# CS 405 Project Two Script Template

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

| **Slide Number** | **Narrative** |
| --- | --- |
| **1** | Hello everyone, my name is Mitchell Lynds. I’ll be presenting on Green Pace’s new C/C++ security policy for software development, as well as some tools for vulnerability detection. |
| **2** | The security policy is a a collection of principles and standards to follow when developing projects in C/C++. Our team is growing and it is now necessary to standardize our security policies in order to ensure that everyone is producing secure code. Following the security policy will confirm that we are implementing security measures in multiple layers and correctly applying the concept of defense in depth. |
| **3** | The coding standards contained within the security policy document cover a range of threats. Threats are graded based on how likely they are to occur as well as how severe the consequences may be when they do. |
| **4** | Many of the vulnerabilities that the coding standards have been written to prevent may be detected automatically with the use of external testing tools. One such tool is demonstrated here. CPPCheck is a static testing tool capable of detecting several violations of the coding standards in this policy. |
| **5** | There are ten guiding principles in this initial version of the C++ security policy.    The first is to Validate input Data. This means that data coming into the system from external sources should be considered untrustworthy.  The second principle is that we need to Heed Compiler Warnings. Modern compilers are equipped to warn us ahead of time about many common vulnerabilities, eliminating these warnings early on will save us from expensive larger remediations later on down the line.  The third principle is to Architect and Design for Security. Sometimes putting a little thought towards structuring software in a way that lends itself to secure functioning can be a layer of security all on its own.  The fourth principle of the security policy is Keep it Simple. This principle is primarily about making code that is as understandable as possible. We are more likely to detect potential vulnerabilities if it is plain to see how the code works.  The fifth principle is called Default Deny. Default Deny refers to the granting of access permissions. The default case should be to deny access. In this way any unexpected behavior or invalid input should result in a denial of access.  The sixth core security principle is called “Adhere to the Principle of Least Privilege”. This principle is about limiting the potential impact of a potential breach. Privileges should be defined in a way that processes have only the privileges necessary to function.  The seventh principle is to Sanitize Data Sent to Other Systems. Data that is output from our systems should only be things that are absolutely necessary. For example, do not output an entire “users” data structure expecting the external system to only grant the user access to their own account details.  The next security principle is principle number eight which is to Practice Defense in Depth. Defense in Depth is the concept of layered security where we do not totally rely on any one security measure to protect the system, and it is a key guiding concept for Green Pace.  Security principle number nine is Use Effective Quality Assurance Techniques. This principle instructs us to apply a variety of proven tools for early detection of potential vulnerabilities. We will demonstrate and review the use of a few of these quality assurance techniques later on in the presentation.  The last security principle is called “Adopt a Secure Coding Standard”. Secure coding standards are a series of explicit rules regarding ways to avoid creating common security vulnerabilities. |
| **6** | The ten coding standards laid out in this security policy are:  Do not access freed memory  Guarantee that storage for strings has sufficient space for character data and the null terminator  Exclude user input from format strings  Prevent SQL injection  Ensure that unsigned integer operations do not wrap  Do not depend on the order of evaluation for side effects  Range check element access  Handle all Exceptions  Declare identifiers before using them  and Incorporate diagnostic tests using assertions  These standards are listed in order of priority based on Severity, Likelihood and remediation cost. With high severity, high likelihood vulnerabilities that are easy to fix listed first. |
| **7** | The security policy also lays out Green Pace’s policies for encryption. There are three types of encryption required by the security policy.  Encryption at rest is the encryption of data storage. This type of encryption is implemented by applying two-way encryption to files containing sensitive data that are stored in databases or on the cloud. This policy prevents data theft or leaks from revealing readable sensitive data. This policy applies anytime sensitive data is stored for later use or recordkeeping.  Encryption at flight is the encryption of data that is in the process of being transmitted on a network. Encryption at flight is implemented by using secure communication protocols for data transfer. It is to be used anytime sensitive data is transmitted over the internet. This policy prevents entities other than the intended destination from reading the data.  Encryption in use is the encryption of data that is in the process of being used by program code. This type of encryption is implemented by not storing sensitive data in memory in a readable form. If a piece of code takes sensitive data as an input from somewhere it should only be stored in an encrypted form. This type of encryption is used to prevent certain types of program crashes and exploits from revealing sensitive data. This policy is used when software needs to hold sensitive data as a variable. It should remain encrypted until it is used. |
| **8** | The security policy contains guidelines for implementation of a Triple-A framework. Triple-A consists of Authentication, Authorization, and Accounting.  The Authentication principle of the Triple-A framework is the principle that users should be properly authenticated before being granted access to data or performing sensitive tasks. This usually takes the form of a user log-in. This principle applies anytime the user is attempting to access sensitive data or tasks. This principle is used to prevent breaches of sensitive data from occurring.  Authorization is the principle that users should only be able to access data or actions that are appropriate to their role. Only certain users should be able to add new users. Authorization is how a user’s level of access is controlled. Authorization is usually implemented by defining user types and privileges that enable or disable certain actions. This principle is used to limit the damage caused by a potential breach as much as possible.  Accounting is the principle of tracking the actions performed within the system. This is implemented by logging the files accessed by users and changes made to the database. This principle is used so that breaches or attempted breaches can be identified and addressed. |
| **9** | Unit testing is an important tool for preventing vulnerabilities due to unexpected behavior. In the next few slides we will examine how unit tests can be used to protect against a common vulnerability in many data structures: accessing freed or out of range indexes. |
| **10** | The first test we perform tests that the methods for “collection” correctly reserves new memory without changing the size parameter of the collection which could allow access to uninitialized memory. |
| **11** | The next test proves that if the at() method is used to access an index larger than the size of the collection that an exception will be thrown rather than accessing uninitialized memory. This could be taken further with additional tests on other invalid inputs like negative numbers. |
| **12** | The third test is a negative test that confirms that attempts to resize the collection to sizes larger than the maximum size will cause an exception rather than allow the storage and access of values at invalid memory locations or causing overflows. This could be taken further by implementing test cases with other invalid values. |
| **13** | The last test confirms that shrink\_to\_fit() will not change the size of the collection inadvertently causing element’s memory to be freed unexpectedly. |
| **14** | The security policy includes guidelines on how to incorporate automation tools into Green Pace’s Dev Sec Ops cycle. |
| **15** | At the Design phase in the DevSecOps cycle automation tools like OWASP dependency check can be used to automatically scan for known vulnerabilities in the open source software that the project utilizes. Many of the secure code standards in this policy are checked for in a dependency check. This can influence design decisions about what tools to use or what measures must be taken if a tool with known vulnerabilities is used.  At the Build phase in the DevSecOps cycle automation tools can be used to prevent insecure code from ever being written. There are IDE extensions like SonarLint which automatically check and flag code for violations of many of the secure code standards in this policy while it is being written. When code is compiled, certain GCC security flags can be used to enable the compiler to automatically check for many of the secure code standards in this policy .  At the Verify and Test phase in the DevSecOps cycle automation tools can be used to identify violations of the secure code standards before they are pushed to production. Static code analysis tools like CPPCheck can be used to automatically identify violations of secure code standards. Dynamic code analysis tools can be used to automatically identify vulnerabilities only apparent at runtime as some of the secure code standards in this policy are. A unit testing framework like GoogleTest can be used to automate the writing and running of unit tests in order to check for unhandled error cases which would be a violation of this policy. A fuzz testing tool can also be used to automate the task of testing code with large amounts of input data which helps to ensure that input data is being sufficiently validated. This is one of the most important principles of this policy. |
| **16** | Acting now to correct existing violations of the secure coding standards could result in down time for the system as well as delays in delivering new features that are currently in progress. On the other hand the benefit of acting now on all violations is that it minimizes the possibility of a severe security breach.  We could also wait to act until it is convenient to the team. For example the next time that piece of code is being revised. The risk is that the vulnerability becomes a breach before it is corrected, while the benefit is that the system would stay up and we would stay on track for our upcoming deliverables. |
| **17** | I recommend that the Green Pace team immediately address high priority violations. That is, violations for which the likelihood and severity are high and the remediation cost is reasonable. Lower priority violations can wait until after the high priority fixes have been completed. This should allow most of the team to continue to work on upcoming deliverables and minimize system down-time, while addressing the most critical vulnerabilities. |
| **18** | The list of coding standards in this current version of the security policy is not exhaustive and the current list leaves several critical gaps in our security policy.  There are several other known vulnerabilities that are graded as P18 maximum priority which have not yet been included in this security policy. At a minimum These additional standards should be incorporated into the policy. |