# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello everyone, and welcome to the green pace security policy presentation. My name is Derek Clark and today I will be directing you all through this comprehensive security policy guide providing implementation guidelines and recommendations for maintaining in the future. |
| **2** | This policy defines security policies, standard coding practices for secure coding, best encryption practices, triple A policies for Authentication, Authorization, and accounting, Aswell as how we plan to integrate the use of automated tools to ensure compliance with the policies and standards outlined in this guide. This policy is necessary to ensure compliance with industry standards and regulations for safeguarding data and support the defense in depth strategy of incorporating multiple overlapping layers of defense to combat various security threats made to our organization, systems, and code. |
| **3** | The current slide illustrates a threat matrix which organizes each of the coding standards incorporated into this policy into 4 distinct groupings, Likely, priority, low priority, and finally unlikely. In the likely category we have STD-001-CPP which is the data type coding standard, STD-002-CPP the data value coding standard, STD-003-CPP the string correctness coding standard, and finally STD-004-CPP the SQL Injection coding standard. This category illustrates standards that must be enforced to counteract vulnerabilities that are likely to appear in the code base. Moving on we have our high priority standards which we will see multiple standards from the likely tab are here as likely vulnerabilities may introduce critical vulnerabilities that will need high amounts of attention in resolving. In this category we have STD-003-CPP string correctness, STD-004-CPP sql injections, STD-005-CPP memory protection, STD-007-CPP exceptions, STD-008-CPP memory safety, and finally STD-010-CPP memory management. Next we have our low priority standards STD-002-CPP data value and STD-006-CPP assertions which still enforce useful standards but are considered low priority as the risks introduced in relation to them do not involve critical issues that need the highest attention. Finally, the last category unlikely which consists of STD-006-CPP assertions, accurately represents that issues involving assertions are unlikely to introduce new vulnerabilities into the code base. |
| **4** | Moving on to the next slide we have defined our 10 security principles intended to be followed to ensure we are proactively mitigated threats and following best practices when it comes to secure coding. First up we have validating input data which is crucial for ensuring that all data received is safe and matches system expectations helping to protect the integrity of the system and the data held within it. Next, we have heed compiler warnings which is crucial for identifying and resolving vulnerabilities present within the code such as but not limited to memory leaks, optimization issues, and unhandled exceptions. Compiler warnings should never be ignored unless the developer can confidently say that it does not introduce immediate additional risks or vulnerabilities. Third we have architecture and design for security principles which are the establishment and enforcement of security policies such as role-based access control, data encryption, and user authentication. Fourth is keeping it simple, which as the name suggests minimizes the complexity of system designs and implementation which can significantly reduce the likelihood of a security flaw or vulnerability going unnoticed. Minimizing complexity can also lead to the code base becoming more maintainable as the code base will become easier to navigate and identify potential issues. Fifth we have default deny which is a general rule of thumb designed to explicitly deny access and permissions by default to reduce the attack surface a malicious use may utilize to bypass security methods in place. A system that does explicitly deny unauthorized users may implicitly grant them access. Sixth we have adhere to the principle of least privilege which co resided with default deny as it is best practice to minimize permission granted to perform necessary functions which in term will reduce the potential impact of a malicious attack done by an unauthorized user gaining access to the system. Utilizing this security policy will also assist in accounting purposes of identifying misuse of privilege such as when an account with lower access is suddenly attempting to give themselves access to higher level functions or data raising a red flag that a user may have their account compromised or is intentionally trying to bypass restrictions. Seventh we have sanitize data sent to other systems which involves the best practice of sanitizing data to ensure data provided or sent has not been corrupted or manipulated. Common sanitization strategies include data masking, using standardized formats, and data encryption specifically encryption in flight when data is being transmitted to other systems. Eight we have practice defense in depth which involves establishing a wide variety of overlapping security measures to protect systems and their data. This will significantly reduce the systems points of failure as there will be multiple layers in place to provide protection and an increased rate in which exploits will be fixed. Ninth we have use effective quality assurance techniques which will significantly reduce the number of vulnerabilities that make their way into production as develops and QA testers will thoroughly test the code to identify and properly take care of vulnerabilities as they come to light. Effective quality assurance testing may include but is not limited to penetration testing, failure testing, code reviews, and performance monitoring. Finally in tenth we have adopt a secure coding standard which is the establishment of strict guidelines and rules for reducing vulnerabilities by employing best coding practices early proactively identifying issues and establishing standards for high quality code. |
| **5** | Our coding standards displayed on this slide are ordered from highest priority ton lowest priority accurately reflecting the importance of enforcing these standards to prevent vulnerabilities and issues that introduce the highest severity and likeliness at the top and lowest severity and least likely on the bottom. As you can see from this ordering the coding standards that involve memory manipulation, protection, and data safety are of high priority as vulnerabilities related to these coding standards are to appear frequently and must be proactively mitigated as soon as possible. Memory issues such as trying to access free memory, attempting to read uninitialized memory, and manual memory management can lead to unexpected behaviors such as memory leakage, memory corruption, and exposure of critical system infrastructure. Preventing sql injections is also considered to be a high priority coding standard as proactively preventing attacks such as SQL injection mitigates the exposure of sensitive data like login credentials, financial information, and personally identifying information. Towards the middle we start to see standards to mitigate moderately appearing vulnerabilities that still may have a high impact on the integrity and security of the system but may be easily correctable or less impactful. Here we have string correctness, enforcing that we always allocate enough space in memory for string data and null terminator which can proactively mitigate vulnerabilities stemming from user input data and string data transmission. Properly handle all thrown exceptions which prevents unexpected errors from being thrown and not properly dealt with by the code. Although data type conversion may occur, and data loss is expected when converting larger data type data into smaller ones it is important that we explicitly do these conversion to minimize confusion and take note of data loss as it occurs through the use of casting. For the low priority standards, we have the use of meaningful and descriptive naming conventions which is important as it assists developers and QA testers in understanding where the data is flowing and the functionality of methods which is where vulnerabilities may appear. When we do not adhere to using this standard, that is when new vulnerabilities may appear as the code becomes confusing and unmaintainable. The data value standard of not reusing variable names in sub scopes is important for maintaining the highest quality code base avoiding the overwriting of data which can introduce vulnerabilities but is also easily manageable, noticed, and fixable as we can easily adjust variable names that are generalized and define variables in the smallest possible scopes possible to avoid overwriting data. Lastly in the lowest priority but still important we have avoiding the use of assertions for verifying runtime errors but instead use the for debugging purposes which will not introduce vulnerabilities unless the developer mistakenly includes critical functionality in assortment statements that will not be included in production mode. |
| **6** | Our encryption policies ensure protection is applied to data in all states, stored, in flight, and in use. Encryption at rest applies encryption algorithms such as AES-256 to data as it is being stored on a disk, database, or other data storage device ensuring that even if a data breach occurs the malicious actor is not able to easily translate the content obtained during the breach to be used in malicious ways. Encryption in flight applies protection on data through encryption as the data is being transmitted over both private and public networks. Protocols like tls and ssl are applied to ensure that the data integrity is maintained between end points and that the data is not manipulated in transit. Encryption in use deals with the last possible state data can be vulnerable to attacks and manipulation allowing for data to remain encrypted while accurately applying computations to it altering its content avoiding the content be decrypted in order to alter the data. For this stage we can utilize secure encrypted virtualization. |
| **7** | Triple-A policies Authentication, Authorization, and accounting are essential for granting and managing user access/permissions. Authentication ensures that all users of a system are who they say they are requiring all users of a system to utilize a basic login involving username and password that is compliant with secure password standards. Additional multi factor authentication may also be applicable for identifying user accounts. Authentication is crucial for preventing unauthorized user access and is incorporated into authorization and accounting policies. Authorization much like the name suggests grants permission and varying levels of access to accounts that have been deemed authorized. Role based access control can be implemented to abide by the security policy of least privilege applying heavy restrictions on roles lower in the company significantly reducing the surface area for attacks. Permissions should be granted for the minimal necessity of their role in the system. Authentication plays a role in authorization as user permissions can be tied to account information and access may only be granted once the use has been verified. Although proper use of the previous 2 policies does apply great protection against unauthorized access and use of a system there are cases in which accounts can become compromised or users are mistakenly given permissions outside of the necessity of their role which is where accounting comes into play. Accounting is the process in which activities are monitored and flags are raised when suspicious activity and possible system compromise takes place. Accounting is essential for book keeping purposes as every action taken by an account must be closely monitored and accounted for ensuring quick resolution of issues such as unauthorized access attempts, system disruptions, and potential security breaches. |
| **8-11** | The use of unit testing is highly regarded as an efficient way of proactively testing for vulnerabilities such as memory management vulnerabilities. Unit tests come in two different flavors negative tests which are intended to check for unexpected behavior and ensuring proper error handling is taking place, and positive unit tests are put into place to ensure the program is operating as intended validating the expected behavior of the program. A majority of the vulnerabilities that are identified in code bases belong under the umbrella term of memory management vulnerabilities so it is crucial for us to test against them to proactively avoid memory corruption, memory leaks, and memory overflows. The first test represented by the test name OutOfRangeExceptionTest is a negative test utilized to identify that an exception is thrown when memory out of range is attempting to be accessed. Ensuring that an exception is thrown for accessing out of range memory is crucial for preventing crashes and data corruption usually tied with overflow and underflow vulnerability exploits that can leak sensitive data and the inner workings of the system.  The next slide demonstrates the use of a positive test called ClearErasesCollectionTest which ensures that when the method clear is called to a collection items added to that collection do not persist in memory. This is important for identifying that entries do not remain in memory once a collection has been cleared avoiding unmanaged resource problems that can degrade performance or leave sensitive data in a system that is no longer being actively protected. It is important to not only ensure that incorrect behavior is being handled but correct behavior is operating as intended.  This next slide demonstrates another positive unit test called resizeDecreasecollectiontest. This test ensures that entired in a collection are erased when the collection is downsized. Identifying whether or not a collection has been downsized is important as like previously it ensures that data that has been cleared does not persist to take up unmanaged memory space that may be required to be used later on.  This last unit test slide provides us with another accurate demonstration of a positive unit test called CollectionSmartPointerIsNotNullTest which ensures that a smart pointer has been initialized and is pointing to a valid memory location. This type of testing is critical for ensuring that any subsequent test conducted on the collection is done on a valid object in memory. |
| **12-13** | The following slide illustrates the DevSecOps process and its various phases. DevSecOps differs from the DevOps process currently being utilized by integrating security into every phase of pre-production and post production enhancing code quality and ensuring compliance with security principles.  This next slide details the various phases within the DevSecOps in which we can integrate the use of automation tools to improve the use of secure coding practices.  In pre production we can integrate automation tools in the planning and design phase and the build and testing phase. In the planning and design phase we should utilize threat modeling tools such as IriusRisk OWASP Threat Dragon, and Microsoft threat modeling to ensure our system architecture and tech stack does not contain vulnerabilities that may weaken the integrity of the system. In the build and test phase we should integrate the use of static analysis tools such as CPPCheck, Coverity, and sonarqube on top of heeding to compiler warnings to improve accurate and expanded identification of known vulnerabilities and recommend us solutions for resolving these issues present in our code base.  In post-production that phases in which the use of automation tools to drive security are the maintenance phase, the transition and health check phases, and finally the monitor and detect phase. In the maintenance phase we should continue to utilize static analysis tools before pushing code to the main code base to ensure we are not adding new vulnerabilities that can be exploited during production alongside using automatic dependency checking tools to ensure that all dependencies are up to date and do not introduce new vulnerabilities. Examples of tools we can use for this are OWASP dependency scanner, CPPDepend, and CPPCheck which if you recall we also utilized for static analysis. For the transition and health check phase we should integrate the use of automatic penetration testing utilizing tools such as Metasploit to confirm the absence of exploitable code or network insecurities. Finally we should integrate the use once again of static analysis tools into the CI/CD pipeline already established in the monitor and detect phase to ensure continuous development follows secure coding principles and coding standards outlined in the security policy. |
| **14** | Vulnerability resolution often incorporates one of two options immediate action or delayed action both with their own benefits and risks in implementation. Taking delayed action towards vulnerability resolution can be costly later on as vulnerabilities may be intertwined with large parts of code that need to be refactored. Refactoring code can be time consuming and costly as additional functionality may rely on vulnerability ridden code. Delayed action can also be considered costly as exploitations can take place risking customer data resulting in loss of customer trust. Immediate action can be beneficial but does still come with problems such as the initial large investment for resolving issues as they appear and costing company time that could be spent releasing new features and upgrading existing systems. The solution for utilizing a delayed response to vulnerabilities is to identify and prioritize finding resolutions to critical issues first when the time comes to fix it. The solution for utilizing an immediate response is to prioritize a shift in focus to counteract problems as they appear using automation tools such as static analysis and dependency checkers. The benefits of a delayed response are that it does not require a huge initial investment of resources meaning that resources can be gradually allocated to resolve issues later. The benefit of an immediate response is that vulnerabilities are resolved immediately reducing the impact of them, reducing the number of vulnerabilities currently impacting the system, and finally reducing the long term cost of fixing the vulnerabilities later on. These two strategies both have their benefits and risks but one thing that I have not covered is the strict guidelines for resource allocation which at different time periods may vary. The steps that we should take for vulnerabilities is overall immediate action. We must first identify the vulnerability or vulnerabilities currently impacting our system, allocate resources so that teams can work on resolving issues, and proactively ensure secure coding compliance is being done by conducting compliance trainings and thurough code reviews/testing to prevent vulnerabilities from being introduced. |
| **15** | There are of course gaps in the security policy such as guidelines on how to properly allocate resources so that we can integrate these tools and security measures into the development process. There will also of course be improvements and additions to this security policy as new secure coding standards are widely adapted, and new vulnerabilities arise with the growth in technology. Recommendations for filling in these gaps consist of allocating additional resources towards improving security purchasing security tool licenses as needed and conduct more trainings to ensure compliance. We should also periodically update our security policies when additional secure coding standards are needed/discovered keeping it up to date. |
| **16** | Additional coding standards that should be adopted into the security policy are proper memory deallocation to optimize performance and ensure all memory is managed, proper exception catching in order, and ensuring that all functions expected to return a value are returned a value of the explicit data type. Integrating this extensive security policy into day to day workflow will ensure we are proactively preventing vulnerabilities rather than resolving them once they have already been exploited. Thank you all for being present for this security policy presentation and have a good day! |