**Green Pace Developer: Security Policy Guide Template: Justin Swinney**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | The validation of all user input to prevent injection via user input fields and other malicious intents. |
| 1. Heed Compiler Warnings | Compiling code at the highest warning level of the compile to quickly identify and eliminate known threats. |
| 1. Architect and Design for Security Policies | Building blocks of security policies and procedures to minimize the risk of vulnerabilities and form a secure minded development team. |
| 1. Keep It Simple | Complex designs lead to more vulnerabilities and mistakes. Small and simple designs allow for easy understanding within the development team and allow for any vulnerabilities to be caught quickly. |
| 1. Default Deny | Permission is needed to complete any task. |
| 1. Adhere to the Principle of Least Privilege | Principle of least privilege, in simple terms only give accesses to what is needed at any given time and remove these privileges when they are not needed. |
| 1. Sanitize Data Sent to Other Systems | All data should be sanitized and validated before sending to other systems. This will avoid attacks that may utilize unused functionality. |
| 1. Practice Defense in Depth | Implementing a multi layered security architecture that will create robust security system, this can include tech and real-world security of a facility. |
| 1. Use Effective Quality Assurance Techniques | Ensure competent QA teams are in place to find and uncover vulnerabilities that may be lurking in the software. |
| 1. Adopt a Secure Coding Standard | Selecting proper industry standard procedures based on the development language and or framework to ensure known vulnerabilities are avoided. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Correct usage of various data types to eliminate vulnerabilities and unwanted errors in relation to data, data size and data type. |

| **Noncompliant Code** |
| --- |
| Passing a number as an integer called max could potentially lead to errors due to the assignment of an integer in cases where the displayed data may be over the max value for an integer. |
| Void displayData(**int** max) {  for (int i = 0; i < max; i++) {  //Print list of data here  }  } |

| **Compliant Code** |
| --- |
| Usage of size\_t assigned to the variable max is compliant as it will handle dynamic values and avoid errors or unwanted issues. |
| Void displayData(**size\_t** max) {  for (int i = 0; i < max; i++) {  //Print list of data here  }  } |

| **Principles(s):**  **1. Validate Input Data:** Relevant if user input is needed for variable max to ensure validation of data type from user and prevent malicious inputs such as scripts from potentially being sent.  **2. Heed Compiler Warnings:** implementing fixes for potential issues that are not fetal to the program compiling can improve security and accuracy of the software, these warnings via IDE or a static tool analysis can provide details of implicit type conversions and allow for correction early in the development process.  **9. Use Effective Quality Assurance Techniques**: Implementing tools such as static code analysis can ensure these issues are discovered and correct during the development process.  **10. Adopt a Secure Coding Standard:** Applying this principle can ensure developers follow industries best practices for handling data, in the compliant code we see size\_t utilized over int to mitigate potential errors or unwanted behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | P2 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [sonarqube](https://www.sonarsource.com/products/sonarqube/) | 10.4 | SonarSource C/C++ analyzer | Overall static testing tool with automation can look for incorrect usage of data types and more |
| [Pvs-studio](https://pvs-studio.com/en/docs/warnings/v104/) | 7.29 | [V104](https://pvs-studio.com/en/docs/warnings/v104/) | implicit type conversion to memsize type |

#### 

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Prevention of vulnerabilities through proper data value handling and validation. |

| **Noncompliant Code** |
| --- |
| User enters an input to find an employee number this code is lacking data validation and proper handling for the user’s input. This can lead to unwanted behavior, errors, and potentially leaked data. |
| Void employeeNumberLoopUp(int userInput) {  Cout << userInput << “:” << “Employee Details here matching user Input” << endl;  } |

| **Compliant Code** |
| --- |
| Managing proper input validation for data values such as the employee number needing to be greater than one is a great way to avoid unwanted behavior and errors. |
| Void employeeNumberLoopUp(int userInput) {  If (userInput > 0) {  Cout << userInput << “:” << “Employee Details here matching user Input” << endl;  }  Else {  Cerr << “Employee Number must be greater than 0” << endl;  } |

| **Principles(s):**  **1. Validate Input Data**: Ensuring user input is validated such as the compliant code ensuring the user input is greater than zero helps prevent potential errors and vulnerabilities that could have been caused by incorrect user input and or malicious user input.  **5.Default Deny:** If validation of input fails accesses would be denied and the desired task at hand would not be complete, this could be permission-based access to a component of a program or other functionality.  **9.Use Effective Quality Assurance Techniques:** Implementing static analysis tools can assist in identifying such issues and vulnerabilities as underflow overflow errors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | **L2** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023b | CERT C: Rule INT32-C | Integer overflow |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.29 | V1026, V1070, V1081, V1083, V1085, V5010 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Utilizing string correctly can help prevent errors and vulnerabilities such as Buffer overflows or injection attacks. |

| **Noncompliant Code** |
| --- |
| Usage of sprint along with too small of a buffer size could lead to overflows and other errors or vulnerabilities. |
| Void stringStructure(char \* buffer, int number) {  **sprintf**(buffer, number);  Cout << “string : “ << buffer << end; |

| **Compliant Code** |
| --- |
| Usage of snprintf along with size\_t bufferSize can remove the potential risk of buffer overflows and ensure proper string formatting (correctness) |
| Void stringStructure(char\* buffer, size\_t bufferSize, int number) {  **snprintf**(buffer, bufferSize, value);  cout << “string:” << buffer << endl; |

| **Principles(s):**  **1. Validate Input Data:** Ensuring input validation like the buffer Size can prevent overflows.  **9. Use Effective Quality Assurance Techniques:** Implementation of static tools can find and notify developers of this potential issue related to buffer overflows as well as string correctness contributing to better readability and security. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Medium | P2 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPcheck | Newest | General static analysis |  |
| Clang Tidy | Newest | General static analysis |  |
| Coverity | Newest | General static analysis |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Standard for providing secure queries and prevent SQL injections |

| **Noncompliant Code** |
| --- |
| Usage of direct concat string to construct the query string can lead to SQL injection attacks as the user’s input could be manipulated to run SQL commands. |
| String userInput = “test’; DROP TABLE items; =”;  String query = “SELECT \* FROM table WHERE items = ‘” + userInput +”’;”; |

| **Compliant Code** |
| --- |
| The user’s input is separated from the query call and passed to it as a parameter utilizing a placeholder. |
| String userInput = “test’; DROP TABLE items; =”;  MYSQL\_STMT\* sx = mysql\_stmt\_init(connection);  Mysql\_stmt\_prepare(sx, “SEKECT \* FROM table WHERE items = ?”, 11); |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):**  **1. Validate Input Data:** Separation of user input and parameterized queries will help prevent possible SQL injection attacks by ensuring user input is validated prior to queries being run.  **5. Default Deny:** Utilizes of Default deny will ensure that if input validation has failed to deny access to communicating to the data storage system in place therefore preventing unauthorized access and or potential injection attacks.  **10. Adopt a Secure Coding Standard:** Utilization of parameterized queries to prevent potential SQL injection attacks follow closely with this standard and provides reassurance that this known vulnerability is mitigated.  **7. Sanitize Data Sent to Other Systems:** Proper sanitization of the user input prior to sending to SQL database can prevent attacks. This is relevant but not 100% included in the example. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2023.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 9.9 | S2077  S3649 | Executing SQL queries is security-sensitive |
| [SpotBugs](https://wiki.sei.cmu.edu/confluence/display/java/SpotBugs) | 4.6.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE  SQL\_PREPARED\_STATEMENT\_ | Implemented |

#### 

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Prevention of memory related vulnerabilities following secure memory handling to prevent these vulnerabilities. |

| **Noncompliant Code** |
| --- |
| This method calls for the freeing of memory by deleting an array. However, no validation or conditional statements are provided which can lead to deletion of non-null pointers and unwater program actions. |
| Void freeMemory(int\* anArray) {  Delete[] anArray;  } |

| **Compliant Code** |
| --- |
| Implementing a conditional statement that implements a null pointer check before deletion of the dynamic memory can help reduce the possibility of unwanted behavior as well as overflows. |
| Void freeMemory(int\* anArray) {  If (anArray != nullptr) {  Delete[] anArray;  anArray = nullptr;  } |

| **Principles(s):**  **5. Default Deny:** The compliant code only allows memory to be freed if it is not null therefore adhering to the default deny, in doing so this makes sure that it only frees memory when safe and prevents access if not.  **9. Use Effective Quality Assurance Techniques**: Utilization of static analysis tools like Parasoft C/C++test will assist in discovering memory related vulnerabilities and errors and allow for proper correction prior to production. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | [CERT C++: MEM50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem50cpp.html) | Pointer access out of bounds, Deallocation of previously deallocated pointer |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 483 S, 484 S | Partially implemented |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | dangling\_pointer\_use | Invalid pointers accessed |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.0p0 | ALLOC.UAF | Use after free |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Implementing assertions and assisting in catching errors within the development process. |

| **Noncompliant Code** |
| --- |
| No validation of deposit amount the user enters therefore if less than 0 is entered potential issues could occur. |
| Void atmDeposit(int& balance, int totalDepo) {  Balance += totalDepo;  } |

| **Compliant Code** |
| --- |
| Inserting Assert within the function allows for a condition to be met and a verification process to be met before updating balance of the users account. |
| Void atmDeposit(int& balance, int totalDepo) {  Assert(totalDepo >= 0);  Balance += totalDepo;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):**  **1. Validate Input Data**: In the case of the example the total deposit could be considered a user input and proper validation of this could be included on top of the compliant code to increase overall error handling and security.  **8. Practice Defense in Depth**: Usage of assert in the compliant code allows an additional layer of security to prevent errors or possible vulnerabilities during runtime by adding a check such as ensuring the deposited is not negative prior to adding the deposit to the account balance, preventing calculation errors.  **9. Use Effective Quality Assurance Techniques:** usage of static analysis tools can assist in discovering bugs and potential errors that are not handled properly. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | High | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023b | [CERT C: Rec. ERR00-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.err00c.html) | Checks for situations where error information is not checked |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Ensure correct usage of exceptions, this standard can help prevent numerous errors and vulnerabilities early in the development process. |

| **Noncompliant Code** |
| --- |
| Although this code will catch all exceptions due to the catch(all) represented by 3 dots, this is non-compliant as identifying specific issues or potential vulnerabilities will be difficult. Specified catches should be implemented. |
| Try {  Test = 3 \* 0 / 12;  } catch (…) {  } |

| **Compliant Code** |
| --- |
| Usage of catch (specific error) and displaying the exception allows for quick and accurate troubleshooting and identifying the vulnerability or error. |
| Try {  Test = 3 \* 0 / 12;  } catch (const runtime\_error& e) {  Cerr << e.what() << endl; // display specific exception error.  } |

| **Principles(s):**  **8. Practice Defense in Depth:** In the compliant code the proper implementation of specific exception handling and displaying these errors not only provides layers of defense by catching potential errors without causing crashes but allows for a developer to quickly identify the issue at hand and correct it.  **9. Use Effective Quality Assurance Techniques:** Utilization of static analysis tools can be used for discovering and mitigating these issues regarding improper handling. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | High | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023b | [CERT C: Rec. ERR00-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.err00c.html) | Checks for situations where error information is not checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Smart Pointers** | STD-008-CPP | Proper and recommended usage of smart pointers. |

| **Noncompliant Code** |
| --- |
| Usage of raw pointer for dynamic memory could be used with smart pointer for automation. |
| Int\* arrayCreation() {  Int\* array = new int[5];  Return array;  } |

| **Compliant Code** |
| --- |
| The usage of unique\_ptr automates deallocation of memory. |
| Unique\_ptr<int[]> arrayCreation() {  Auto array = make\_unique<int[]>(5);  Return array;  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):**  **4. Keep It Simple:** utilization of smart pointers ensures for a simplified memory management process and can reduce errors and vulnerabilities related to memory deallocation.  **9. Use Effective Quality Assurance Techniques:** Static analysis tools can check and verify improper usage of pointers and allow for corrections early in the development process. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Medium | P4 | P2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPcheck | 2.13 | General static analysis |  |
| Clang Tidy | 19.0.0 | General static analysis |  |

#### 

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Standard Library Usage** | STD-009-CPP | Ensuring proper usage of C++ standard library. |

| **Noncompliant Code** |
| --- |
| Not utilizing the standard library by manually placing pi and multiplication of X twice. |
| Double calcAreaOfCircle(double x) {  return 3.14 \* x \* x;  } |

| **Compliant Code** |
| --- |
| Usage of M\_pi to represent pi accurately and pow to calculate the diameter of the circle. This can provide accurate calculation unlike the non-compliant code above. |
| Double calcAreaOfCircle(double x) {  return M\_PI \* pow(x, 2);  } |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):**  **1. Validate Input Data:** Proper validation of x could be considered if it is an input therefore this principle could apply on a situational basis, ensuring that x is indeed a double value to increase accuracy of outputs as well as prevent potential errors.  **4. Keep It Simple**: Usage of standard C++ libraires allow for simplification and readability of code to improve by reducing the need to implement more variables therefore adhering to the keep it simple principle. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang Tidy](https://clang.llvm.org/extra/clang-tidy/checks/readability/identifier-naming.html) | 19.0.0 | General static test |  |
| CPPcheck | 2.13 | General static test |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Naming Convention** | STD-010-CPP | Keeping consistent naming conventions to maintain readability across the entire application. |

| **Noncompliant Code** |
| --- |
| Camel case used instead of snake case as well as lowercase start to the class name. |
| Double aNumber = 1;  Class testClass {} |

| **Compliant Code** |
| --- |
| Proper formatting of variables names to follow C++ standards and maintain readability across multiple developments within an application. |
| Double a\_number = 1;  Class TestClass {} |

**Note: Stop here for the milestone. Complete this section for Project One in Module Six.**

| **Principles(s):**  **4. Keep It Simple:** Utilizing a languages standard naming convention allows for improved readability and consistency adhering to the keep it simple principle in this example we see inconsistences of class names and variables names not adhering to C++ naming convention.  **9. Use Effective Quality Assurance Technique:** Utilizing static analysis tools like code sonar can detect consistencies in naming conventions and warn developers of the decreased readability. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang Tidy](https://clang.llvm.org/extra/clang-tidy/checks/readability/identifier-naming.html) | 19.0.0 | readability-identifier-naming | Enforce coding guidelines. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.0p0 | LANG.ID.AMBIG | Typographically ambiguous identifiers |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | CERT\_C-DCL02-a | Use visually distinct identifiers |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Green has a well established DevOps process and infrastructure in place currently, Automation should be implemented in the following, starting with Access and plan – Once a chosen framework, language associated and other libraries are chosen create a list for automation tools that can or could be utilized, next is the design, build, verify and test stages, these three stages can utilizes tools static code analysis tools to automate and test code as it is being developed as well as automatically scanning the framework for potential security risk and vulnerabilities. Implementing Automatic monitoring of production code can allow for consistent checks for vulnerabilities new and existing as well as potential errors and or bugs. Some automation tools can allow for procedures or guidelines to be delivered to the developer upon an automated scan finding an issue, this allows for prompt action and allows the developer to have a guide while finding and correcting the issue. And finally, the Maintain component of the DevOps process can implement automated patching systems to ensure dependencies and other software related items are updated to the current versions although this can introduce potential errors it can be configured per application to ensure error prone transitioning from versions.

### Summary of Risk Assessments

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Likely | Low | P2 | 2 |
| STD-002-CPP | High | Likely | High | P9 | 2 |
| STD-003-CPP | Medium | Likely | Medium | P2 | 2 |
| STD-004-CPP | High | Likely | Medium | P18 | 1 |
| STD-005-CPP | High | Likely | Medium | P18 | 1 |
| STD-006-CPP | Medium | Probable | High | P4 | 3 |
| STD-007-CPP | Medium | Probable | High | P4 | 3 |
| STD-008-CPP | Medium | Likely | Medium | P4 | 2 |
| STD-009-CPP | Low | Unlikely | Low | P3 | 3 |
| STD-010-CPP | Low | Unlikely | Medium | P2 | 3 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest involves encrypting data while stored in the cloud, a database or alternative file storage server, in simpler terms the data is not moving. Utilizing a robust encryption algorithm and key management system allows for sensitive data to be unreadable in the case of a security breach, a strong algorithm converts the readable data into various characters and numbers depending on the implementation preventing unwanted guest from being able to decipher or decrypt the information without having a decryption key. |
| Encryption at flight | Encryption at flight or data on the move from point a to point b involves encrypting the data as it is transferred over the internet, this policy is crucial to preventing an unwanted individual from intercepting sensitive data, this policy can mandate the use of encryption protocols like SSL to all communication components such as API where data is sent from a backend to front end or vise versa. |
| Encryption in use | Encryption in use refers to encrypting data while the user is accessing it, for example a user is on a client page where client information is displayed from a database to the user and the user can modify client information and save this information back to a database. This policy can enforce encrypting data while it is locally stored on the users end and any part of the processing pipeline prior to being sent back to encryption at flight. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication ensures that a user attempting to access the software is verified and allowed or declined access to the data or software while the user logins. This policy can include multi-factor authentication and face recognition. This policy can go beyond the scope of allowing a user into the program and can be implemented to API access. |
| Authorization | Authorization can prevent access to users once authentication is successful, this can include implementing a role-based system where users have different views within a program and different access to data based on roles. Utilizing the principle of least privilege to only provide the needed accesses at that time can reduce the potential for security issues and data security. While creating a new user or allowing the creation of a new user from the user end implementing the least privilege to them unless otherwise specified is a good start to implementing the principle of least privilege. |
| Accounting | Accounting can track and save a user’s “paper trial”, any user interactions can be logged and tracked to be reviewed or monitored for suspicious activity, this can include a user changing data information or the user access components of the software. This policy can enforce the maintenance of these logs and is a crucial component to investigating issues that occurred, this policy can also follow compliance standards that may be in place. |

### Map the Principles

Map the principles to each of the standards and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 01/26/2024 | Coding Standards | Justin Swinney |  |
| 3.0 | 02/18/2024 | Completed Project | Justin Swinney |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

| Language | Acronym |
| --- | --- |
| C++ | CPP |
| C | CLG |
| Java | JAV |