# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is Levi and this is my security policy presentation for CS-405: Secure Coding. |
| **2** | This security policy was created for the organization Green Pace and it aims to specify coding standards and best practices that must be followed in order to maintain the safety and integrity of our application as well as our user’s data. The individual parts of this security policy make up some of the layers that are needed for the defense in depth strategy. |
| **3** | Using SEI CERT’s definitions for the risk assessment section, I’ve added all of our security risks into their respective matrices based on that. Starting with the “Likely” matrix, we can see four security risks that are likely to be exploited. Its counterpart in the bottom right, also has four security risks but these are unlikely to be exploited. The “Priority” matrix has three security risks in it and it means that the risk is likely to be exploited and the severity of it being quite high. Finally, the “Low Priority” matrix has four security risks but they are unlikely to be exploited and have a low severity associated with them. Using a static analysis tool like CPPcheck, we can detect these vulnerabilities early on. |
| **4** | In this security policy, each of the coding standards adhere to a coding principle. As can be seen from the table, these ten principles are validate input, heed compiler warnings, architect and design for security policies, keep it simple, default deny, adhere to the principle of least privilege, sanitize data sent to other systems, practice defense in depth, use effective quality assurance techniques, and finally, adopt a secure coding standard. |
| **5** | Here are our coding standards, each one is ranked in order of prioritization based on our threat matrix. One being the highest level and ten being the lowest level.   1. Ensure that operations on signed integers do not overflow.    1. This is number one because it is relatively simple to do, anyone just entering a number that is too large could cause this issue and the result can end up with someone inputting arbitrary code. This coding standard should be enforced whenever we are using signed integers and taking in user input. 2. Do not read uninitialized memory.    1. This is another one that if violated, can allow for users to run arbitrary code. This coding standard should be followed whenever we're working with variables and plan to use them in an expression. 3. Do not access freed memory.    1. This is yet another one that can be exploited for arbitrary code execution if not adhered to. This coding standard should be implemented anytime we are working with pointers. Dangling pointers can lead to a whole host of issues like data corruption, system crashes, and as I just said, the execution of arbitrary code. 4. Sanitize data passed to complex subsystems.    1. If violated, allows for the execution of arbitrary code. This coding standard should be applied every time we are passing string data into our subsystems, for example anytime we are calling the system() function or a third-party component. 5. Do not modify the standard namespaces.    1. This standard should be applied in every situation where we might add or extend declarations in the standard namespace such as std. If the standard namespaces are modified, this runs the risk of naming conflicts when adding new functions, classes, etc.. 6. Write constructor member initializers in the canonical order.    1. This is our first medium severity security risk but nevertheless it’s just as important to not violate. This coding standard should be applied every time we are writing or modifying a class constructor that uses an initializer list. 7. Obey the one-definition rule.    1. This one has a high severity, but it is unlikely to be exploited, which is why it’s down so far on the list. This coding standard should be applied universally in any C++ program that spans multiple source files. This ensures that every non-inline function, variable, or object has exactly one definition across the entire program. 8. Handle all exceptions.    1. This coding standard should be applied for any code that is likely to fail. For example, opening and reading or writing a file, taking in user input, or performing arithmetic. Without proper exception handling, this can lead to the application crashing. 9. Do not attempt to modify string literals.    1. While this security risk is low severity, it’s still best practice to implement this coding standard anytime string literals are being used. 10. Use a static assertion to test the value of a constant expression.     1. Assertions are a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities. static\_assert allows incorrect assumptions to be diagnosed at compile time rather than runtime therefore there is no overhead and should be used instead of regular assertions when dealing with constant expressions. |
| **6** | **Encryption at rest**  Encryption at rest is when stored data, like on a smartphone or hard drive, is in an encrypted state by using encryption algorithms. So even if the data is obtained through malicious means, it’s still unreadable unless the encryption key is also known for it to then be decrypted. This policy is important because we need to protect the user data that we collect and store. Without this policy, we would be an incredibly easy target for malicious attacks. A very common asymmetric encryption algorithm is advanced encryption standard (AES) that comes in 128, 192, or 256 bits as key lengths. I highly recommend this encryption algorithm for encrypting data stored in our database because without access to the key, the data is unreadable and practically impossible to crack. So even if someone were to illegally gain access to our stored data, it would still be secure because the attackers would not be able to gain any useful information from the encrypted data. It’s also vital to have secure management of encryption keys. A key management system can help automate and enforce key policies like making sure they are secure, tracked, and maintained throughout their lifecycle. It’s also important to note that key management systems allow us to store keys separately from our encrypted data. To determine who has access to these keys, we can use role-based access control and regularly audit to see who and when someone accesses these keys.  **Encryption in flight**  Encryption in flight refers to data being in an encrypted state as it is in transit from source to destination, typically over a network. While data is in transit, it must be encrypted otherwise if someone intercepts that data while it is in transit, they are able to gain access to the data in its natural unencrypted format. Transport layer security (TLS) is a protocol that is designed to protect data in transit by encrypting the communication between two endpoints, in most cases, this is between a client and a server. This process is done through what's called a handshake where the client and server agree on what cipher suite to use, authenticate certificates on at least one end of the connection, typically the server side, and then securely exchange keys to use symmetric encryption when the handshake is complete.  **Encryption in use**  Encryption in use is the encryption of data while the system is using it (i.e. processing, updating, reading, etc.) This policy aims to fill in the gap as when data is in flight and at rest is now encrypted, data while it is in use becomes the weak link and will naturally be targeted. Therefore it is important to mitigate any possible data exposure by encrypting data while it’s in use. A common technique for doing this is homomorphic encryption (HE), which allows for encrypted data to be worked on as if it is in plain text while keeping it in cipher text. To give a real world use case for this policy, a hospital can encrypt patient data and send it to a research lab or another hospital, that data can be used on a machine learning model without ever seeing the actual patient data. Therefore, not violating any laws like HIPAA while also helping further researching efforts. It’s important to enforce this policy whenever we are taking sensitive data in memory. This is because data is at its most vulnerable in this stage because it's immediately available to be used. It can also lead to data loss or sensitive info being exposed during runtime to hackers. |
| **7** | **Authentication**  Authentication is the process of confirming a user is who they say they are. It’s essentially asking the user “Who are You?” in which the user will provide some credentials that the application already has stored in order to authenticate whether the provided credentials matches what the application has in storage. If what they enter does not match any of our stored credentials, then they are not given access until they do enter something that matches. If a user is new, then we need to ensure the data given to us is properly stored and ready to use for future authentication. In most cases it is a good idea to limit the amount of times a user can get it wrong before the system enacts some lockout policy to prevent further attempts to login from a specific user for a limited time. This policy applies whenever a user is trying to sign in to our application and we need to verify the credentials given. The next step is authorization:  **Authorization**  Authorization is simply the set of attributes a user is authorized to perform. Certain users have a certain level of access to the network, system, or application. This step is asking “What are you allowed to do?” The way this is typically implemented is either through role-based access control (RBAC) or attribute-based access control (ABAC). For RBAC, permissions are tied to the roles a user has, which are usually predefined with a specific set of allowed actions. So instead of assigning permissions to each user, we can simply assign roles to users such as “admin” or “user” that come with their own set of permissions (Nolan, 2025). This is important because it enforces the principle of least privilege.  ABAC is a similar approach but instead uses attributes of users to determine the level of access they have. Conducting regular permission audits are a great way to keep the access control policy up-to-date as projects evolve, employee roles change, and organizational policies change. This ensures that each user has the appropriate level of permissions for what they need to do. This policy applies whenever a user is trying to perform an action within the system. This helps prevent regular users from doing actions or accessing parts of a system they shouldn’t be allowed to do or access like accessing sensitive files or adding changes to the database through commands.  **Accounting**  Accounting is the recording and monitoring of users actions on the network and or application who have been authenticated and authorized. This step is asking “What did you do?”  This policy is important to know what is going on within the system. For example, if someone notices unusual behavior within the system, an audit can be done in order to see anything that may have led to that unusual behavior.  This allows us to gain insight into what was done to exploit a vulnerability in our system.  In order to store these records and make sure they are available, we can use a Security information and event management (SIEM) system to store the logs and of course, be sure to apply our encryption policies to these logs as well. |
| **8** | These next couple of slides are various unit tests using the framework Gtest from Google. The purpose of unit testing is to verify that certain units of the application are working as it is expected to. This is done in order to find any issues early on in the development process. It’s considered best practice to use unit testing frameworks instead of writing custom tests. This is because it helps standardize and sometimes automate the testing across all projects. Unit testing is done in the testing phase of the DevSecOps pipeline and should be done in almost every project to ensure every unit is working as it should.  In this slide, we are testing that if the collection vector exceeds its max size, it throws the appropriate exception and from our results, we can see that it appears to be the case. |
| **9** | It’s a good idea to have a mix of positive and negative test cases, the last one was a negative test, and this one is a positive test to verify that we can add five elements to the vector collection. Once again, from the results, we can see that that is the case. |
| **10** | This unit test is another positive test to validate whether or not the resize function changes the size of the vector to 10 elements. |
| **11** | This final unit test is a negative test case that verifies that an out of range exception is thrown when we try to access beyond the size of the collection vector. |
| **12** | The DevSecOps pipeline as shown in the diagram, depicts the entire process of developing and maintaining a secure application, from pre-production to post-production. A principle of DevSecOps is shift left security, meaning to integrate security as early as possible and in as many stages as possible. Many stages in the DevSecOps pipeline can benefit from automated tools to expedite this process. |
| **13** | Some of these tools include a threat modeling tool for the assess and plan stage, an IDE along with security plugins and Vulnerability scanning tools for the design stage. For the build stage, we have SAST tools, compilers, and dependency checking tools. Verify and testing has Dynamic application security testing tools, test coverage tools, unit testing frameworks, and fuzz testing tools. And finally, for the monitor and detect stage, we have security information & event management tools and information security continuous monitoring tools. |
| **14** | There is a solid consensus among software security professionals that security should be done as early as possible, the earlier the better. Traditionally, security was an afterthought, often just bolted on near the end of the development process, this led to finding vulnerabilities incredibly late and would be a huge hit in time and money having to fix it. The benefit of acting now and integrating security as early as possible, we solve those two issues relatively easily. Along with it being easier to implement from the beginning, our application gains a greater resilience to threats. However, this philosophy has some shortcomings of its own. For instance, there may be some organizational resistance to this cultural change of shift left security, the learning curve for all the tools necessary can be steep and there may not be adequate training to deal with that. |
| **15** | So here are my recommendations  The senior staff at Green Pace need to foster the cultural shift that DevSecOps requires in order to fully utilize its benefits. Perform regular audits to identify where security may be lacking whether that be things that are not included in this security policy or things that just aren't being done at an acceptable level.  A more clear idea of the level of acceptable risk should be determined in the security policy. Some of the risks are low severity yet are high or medium in remediation cost like STD-007-CPP. Green Pace should make the decision on whether or not these security flaws should be focused on. Determining the specific roles or attributes for access control. We also should have a clear definition of who should have what roles or attributes for access control.  And finally, we should have a clear classification of the data we will be collecting. |
| **16** | In conclusion, security must be considered from the very beginning of the development process and needs to have a proactive approach to it. DevSecOps training and culture must be embraced from the top down for it to work at its full capacity. Conducting regular gap analysis will help identify any weak links in our process. The security policy needs to be reviewed regularly and updated when necessary as the project changes and our organization changes.  Thank you. |