# CS 405 Project Two Script Aaron McDonald

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello everyone, my name is Aaron McDonald. Today we will be discussing our security policy and coding best practices for Green Pace. |
| **2** | The main theme we will be following is a theme of Defense in Depth. As shown with this diagram here, defense in depth is a defense technique with layers of different security requirements. Each layer provides a level of redundancy to our security system. Most importantly with a defense in depth technique, security should be designed into systems from the beginning and never left to the end. Adding security procedures to a system can add a lot of complexity to that system, meaning that it becomes far more work later in the project than it does at the beginning of the project. |
| **3** | This is our threat matrix of the different coding standards that I will be proposing and how likely they are vs. their priority to resolve. Each of these represents a different coding standard and represent a level of severity. As an example, Strings are in the likely category and the priority category. This is because Strings are likely to be misused and are a highly problematic source of exploitation.  Similarly, assertions are a less likely to be a problem if they are not used while they are also less of a priority as the other coding standards protect data much better than assertions do. |
| **4** | These are the 10 core principles of security. Each one has a different level of defense that they intend to protect. Furthermore, I have attached each coding standard that aligns with the given principle to further make the connection between the coding standards and the principles that I used to come to these conclusions.  Validating input data ensures that the data we are receiving is valid for use and as you can see this includes standards such as string correctness, SQL injection checking and assertions.  Heeding compiler warnings is the process of keeping an eye on what the compiler is telling you is wrong with your code. Architect and design for security policies is performed by following this presentation and using the information here to change how you create systems. Keeping it simple is to make things as simple as possible without reinventing the wheel, this helps with debugging and troubleshooting for where an error may be occurring.  Default Deny is the principle of denying access as the default course of action to prevent unneeded access to systems. Adhering to the principles of least privilege ensures that people only have as much access as they need to perform their tasks. Sanitizing data sent to other systems means that we are treating data very securely as it is sent from our system to other systems.  Practicing Defense in Depth is to provide different layers of security, such as employing this document for software design and then also employing a firewall to act as an additional layer against nefarious actors. Quality assurance techniques mean to do better coding inspection and write code that will maintain a higher level of quality such as using assertions.  Lastly, adopting a secure coding standard means to create policies for what type of code is valid in certain situations and what code should never be used. We will go into more detail regarding these in the next slide. |
| **5** | Here is the list of coding standards that I have created for this presentation, in order of highest priority. You can find more details about each of these in the Security Policy document as well but to describe a couple here, String Correctness is the first and foremost coding standard that should be addressed.  In this case, the use of C-Strings should be limited to specific scenarios where standard C++ strings cannot accomplish the given task. This is because C-Strings require a lot of careful consideration when being used and they can be easily exploited by attackers to do a lot of damage. Due to these reasons, String correctness is the number 1 priority when developing secure code.  On the other hand, for lower priority coding standards, we have examples such as the Order of Operations. With this coding standard I express the importance of making it clear with parentheses in what order you expect the code to operate. This is more important when you have long lines of code that are doing a lot of Boolean operations since the operations may have different levels of priority when they operate. Using parentheses will make the code more secure by ensuring it executes in the order you were expecting but also makes the code more readable. This, however, isn’t highly vulnerable to attacks and won’t cause a huge amount of damage per attack and is therefore not a high priority but should still be considered. |
| **6** | Encryption is just as important a policy as are the coding standards that I have created. In the case with encryption, we essentially need to have everything encrypted until it is not feasible to do so. Data at Rest must remain encrypted to avoid prying eyes or in the worst case a data leak. Data in flight should be encrypted in a reversable encryption that is nearly impossible to brute force. This way it can be used upon being received but would be impossible for a man-in-the-middle attack to take advantage of the data. Lastly, Data in Use should also remain encrypted until it needs to be used in a decrypted state. Some data can even be manipulated while still encrypted and this should be the preferred method when possible.  The easiest way to think about encryption is that everyone everywhere wants access to this data, including people working from within the company. There should be a very limited time frame of when the data is in it’s raw state. Furthermore, we should make sure we use modern cryptographic libraries. This goes back to keeping things simple and we shouldn’t reinvent the wheel when it comes to making data safe and secure. |
| **7** | The Triple-A policies involve the different overarching security principles that should be kept in mind when formulating a security plan. These include Authentication, Authorization, and Accounting. Authentication is the process of allowing access to a system, this usually includes a password and possibly a multifactor authentication.  Authorization is what someone is authorized to do when they are within the system. This includes read and write access to different files and the ability to add or delete existing users. Accounting is the process of tracking how users are using the given system. This includes a logging system which would log any modifications made by users.  Generally speaking, with these principles in mind you should adopt a few policies such as default deny, policy of least privilege and containing a logging system. Default deny is to deny access as the default and require certain steps to regain access when access has been locked out. A policy of least privilege is to only allow people as much access as they need to complete the job they were set out to achieve. Lastly, circling back on the logging system, this allows better tracking of all user transactions, allow for a better understanding of how attacks may have occurred to better deal with them in the future. |
| **8** | Now we get to the code testing side of security, in this case unit testing with Gtest. Gtest is a google created testing framework that is very simple to utilize and in this slide I show you the setup for a Gtest environment. For each test we will be calling an object that we will use to setup the environment and in this case we will be testing the standard C++ vector object. Here we have a few setup and clean up functions that reset and clear the vector respectively so our working set is as sanitary as possible before starting any tests. |
| **9** | Here is an example of a Gtest unit test where we are simply testing the appending function of the standard vector. As you can see, when creating the unit test we use the TEST\_F macro followed by the name of the object that will perform the setup tasks and the name of the test. Inside this macro block we define what the test is doing, in this case we are using other Gtest macros of ASSERT\_TRUE, ASSERT\_EQ, and ASSERT\_FALSE. We are asserting expectations for the data set, in this case to be empty, before modifying the data set. We add our single entry and then we are asserting that the data set is in-fact no longer empty and also assert that it is equal to the amount of entries we provided. |
| **10** | In this next test we are testing that the max size changes relative to the number of entries currently appended into the vector. |
| **11** | Next we are asserting that the size will become empty when we use the clear function for the vector. |
| **12** | Next we are testing how the capacity of the vector changes as we utilize the reserve function, ensuring that the reserve function is increasing the capacity. |
| **13** | In this next unit test we are utilizing a negative test case of the vector throwing an error. These are also important to test as your code should fail gracefully rather than causing the entire application to crash. In this case we are asserting that the vector will throw an error when attempting to access an element out of range. |
| **14** | Finally, we are testing another case of throwing an error, but in this case it is after the vector has been resized. |
| **15** | This is finally where automation comes into the picture. The process of DevSecOps is one that requires security to be taken into consideration at every stage. There are incredibly useful tools that can help with creating working test suites such as Gtest for creating unit tests and static code analysis for automated bug checking. Static code analysis is a very important part of testing software for quality as many of the working tools can detect many of the coding oversights that I have described in this presentation. Furthermore, static code analysis can find places in the code where optimization can be done to better improve performance. |
| **16** | Finally to reiterate, security must be part of the plan from the beginning, and this is where we can use static code analysis tools such as CPPCheck, Sonar, Cycode and most importantly developers.  There will still be issues that can be overlooked by static code analysis tools and these tools may even show false negatives. It is important that developers are involved at every step of the way to be reviewing what these tools are showing as issues and to look through the code thoroughly for any complex issues that may be missed by these tools. |
| **17** | This is now where we can discuss the risks and benefits of acting now vs acting later. The benefits of acting now are that security is worked in from the beginning. This means that the complexity of securing these projects is planned for and integrated as the project evolves, making adding other layers of security easier. Along with that there will be less security breaches from the beginning, especially as the more obvious security vulnerabilities are dealt with before they become an issue.  The risk of acting now will be that there will be more upfront cost, and our company may not be the most obvious target for a cyber threat. Integrating security measures now will require more man hours as there will be a need for more planning and for more development. Furthermore, if we are creating something simple that doesn’t house a lot of vital information then this may not be a needed cost immediately.  Transitioning into the benefit of waiting until later, the overall system will be in production much quicker without concern for security and will therefore cost less money upfront. On the other hand, the risks of waiting to integrate security may be highly more costly than the cost of designing with security from the beginning. With the heightened chance of security breaches, we may lose a lot of customer trust and therefore the cost of losing those customers due to the breach of their data. This could also lead to lawsuits and will require more development time as now these systems will have to be redesigned with the new forms of security. Developing the security at a later stage means that the system may need a lot of redesign as security can be as complex as the system itself to work well. |
| **18** | Overall, my recommendations would be to act now to build a better security policy and secure coding standards. There are many instances of large corporations that acted too late or were slow to respond to security concerns. In this case I will use Yahoo as they are one of the biggest cases of security breach in modern history.  Yahoo had been breached in 2013 due to insecure coding standards losing up to 1 billion accounts worth of user data. This includes a lot of personal information however even worse; Yahoo was slow to act against these insecure practices, meaning that up to 3 billion accounts were affected in total by the end of 2016. Let’s not be like Yahoo, let us take security into account at every step of the way and ensure that we are creating a system that customers will have trust in and create something that will not put our customers at risk of losing their valuable data. |
| **19** | In summary, it will always be best practice to generally follow the ideas of the security policies. Each of these encompasses a general attitude towards what type of code should be produced and how people should be treating access to our system. Furthermore, we should ensure that everyone is creating code that meets our standards for best coding practices. Security should never be left to the end of a project as this increases unneeded risks to the customers and the business.  Encryption and the Triple-A policies should also be adhered to as these will further create a defense-in-depth approach to solving security issues while keeping customer data as safe as possible. Along with all of these policies we should be employing the use of automated tools such as CPPCheck or Sonar to help identify any other security weaknesses. Lastly, we as developers should continue to learn and grow to be more aware of best practices when creating code and how to produce more secure code overall. |
| **20** | Here are the references for the presentation and thanks for listening. |