# CS 405 Project Two Script Template

Treyton Davis

6/23/2024

Link:

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Welcome and good morning everyone. I am here to present our new security policy for Green Pace. |
| **2** | Defense In Depth talks about having a multi-layered defense system in place to reduce information being stolen or leaked by bad actor(s). We need to ensure that we are producing secure software to help protect ourselves and our customers. |
| **3** | In this threat matrix, I listed the 4 threats to security that matches likelihood and priority on each side. I listed DCL55-CPP (Avoid Information Leakage When Passing A Class Object Across A Trust Boundary) in the unlikely threat since most of our code will be within bounds of trust. I listed ERR57-CPP (Do Not Leak Resources When Handling Exceptions) as Low Priority since we can ensure exceptions in the code will only show what we would need to know. I listed MEM50-CPP (Do Not Access Freed Memory) in Priority because if we try to access memory that has been freed by the program, then it can crash the program. Finally, I listed EXP45-C (Do Not Perform Assignments in Selection Statements) in the likely threat because it can be a simple error to happen to anyone, yet it still happens all the time due to typing quick. |
| **4** | The 10 principles I would like us to implement, and follow are on the screen. We would need to ensure that all data being input by users is validated before the program accepts it to ensure accuracy and allows sanitizing in case of SQL injection. Compiler warnings do not affect or prevent execution of the programs, but these warnings signal an issue within the code that could reveal a vulnerability. Account security policies should be developed from the beginning and ensures adherence to the policies across the entire program. KISS (Keep It Secure and Simple) means to prioritize straightforward code before solutions that are complex. The higher the complexity, the harder it will be to fix and higher the likelihood of errors and vulnerabilities. Default Deny means to only grant access upon request and helps keep problems or security breaches to a specific area. Least Privilege means that the lowest level of access required to work is what should be granted. Limits potential problems to a restricted domain. Verify validity and format of data of receiving system before allowing to send. Offers advantage of mitigating risks and issues of receiving system not accepting information. Defense In Depth involves multiple layers of protection. Ensures redundancy in case a layer fails; the system is still protected. Testing and effective QA plays a part ensuring any issues are found and corrected before exploitation begins. Creates a more secure system. Secure Coding Standards are essential for consistent delivery of secure software. |
| **5** | The first standard I identified we would need to use is Avoid Information Leakage When Passing A Class Object Over A Trust Boundary. When we need to pass a pointer to a different trusted domain, the programmer must ensure that the padding bits do not contain sensitive information. This ensures all data stays within the boundaries of what can be trusted. In the non-compliant example, the code copies data from the kernel space to the user space. The compliant example instead serializes the structure data before copying it to an untrusted space. This standard maps to Validate Input Data since data that needs to be passed correctly can be contained in a class object. |
| **6** | The second standard I identified we would need to use is Ensure Division and Remainder Operations Do Not Result in Divide-By-Zero errors. If a user is asked what values they would like to divide by, then we need to have a check in place to ensure if the denominator is 0, then the system tells the user that it cannot be performed and asks for different values. In the non-compliant example, there is no check for 0 showing up. The compliant example has this fixed and ensure there would not be a divide-by-zero error. This maps to Validate Input Data since there should be a check to ensure the system doesn’t crash or leak data for anyone to access. |
| **7** | The third standard I identified we would need to use is Do Not Attempt To Modify String Literals. We shouldn’t be assigning strings to pointers to non-const or casting a string literal to a pointer to a non-const. In the non-compliant example, the variable str is assigned as a pointer. This can cause issues when trying to modify because the program won’t know where it begins. The compliant example assigns the string to an array that ensures each letter is in its own space to be reassigned a character. This standard maps to Validate Input Data because the string being assigned or casted to a non-const can cause many issues and could even cause the string to not be changed when it needs to. |
| **8** | The fourth standard I identified we would need to use is Normalize Strings Before Validating Them. If the program accepts untrusted inputs, then we should normalize before validating. Normalization is important because Unicode can have the same string but many different representations. In the non-compliant example, the data is normalized after validating. In the compliant example, the data is normalized then validated to ensure there is no injection happening from the user input. This standard maps to Validate Input Data because of Unicode having many representations for a single character. |
| **9** | The fifth standard I identified we would need to use is Do Not Access Freed Memory. It is not our choice when the memory is reassigned or reallocated. That is up to the memory manager. If the memory is freed, then all pointers are invalid that point to that location. In the non-compliant code, the memory is freed before the pointer can move the data from itself to another location. That is fixed in the compliant example where the pointer is assigned then deleted. This standard maps to Practice Defense In Depth since no one should be trying to actively access freed memory and the memory should not have any data since it was returned to the memory manager and the operating system. |
| **10** | The sixth standard I identified we would need to use is Do Not Perform Assignments in Selection Statements. This can cause unexpected behavior from the program because it would be assigning variables different values than we would like and can cause the program to always enter the else statement instead of the if statement. In the non-compliant example, a is being assigned the value of b, which would cause the statement to be false. The compliant example shows that it is seeing if a is equal to b, if so then enter the if statements code block. This standard maps to Keep It Simple because it is a simple error that can be fixed easily, unless it is in a multi-layered if statement which can be difficult to find without a compiler saying where the error is at. |
| **11** | The seventh standard I identified we would need to use is Do Not Leak Resources When Handling Exceptions. We shouldn’t leak resources when throwing an exception, even when we are construction an object to be used by the system. In the non-compliant code, if there is an exception the system doesn’t delete the variable and can leak resources if not careful. That is why in the compliant example, inside the catch statement the program will delete the variable to ensure it doesn’t get leaked from the exception occurring. This standard maps to Defense In Depth since exceptions can be used to force access to the data storage of the program and ensures data is protected if correctly caught by the program. |
| **12** | The eighth standard I identified we would need to use is Do Not Destroy A Mutex While It Is Locked. Mutex objects are used to protect data from being concurrently accessed. If destroyed while blocked, then shared data is no longer protected. In the non-compliant example, the mutex is initialized inside start\_threads(), which can cause the mutex to be deleted prematurely. In the compliant example, the mutex is initialized outside of any methods, so we can ensure it doesn’t get destroyed early. This standard maps to Default Deny because is the system is locking data, when it should be the only one to destroy the data being used. |
| **13** | The ninth standard I identified we would need to use is Use Valid Iterator Ranges. Using valid ranges ensures the program knows where to start and where to end. In the non-compliant example, we see an error in the iterator. The error is that the end() and begin() are swapped, which means the iterator will start at the end and then throw an out-of-bounds error. In the compliant example, the begin() and end() are in the correct spots, which ensures the iterator knows how far to go through the memory to find the values needed. This standard maps to Validate Input Data because if we mess up the start and end of the range, it can cause an error in the program. |
| **14** | The tenth standard I identified we would need to use is Close Files When They Are No Longer Needed. If we don’t close the files before ending the program or when the life of the pointer ends, then we risk data loss or corruption. In the non-compliant example, we can see there is no call to close the file, while in the compliant example there is a call to close and a check to ensure the file did close. If the file does fail to close, then we can throw an exception to ensure the file doesn’t corrupt. This standard maps to Default Deny because we should be closing files as soon as we are done using them to ensure no corruption of the files. |
| **15** | Encryption policies are important because if we don’t have data being encrypted at each level, then we risk data leaks happening. Encryption At Risk involves data that is being stored on a disk that has a decryption key stored elsewhere to ensure no one can decrypt the data easily. Encryption In Flight is used to protect data that is moving across a network. While our network should not be accessible from outside the company, when we move the data from our servers to the customer’s servers then we need to encrypt it to ensure no one outside the two companies are able to access it. Encryption In Use means to encrypt data that is being accessed, updated, read, or input into a program or system. |
| **16** | Triple-A (Authorization, Authentication, and Accounting) Policies are designed to ensure proper file access across the company and ensures our customers protection by knowing when someone is accessing certain things. Authentication needs to be used to ensure each person accessing the system is accessing information at their level. Authorization needs to be used to verify the user and allows for the system to have the ability to log who accessed what and when. Accounting uses that information to correctly log who accessed that information. |
| **17** | The main difference between the current DevOps process and the new DevSecOps process will be adding security testing tools to the current set of CI/CD tools being used by the team to ensure security is being involved from the start. While transitioning, Green Pace needs to choose the right tools that are relevant to the code to satisfy use cases now and in the future, involve the security team, prioritize quality over speed to ensure everything is secured, and ensuring code is always monitored. |
| **18** | CodeSonar and Axivion Bauhaus Suite were the two tools I found that would help us adopt a DevSecOps approach to development. Code Sonar protected eight different standards while Axivion protected against 5, some did overlap while some were only protected by one. |
| **19** | There are always risks to adopting new policies that could cause things to not work the way we would like. If we enact and start now, we can ensure no data is stolen from ourselves or our customers and ensure they have full confidence in using the software we have developed for them. |
| **20** | When we look at revising the policies next year, we should ensure that we determine how to better address many of the principles not covered by the coding standards presented here. |
| **21** | In conclusion, I believe we should adopt all encryption and Triple-A policies to ensure we stay ahead of any future attempts of our systems being hacked while also helping our customers to know we are taking these policies seriously in the off chance we do get hacked. |