.  Here, you can see a diagram demonstrating defense in depth. It depicts the various layers of security measures that should be implemented. Furthermore, it shows multiple resources, tools, and strategies to help with security. |
| **3** | This threat matrix table shows where each coding standard is placed in terms of priority and likeliness.  As you can see, standards 3 and 5, are likely to occur and are a high priority.  Whereas standards 1, 6, 8, and 9 ( are unlikely and low priority.  Standards 4 and 10 are likely but only of medium priority.  Standard 7 may happen, but is of low priority.  Finally, standard 2 may happen and is of medium priority. |
| **4** | Here are the ten security principles  First, validate the input data, which is to say that any input a program or system receives should be validated. This means that the integrity and correctness of the data should be checked. This will ensure that only expected and safe inputs are accepted, preventing attacks such as injections.  Second, heed compiler warnings. Compiler warnings can notify the developer of errors in their code and provide insight into what is wrong with the section identified. This can help reduce the number of vulnerabilities and bugs in code early on.  Third is architect and design for security policies. Architecting and designing software with security policies in mind can help ensure that robust security measures are implemented.  Fourth we have keep it simple. Keeping code simple can improve security, as complex code can introduce vulnerabilities or errors that can be overlooked. Additionally, simple code can enhance its readability and maintainability.  Next up is default deny. Access permissions should default to deny rather than only deny in certain situations. This can minimize the attack surface and ensures only those with explicit permission can access certain things.  Number six is adhere to the principle of least privilege. Which says that users or processes should only have the minimum permissions to accomplish their tasks. By adhering to this, developers can ensure that users can not access more than they need to, and processes can not be manipulated to access things that are not supposed to.  Seventh, sanitize data sent to other systems. Sanitizing the data sent to other systems by cleaning and validating the data helps protect other system components.  Number eight is practice defense in depth. Defense in Depth is a security approach that utilizes multiple layers of security measures. This helps to protect the system because if one layer fails, then there are other layers that can stop the attack.  Ninth is use effective quality assurance techniques. Effective quality assurance techniques can help ensure the program is secure by identifying potential vulnerabilities to be fixed. Additionally, this helps eliminate bugs or other issues with the software.  Finally, adopt a secure coding standard. Using a secure coding standard can give developers a guideline for coding best practices. Adhering to one can remove common security risks.  You can also see on the right which coding standards are associated with which principles. |
| **5** | Once again we have the 10 coding standards that were mapped on the threats matrix. Here they are ranked from most to least important, though they are all important. You may notice that there are some similarities with the grouping in the standards. This is because I ranked them based on their likeliness and priority levels.  Ranked first as most important is: detect and handle memory allocation errors. Depending on the method used to allocate memory, a developer may not receive an exception for a failed allocation. Instead, a nullptr may be returned. Attempting to access the resulting pointer can result in unexpected behavior and errors.  In second place we have: do not attempt to create a string from a null pointer. Doing so can cause undefined behavior.  Third is: Do not alternate input and output from a file stream without an intervening positioning call. This can lead to data being handled incorrectly. Which can result in vulnerabilities being introduced and leaving the system susceptible to SQL injection attacks.  Next is: guarantee that container indices and iterators are within valid range. Trying to access an out of range index will throw an exception, which if not handled can cause an abnormal termination.  Fifth is: do not rely on the value of a moved-from object. Some types do not fully support move operations. This can leave an object in a valid but unspecified state.  The sixth is handle all exceptions. An unhandled exception will call the terminate function, which will abnormally terminate the process. An abnormal process termination is the typical vector for denial-of-service attacks. Thus, all exceptions thrown must be caught by a matching exception handler. Or, at the very least, the stack should be unwound before terminating the process.  Seventh is close files when they are no longer needed. Files that remain open are vulnerable to data corruption. Additionally, closing them will free up resources which can improve system performance.  In eighth we have: do not return from a function declared no-return. Doing so will cause unexpected behavior.  Second to last is never qualify a reference type with const or volatile. As, attempting to cv-qualify a reference type will result in undefined behavior.  Last but not least is: do not treat a predefined identifier as an object if it might only be implemented as a macro. In C++ macros can redefine identities. This can lead to unexpected behaviors. |
| **6** | Here we have our encryption policies  Encryption at rest applies to data that is not being used or transferred. This works to prevent unauthorized access to the data even if the device it is stored on is compromised.  Encryption in flight refers to data that is being transferred over a network. This ensures that the data is safe even if the transfer is intercepted.  Encryption in use refers to data that is being processed or used by applications or processes. This works to ensure that the data is safe from data manipulation and unauthorized access. |
| **7** | Triple-A Polcies, that is authentication, authorization, and accounting  Authentication verifies that the user is who they say they are. It is typically used for user logins, database changes, and the addition of new users.  Authorization checks the level of permissions a user has. It can be used to check if a user has permission to make changes to a database or access certain files. Additionally, depending on the system, only users of a certain level can register new users.  Accounting is recording the various actions taken in relation to the system. This can be used to monitor who accesses what files and who makes what changes to which databases. |
| **8** | Unit tests are useful because they allow developers to verify the correctness of individual components of software, ensuring functionality, identifying bugs early, and facilitating easier debugging and maintenance.  We will take a closer look at four unit tests. These are: Verify Length Error Thrown, Verify Out Of Range Thrown, Verify Clear Erases Collection, and Verify Resize On Decrease. |
| **9** | Verify Length Error Thrown. This test checks that a length error exception is thrown when attempting to resize the collection below 0. This test is also a negative test as it is checking that an exception is thrown rather than verifying the result is as it should be. |
| **10** | Verify Out Of Range Thrown. This is another negative test. It checks that an out of range exception is correctly thrown when trying to access an index that is out of bounds. |
| **11** | Verify Clear Erases Collection. This is a positive test. It checks that the collection is empty when the clear function is used. |
| **12** | Verify Resize On Decrease. This test checks that the collection’s size is smaller when using the resize function to make it smaller. |
| **13** | Here is a DevSecOps diagram. It is broken into two overarching stages: pre-production and production.  In pre-production we will assess threats and plan how to defend against them, design security measures, build and develop the code, and finally, verify and test the new features with vulnerability scans and functional, compliancy, and security tests.  In production, we begin by transitioning and preforming a health check. This is where we would configure security settings and preform a penetration test. Next, we monitor the system and detect any attacks. When an attack happens we respond. This can include turning off services and preforming a roll back on the system. Finally, we stabize and maintain the system, and return to pre-production to address new threats. |
| **14** | Here are eleven external tools that can help check for vulnerabilities. This would be used regularly throughout development and before deployment to ensure any vulnerabilities introduced are caught and handled appropriately.  Some of these tools do need specific flags or options enabled to detect certain violations. Without these enabled the tools may overlook some vulnerabilities. |
| **15** | Currently, we face code vulnerabilities and lack standardized security policies. This can lead to unauthorized access or modification of data.  Acting now has the risk of being potentially costly. Additionally, there may be some resistance from employees. However, it also has the benefit of improving our system’s security and saves us resources in the long run.  On the other hand, waiting can potentially save us resources in the short term. Furthermore, it will give us time to train employees on the new policies. However, this leaves us open to more attacks for longer. Without the new policies in place, we could suffer more severe data breachs. Additionally, any attacks would result in a loss of trust and reputation. |
| **16** | There are other potential gaps in the security policy, such as social engineering and the lack of an incident response plan.  Social engineering is where a victim is manipulated, influenced, or deceived in order to gain control over a computer system (CMU, n.d.). To counter-act this we should provide training on how this may occur and how to avoid falling victim to it.  An incident response plan will outline what to do when an incident occurs. This can minimize system downtime, data loss, and reputational damages. |
| **17** | In conclusion, to prevent future problems, we should begin implementing industry best practices and adopt the policies outline in this presentation and the security policy document. These include:  Implmenting defense in depth, which is the layering of security measures. This can help prevent attacks by providing more layers an attacker needs to get through.  Implementing a zero-trust policy with strict identity verification, which can prevent uninteded access to the system. This also means that any user or device trying to access the systems will need their identity and permissions verified before that can access the system, even if they have previously been verified.  Training employees on how to respond to attempted attacks, such as those from social engineering, or actual attacks, such as data breaches, can help prevent or minimize the losses of such attacks.  Creating an incident reponse plan is vital for those same reasons. |