Slide 1:

The following presentation is a Security Policy Presentation, compiled and narrated by Caroline Wilson

Slide 2:

Defense in depth is the practice and study of compiling security such that it acts as layers of protection through a fort. If the first layer of protection is breached, here is another line of defense waiting in wake for the next strike. This is the same concept as Defense in Depth. If an attacker manages to make it through the first line of defense, they will have more hurdles to jump over before they reach their payload. It is important however, to remember not to take it too far. While you want your code to be as secure as possible, you still want it to be simple, modular, and readable, as always. Don’t weigh down and convolute good code with unnecessary bloat.

Slide 3:

When looking at the threats matrix for the potential errors, it is important to understand how they fall on a list of priorities. A risk with a high severity and high probability of occurrence needs to be taken care of immediately and would be grouped into priority. A risk with a lessened probability, but with a higher severity would be ranked in with the likely, and so on down to unlikely where the risk poses the least threat. I have grouped the risk assessments of my coding standards into these categories.

Slide 4:

The 10 Core Security Principals set a frame work for best practices in the successful creation of secure code. In taking these forward into coding standards, it is possible to see, as in the case of FIO30-C, “Exclude user input from format strings” in an attempt to lessen SQL injection, there can be more than one principal in play. In this case defense in depth, validate input, and secure coding standards all can be seen at work.

Slide 5:

The ten coding standards studied in this report have been ranked by priority and Severity. The cases with the highest risk are Guarantee that storage for the strings has sufficient space for the character data and the null terminator, because this can lead to buffer overflow and the great potential for manipulated data and FIO30-C which says to exclude user input from strings. This is because excepting an entire line of user input could all for malicious code to be input and allows for easy exploitation through this vulnerability.

Examples of a medium high risk are MSC41-C, never hard code sensitive information, and ENV33-C, do not call system(). These both can allow for open vulnerabilities. In calling system(), it would give the attacker an immediate line to the directory. In including sensitive information in the code, it could give them the authenticating data available to them to access data.

Slide 6:

Encryption at rest is the practice of encrypting the physical media, as in a hard drive on a computer. By requiring authentication on the physical device itself, it adds an additional level of security that would disallow malicious activity against the devices within a network. Instead of having an “open door” for the attacker, even if the device were to fall into their hands, and device. additional guard would be in place that would hinder their ability to use the information on the

Encryption at flight is the process that considers encryption while information is being transmitted. This may include hashing, or encrypting, data as it is passed through a network. By implementing this theory, information that is being passed would be useless without the proper decryption information. This protects the data from attack while it is being transmitted.

Encryption in use is the act of needing to encrypt or decrypt data. This can be rationalized as a user needing to enter their password in order to verify their identity and authenticate the user. By implementing this practice, unauthorized users or attackers will be blocked access to the data.

Slide 7:

Authentication is a policy in which a user needs to prove who they are and their privileges before being granted access to data. Within this, the user needs to provide their personal username and password before being allowed to access data. This prevents unauthorized users from being able to access data or a network that they should not be allowed to.

Once a user is authenticated, authorization must be granted. Authorization is the level that dictates what information may be accessed and when by an authenticated user. The principle of least privilege could come into play in this policy. A user should only be allowed to access the information that is required for their position and should not be allowed to access more secure files.

Accounting is the policy that tracks the overall usage of a user on a network. This could include the system resources that are being allocated or the levels of access that are being used. With this, we can assure that network assets are not being abused, and can help to recognize when there is suspicious activity in play.

Slide 8:

Unit testing is the ability to test blocks of code for functionality based on parameters. As pictured in this slide, this block uses assert to ensure that the collection in question is not greater than or equal to the set max size. This could cause errors in running the code should the sizes not meet requirements.

Slide 9:

This block of code uses assert to ensure that the size of the collection can be changed should it need to be.

Slide 10:

Likewise, it is checked to ensure that the size can be decreased, thus saving resources.

Sile 11:

This example of a negative test ensures that certain logic errors would not occur. The format of this test allows for code modularity in that many forms of logic can be checked.

Slide 12:

This is a positive test to ensure that mathematical functionality is working and the correct coding standards are in place for this to work.

Slide 13:

Automation in secure coding the a best practice when coding securely. With the life cycle of the program being the core of automation, this process ensures that creative processes, testing and maintenance are completed through the whole process. It is important to run scans early and throughout the entire development process to ensure that there are no major bugs or vulnerabilities in code as the project progressed. This visual diagram provides a walkthrough of the best practices in creating secure code. From the planning stages to production stages, testing and detection provide a secure knowledge of possible bugs and vulnerabilities, while using a roadmap to secure coding.

Slide 14:

The DevSecOps pipeline is a practice that adds to the standard DevOps practice by addressing security early and often through the process.

There are a number of external resources that can be used to check code for security and functionality. While some may be embedded in whatever IDE the developer may be using, external resources allow for deeper and more comprehensive analysis. Klocwork, LDRA toot suite, Parasoft, RuleChecker, SonarQube, CodeSonar and Clang are just a few of the options available, each checking for lapses in security and errors in code.

Slide 15:

In moving forward with the security policies, it is important that the gaps in the latter policy are immediately acted upon. The longer that a vulnerability is open, the more the chance that there be an attack. As this presentation has shown, there could be a problem lurking around any corner. From something so simple as the wrong syntax, or an invalid entry, to severe issues such as proprietary information being embedded in an available block of code, there are a number of issued that could arise.

Slide 16:

The most important thing going forward to use best practices, but for projects that have already been completed, automation should be utilized immediately to ensure that nothing was missed in the initial development process. Once vulnerabilities are brought to the forefront of the process, it is important to create a risk assessment and tackle the issues in that order. In order to ensure ongoing success, it is important that everyone on the development team has training in the security policies, has a firm understanding as to how to use the automation tools and when, and knows to follow the ten core security principals.

Slide 17:

In summation, if the policies here in are followed, and the teams in question are trained on them, the probability of the production of secure, well written, functional code will sky rocket. Practicing the ten core security principals and baring in mind the constant need for defense in depth will allow for a better understanding on how to product quality work. Paying attention to the DevSecOps process will create a road map for how to get from the planning stages to finished work with the least potential for vulnerabilities within the system. It all comes down to details. It is important to understand the data and functionality, but it is also important to know your role as a developer and what you can do to make that code and functionality secure for the user. Their data security, and your success are at your fingertips!