# CS 405 Project Two Script

Andrew Laipple

12-15-2024

<https://youtu.be/LM7VP2PnZvM>

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello, my name is Andrew Laipple and I will be going over the Green Pace security policy with you today. |
| **2** | The security policy at Green Pace outlines the top security principles and coding standards. By following the policy, it allows the developers to create secure programs that use the defense in depth strategy. This strategy provides multiple layers of security to give the best chance at preventing attacks. For example, first you might want to prevent an unwanted user from accessing your system, that’s one layer. If they get into the system, you want to prevent them from accessing any restricted data, that’s another layer. If they then somehow access sensitive data, you want that data to be encrypted so they can’t use it, that’s another layer. Each layer addresses a certain vulnerability to form a more secure system as a whole. |
| **3** | The threat matrix is a summary of the security risks at Green Pace. These are broken down into high and low priority as well as likely and unlikely risks. The top priority risks are things like sql injection, data theft, and unauthorized access. Any one of these attacks could cause serious consequences and should be prevented at all costs. Likely risks include Buffer Overflows, String Manipulation Errors, Wrap Around/Truncation. These are coding type errors that could lead to remote code execution that definitely should be addressed during the coding phase. Low priority issues include System crashes, Exceptions, and Out of Memory issues. These types of things should be caught during the testing phase, so they aren’t high priority. Unlikely issues are things like Denial-of-Service attacks, Insider Attacks, or Phishing attempts. While these are serious issues, they are much less likely to occur. You’ll notice this is the first slide with a superscript. This number refers to the references slide at the end of the presentation. So for this slide reference number 4 was used as a source. |
| **4** | The 10 core security principles at Green Pace are Validate Input Data, 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 Adopt a Secure Coding Standard. You’ll notice that each coding standard is mapped to a principle. I will go over each coding standard in the next slide, but as you can see each coding standard applies to one or more core principles. |
| **5** | There are three levels of coding standard that are ranked based on the priority of the standard. The highest priority is level 1, the lowest priority is level 3. The system of prioritization is based on how severe the risk is, how likely it is to occur, and how costly it is to fix. Level 1 standards are associated with severe risks and would be highly costly to fix. Level 3 standards are associated with low risks and low cost to fix.  The highest priority coding standard is STD-SQI-001 which is Use Prepared Statements to prevent SQL injection. SQL injection is a serious threat that can lead directly to data theft of sensitive information. Next is STD-STC-002 which is Null Terminate All Strings. If strings are not null terminated it could lead to things like remote code execution. Next is STD-MEP-001 which is Don’t Mix and Match Memory Management Functions. Not deallocating memory properly can cause undefined behavior which could lead to system crashes or severe vulnerabilites in general. STD-STC-001 which is Do Not Use Unbounded String Operations. This can lead to buffer overflows which could cause crashes or things like arbitrary code execution. The last level 1 coding standard is STD-MEP-001 which is Avoid Freeing Memory Multiple Times. Memory management issues can lead to arbitrary code execution as well. The first level 2 standard is STD-DTV-001 which is check for wrap-around. This risk can cause calculations in the code to be incorrect, possibly leading to buffer overflows. Next is STD-DTV-002 which is avoid integer truncation. This is similar to DTV-001 and can lead to similar vulnerabilities. The last level 2 coding standard is STD-DDT-001 which is Use Minimum Data Types. Just like the other two standards at this level this could lead to calculations being incorrect and buffer overflow issues. The first level 3 coding standard is STD-EXC-001 which is Exceptions Should be Handled. Obviously you don't want to have exceptions happen at all, but if they do they need to be handled to know exactly what happened. Then the problem can be investigated and fixed. The last level 3 coding standard is STD-ASS-001 which is Do Not Modify Values in Assertions. Assertions do not run on release builds so no modifications of values or function calls should occur in an assert statement. If they do then the release build will not perform these actions resulting in the program not working properly. |
| **6** | The Green Pace security policy outlines 3 encryption policies. Encryption at rest is for any Data not currently being used or sent over a network. This would include sensitive data such as usernames, passwords, maybe financial information. Because this data is not actively being used it needs to be encrypted using a very strong encryption algorithm in case an attacker did get their hands on it they wouldn't be able to decode the data. Encryption in flight is for any data that is sent across a network. This includes things like login credentials, possibly account information. A strong industry standard encryption algorithm should be used for this such as TLS. Encryption in use is for data that has been loaded into memory while a program is running. This can include the same types of data in the other scenario such as usernames and passwords, payment information or account information. Depending on how often this data is used during execution will determine the type of encryption algorithm that should be used. Using too strong an algorithm would cause the program to run very slow so there is a tradeoff that must be analyzed. |
| **7** | The Green Pace security policy outlines the triple-a strategy for security. The first a is authentication which refers to validating users that have access to the system. Typically this is done with a user name and a password. For more security a two-factor authentication system should be used such as sending a code to an e-mail or phone number. This data should be encrypted when sent and stored per the encryption policies. The second a is authorization which is used to limit the actions of the users of the system. Each category of user should be allowed to perform different tasks and the principle of least privilege should be followed which means that only actions that allow a user to do their job should be authorized. The only user that should be able to set the privileges of other users is an administrator. The last a is accounting which is logging and recording the activities of users as they perform actions in the system. If an attack occurs these logs can be reviewed to see when, where, and how an attack occurred. They are very useful in being able to fix problems and prevent similar attacks in the future. There are tools such as Logic Monitor or DynaTrace that record this type of system information for accounting purposes. |
| **8** | All coding standards should be unit tested to make sure they are being adhered to. For the next 4 slides we will focus on the unit tests associated with STD-STC-002 which if you remember is to make sure all strings are null terminated before use. In this first unit test titled endless loop we create a string that is not null terminated and then loop through the characters of the string until a null terminator is hit. We then verify that the times that the loop was entered is not the same as the maximum length null terminated string that could fit inside dest\_str. In practice most of the time this won’t necessarily result in an endless loop since the loop will terminate once a byte of 0 is encountered. But it illustrates the point that inside the loop there could be logic occurring that could end up overwriting memory or cause a system crash since the loop should only be performed a maximum of 9 times given this scenario. |
| **9** | This 2nd unit test, named non endless loop, is similar to the first only this time the code ensures that the string is null terminated before looping. We verify that the number of times looped matches the length of the string. |
| **10** | The 3rd test, named large copy crash, is a negative unit test and shows how a non-null terminated string can cause a system crash. In this test a strcpy is performed on a non-null terminated string. For the purposes of the test, we create a very large string buffer in order to simulate a potential non-null terminated string. We verify that the system crashes when the strcpy is performed. |
| **11** | The last unit test, named large copy no crash, is similar to the 3rd only this time the code ensures the string is null terminated before the large copy occurs. We verify that after the copy the strings are equal and no memory has been overwritten. |
| **12** | On this slide we see the results of all 4 unit tests, which all pass. |
| **13** | This slide shows a diagram of the entire DevSecOps pipeline, from the planning stage to the release and maintenance stages. |
| **14** | The DevSecOps pipeline is a process that is used to incorporate security into all phases of the software development lifecycle. It is important to remember that security should not be limited to a single phase such as coding or testing. Instead, it needs to be thought of and incorporated at every phase for a robust security policy. There are many tools that can help as well as automate security during the various phases. At the requirements phase the SQUARE tool can be used. SQUARE stands for secure quality requirement engineering. This isn’t so much an automated tool, but a process developed by SEI for creating and analyzing security requirements. At the design phase threat modeling is helpful to determine exactly what threats the system my face. At the coding phase static analyzer tools such as cppcheck can be used to fix common vulnerabilities. A tool like polyspace bug finder can be used specifically to make sure any coding standard violations have occurred. Also, the compiler can be adjusted to be more strict and give warnings when encountering vulnerabilities. The verification phase is the final phase before release and where rigorous testing like fuzz testing and attack surface reviews should take place. Once a program is released tools for monitoring the program need to be in place to catch any security breaches. As mentioned in an earlier slide tools like DynaTrace or Logic Monitor could be used. |
| **15** | For this slide we’ll take about the risks and benefits of acting immediately or waiting to fix security issues. In general, it is always better to find and fix problems as early as possible during development because the later in the development cycle an issue is found, the more costly it is to fix. So, for example if an issue is found in the verification phase, it could be that requirement, design, and code need to change in order to correct the issue. If the problem originates from the requirements, then if the issue would have been found during that phase it wouldn’t have caused any further issues in other phases. But because it is more costly that doesn’t mean worth it to ignore an issue just to save money or release a project on schedule. Doing so could cause catastrophic effects. For example, let’s say an SQL injection vulnerability is discovered at the end of the verification cycle. Fixing the issue would delay a project costing a company thousands of dollars so they decide to not delay and release the program with the vulnerability hoping to fix it before it becomes an issue. If it is not fixed in time and the vulnerability is exploited to say steal someone’s financial information it will cost much more than in the long run. The reputation of the company will be at stake and may cause users and customers to not trust them and the entire company goes under just to save some money. |
| **16** | Here are some recommendations to improve the security policy. First the security policy should be updated to include code standards for all principles. The current version has no code standard that maps directly to principles 3, 6, 7, and 10. Next, it might be better for the policy to recommend a single tool for static analysis of code instead of multiple tools. That way developers will have a consistent tool set on their computers and won’t have to have multiple programs installed. As with any company document, it needs to be updated over time based on the experience gathered while using the document. This same strategy should be used for the security policy. For example, if an attack occurs after a product has been released, once the vulnerability has been identified and corrected a new principle or code standard might need to be added to address the vulnerability if it was completely overlooked befofe. Lastly, the current security policy is written specifically for C/C++. If Green Pace expands the programming languages it uses it may be beneficial to incorporate those into the existing document or create separate security policy documents for each language. |
| **17** | In conclusion, the current security policy is a great start to get developers at Green Pace to start thinking about security at all phases of development and to follow standards while coding to reduce the likelihood of having known vulnerabilities within programs. However, there are many known vulnerabilities out there. The SEI Coding standard for C and C++ has multiple categories of standards that have not been addressed by the current security policy such as expressions, floats, and APIs. This doesn’t mean that Green Pace needs to adopt all these coding standards in their security policy, but it does mean that these categories should be reviewed to determine how relevant they are to the code and programs that Green Pace produces. If they are important then they should be incorporated. |
| **18** | These last 2 slides show the references that were used to help create this presentation. |
| **19** | Thank you so much for your time! |