# CS 405 Project Two Script

Youtube Link: <https://www.youtube.com/watch?v=zVNcZOM9y9Y>

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello everyone, my name is Alexis Fuerte, and today I will be presenting the Green Pace security policy guide and providing implementation guidelines and recommendations for maintaining it in the future. |
| **2** | This security policy is going to be used to define the core security principles, c/c++ coding standards, authorization, authentication, and auditing standards, and data encryption standards. This policy was necessary in order to ensure that as the organization grows, and as security threats multiply, our company will have principles and rules specific to our area of work that standardize the way that our developers meet the needs of security for our company. This security policy will be used to support the defense in depth best practice because it embodies a few of the different layers, such as the network security, host security, endpoint security, and app security portions. We are ensuring that there is a specific guideline on how our code should be written in order to avoid any additional security vulnerabilities and performing constant checks and vulnerability scans throughout the course of code creation. |
| **3** | The threat matrix table has four different options, likely priority low priority and unlikely. Under likely, we’re going to have or Data Value coding standard, which is there to prevent integer overflows, the Data type coding standard which also ensures there aren’t any instances of overflow except on signed integers specifically, and the Exceptions policy which handles all of the exceptions thrown. Under the Priority section, we have the SQL injection coding standard, which defends against malicious data accessing intricate subsystems that could possibly execute commands or trigger unintended actions. Then there is the String Correctness coding standard, which ensures that storage for strings is adequately sized. Lastly for this section we have the Memory Protection standard, which ensures uninitialized memory is not read. In the low priority section, we have the function return value coding standard, which ensures a value is returned from all exits, and the Qualifying reference types standard which ensures a reference type is not qualified with const or volatile. In the unlikely section, we have the commenting standard. We all know commenting is extremely important, but in terms of threat possibilities, it is unlikely that a lack of commenting will lead to a security breach. This isn’t to say that it is impossible, but we just aren’t as likely to see it. Lastly we have the assertions coding standard. This coding standard tests for conditions that should never be true. |
| **4** | These are the ten core security principles. First we have validate input data. Input validation is a process that ensures data entered by users is safe and reliable. This means examining and verifying the integrity, formatting, and type of input data to protect against errors, vulnerabilities, and malicious activities. This will aid with preventing/defending against data corruption, injection attacks, buffer overflows, etc. Under this principle, I have the Data Type coding standard, Data Value coding standard, String correctness data standard, and SQL injection coding standard. At number two, we have Heed compiler warnings. To heed compiler warnings is to pay attention to warnings that are issued by the compiler during the code compilation process. Under this principle, I have the Data Type coding standard, String Correctness coding standard, SQL Injection coding standard, and the Function Return Values coding standard. Up third is the Architect and design for security policy principle. I don’t have anything specific under this principle, but it is a good overall standard that makes sure code is meeting requirements in multiple areas, for example if creating code that requires different privileges, making sure that there are subsystems that have the matching privileges. Fourth we have to Keep it simple. To keep it simple is to make sure that your code is readable, and makes sense. If the code is too complex, then it could be easy to miss errors when the code is being ran. This is avoided by creating smaller, more readable code. A few coding standards I have under this principle are SQL Injection and Qualifying reference types. At the fifth position is default deny. This one ties in to the principle in the number 6 position, Adhere to the principle of least privilege. Both of these are just ensuring that processes will execute with the least amount of privilege necessary to complete the job. This minimizes the potential impacts of security breaches and compromised accounts. I would say that SQL injection can also fall into this section. At number 7, we have Sanitize data sent to other systems. This mitigates potential vulnerabilities and attacks triggered by malformed or malicious data, ensuring the integrity and reliability of the software system. String correctness falls under this category. Up next is Defense in depth. We just hit on this a few slides ago, and it is also not one of the main principles exhibited in the coding standards, but it is a good principle to always keep in mind. It helps to ensure that if a defense method fails, there are multiple additional layers of defense available to protect a system. This makes it more difficult for attackers to get through and minimizes any potential damage caused by breaches. At 9 is to use effective quality assurance techniques. Quality assurance exists to help find and eliminate vulnerabilities within the code. This could be done in many ways, such as through a contracted company paid to try and hack into your code and find vulnerabilities, or through multiple levels of testing. A few coding standards that fall under this category are Assertions, Exceptions, and Function return values. Last but not least is to Adopt a secure coding standard. To adopt a secure coding standard means to just check whatever the secure coding standard is for whichever language you are working with. A few coding standards fall under this too, such as Commenting, Assertions, and Memory protection. |
| **5** | My personal ranking of importance for the coding standards is based off of general best practices and common considerations. In my number one spot is ensuring that operations on signed integers do not result in overflow. Preventing overflow is important to avoid unexpected behavior and ensure accurate calculations. Up next is using a valid range of integer data types to prevent integer overflow between both signed and unsigned integer conversions. Properly managing integer ranges helps prevent data loss and maintain consistency in calculations. At number three is ensuring that the allocated storage for strings is adequately sized to accommodate the character data along with the null terminator. Sizing strings correctly is crucial to prevent buffer overflows and memory corruption. Then we have “Do not read uninitialized memory”. Reading uninitialized memory can lead to unpredictable results and security vulnerabilities. At number five is cleanse the data that is passed to intricate subsystems. Ensuring data integrity and validity is important for the correct functioning of intricate subsystems. For 6, we have handle all exceptions thrown. Proper exception handling helps maintain the stability and reliability of the software. Then we have functions that return a value must return prevent potential bugs and unexpected behavior. Consistent and complete function returns prevent potential bugs and unexpected behavior. At number 8 is to use assert statements to handle errors that test for conditions that should never be true. Assert statements help catch logical errors during development and aid with debugging. I put commenting at number nine. Commenting is common practice and lets other developers understand what that block of code is meant to do. Code commenting promotes code readability and collaboration among developers. Last at number 10 is to never qualify a reference type with const or volatile. It is important to understand the implications, but it is much less of a critical concern compared to other items in my list. |
| **6** | Now we’re gonna talk about the three types of encryption. Encryption in rest helps protect data that is stored on a disk. It applies because we need to keep that data secure from possible attacks should the hardware get into the hands of others, or should an attacker get into your personal system. Encryption at flight protects data confidentiality between a client and a server. This data can be intercepted by an attacker, so it protects it through encryption.  Encryption in use ensures that there is continuous protection of data no matter what stage or location. This prevents it from being unsecured. This policy is important because it makes sure data is always being safeguarded, reducing the risk of a possible breach. |
| **7** | The triple A framework consists of authentication, authorization, and accounting. Authentication is used to verify the identity of a user. This can be done in a few ways, such as usernames, passwords, biometrics, etc. Through authentication, systems are ensuring that only authorized users have access to resources. This keeps the information secure. Authorization determines what roles, privileges, and accesses users have after authentication. This controls access to specific functions, resources, and data in a system, ensuring that only authorized users are performing certain actions. This keeps information secure because people can only access data that they are set to be able to access. Accounting tracks the use of resources and actions done by users that have been authenticated and authorized. Examples of information logged are time of access, what they did, and who did it. This helps to recognize who might be accessing things that they do not need to access. |
| **8** | In this course, we had to perform unit testing on some of the code we were provided. Unit testing is a software testing technique that tests individual components of software in isolation to make sure that they are functioning correctly. This is important because it helps with early bug detection, improves code quality, explains how code is expected to behave, and boosts overall confidence in code security. In the module where we performed some of this testing, we saw that most tests ran correctly, for example the test that ensured that collection is empty when collected, or that something can be added to an empty vector. |
| **9** | This diagram represents the process of code creation and implementation in a figure eight pattern, with the left side being pre production and the right side showing actual production. Generally, you start with assessing/planning in order to best visualize what threats you might encounter, how you would respond to new threats, etc. Then you go into the designing portion, where you ensure you have a security test driven design and follow best practices. In the build section, you make sure that you are building secure code. Then you verify and test through vulnerability scanning, and security testing. Then you move into the production side of the figure eight, where you transition and perform health checks. At this point, you are configuring and deploying your code and verifying that it is working properly. After that, you go into the monitor and detect section, where you focus on who is accessing your creation, and check for any intrusions. If there are, you respond by blocking attacks and turning off services if necessary. Then you go into the last portion, which is maintaining and stabilizing, where you return to a stable state again after an attack before starting the cycle over while making repairs. |
| **10** | The DevSecOps pipeline integrates security practices and tools throughout the entire development process. I just explained it in the last slide so I’ll move into the next portion, which is to summarize external tools used in the diagram. Some external tools could be SAST tools which are used in the development phase to analyze source code for vulnerabilities. There is also DAST tools which simulate attacks on applications. SIEM tools could also be used to collect and analyze security events for incident detection and response. You could also use parasoft C/C++ testing to verify and test code for vulnerabilities as well. These tools play a crucial role in enhancing security by identifying vulnerabilities, simulating attacks, monitoring security events, and validating code quality in the DevSecOps pipeline. |
| **11** | For one of our journals, I wrote about how “Don’t leave security until the end” is a best practice in coding. It focuses on ensuring that security is thought about and implemented throughout the coding process. This helps to ensure that the developer doesn’t miss anything at the end that could leave the code vulnerable to attack, because these threats and vulnerabilities are identified and addressed as the code is being developed. It is a proactive approach that reduces risks of possible security breaches, creating a more solid product. Should you chose to wait until the end to start worrying about code security, you could increase the chances of vulnerabilities, and you risk having to pay more costs in remediation. This could also impact project timelines as more time, resources, and effort is needed to fix these issues, especially if the issues require extensive code changes. You might also lose some trust from your customer base is a vulnerability is missed that could have easily been caught early on. |
| **12** | This guide is just a starting point for the team to use, but as time goes on I expect that more standards will need to be added, and that is what we want. Having a document for our developers to rely on and constantly refer to will ensure that our code is standardized and easily readable. With these rules on hand, and any updates that come to follow, we know that our team will truly look as if we are working as one unit as opposed to a bunch of different developers code being merged into one product. Threats are also always developing, so we will need to continue to update the policy as we go in order to accurately defend against new threats and vulnerabilities. I also did not have a coding standard to go under each of the ten principles, so finding new coding standards to add that will fully represent those principles is also something we hope to do in the future. |
| **13** | Some standards that should be adopted in order to prevent future problems are testing often to exploit vulnerabilities, making sure we are focusing on our defense in depth policy, which we also lacked additional coding standards in, and that we continually monitor the code and look for ways we can change it to make it more secure. By testing regularly, we can detect vulnerabilities early on and save time and effort for the team when it comes to fixing these errors, as well as reduce the likelihood of exploitation. By using the defense in depth policy as a standard, we ensure that we have multiple layers of added protection to truly secure our code through standard coding practices such as input validation, error handling, authentication and authorization, and more. Lastly, through regular monitoring, we can identify security gaps and other areas for improvement, as well as perform security audits to validate that we have not experienced any breaches. |
| **14** | Here are some of my sources, thank you for listening and I hope that you gained a better understanding of the new green pace security policy! |