**Brianna Marsh’s Green Pace Developer: Security Policy Guide Template**



# 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 | Validating the input data checks the accuracy and integrity of input data, prior to it being processed. This is an essential tool that helps prevent security vulnerabilities. This also enhances security for the overall application. |
| 1. Heed Compiler Warnings | Paying attention to compiler warnings is imperative to ensuring that vulnerabilities don’t happen. It is easy to ignore them, but not paying attention and addressing them could lead to a breech in your code. Ignoring the compiler warnings could also lead to malicious code getting in. |
| 1. Architect and Design for Security Policies | This creates a complex framework that sets guidelines that dictates exactly how the security should be enforced. In a nutshell, this incorporates security into your application. Overall, this makes your coding environment more secure. |
| 1. Keep It Simple | The less lines of code that your application has, the less areas where hackers are able to implement their malicious code. It is also easier for a developer to spot issues in the code, being that it is simple. The more complex your code is, the more area where it leaves for bugs and unnecessary incidents to get in. |
| 1. Default Deny | This principle is very important because it does not just allow access to all and anyone who wants access to your code; Yet instead, this principle allows access to necessary parties. Default deny keeps out the unauthorized access to your applications. |
| 1. Adhere to the Principle of Least Privilege | This principle ensures that some users should be given minimal access and or permission to view, read, or change tasks within the application. Default deny can be apart of this process as well. This process helps protect sensitive data as well as helping to avoid security breaches. |
| 1. Sanitize Data Sent to Other Systems | The process of this principle involves cleaning the code prior to processing it and sending it out. This helps to ensure that there is no malicious code or vulnerabilities prior to processing it. Therefore, this prevents forms of data manipulation. |
| 1. Practice Defense in Depth | This is a strategy that has multiple layers of security to protect the security health of an organization. This could sometimes mean giving organization users little access to their system. This is something that is done normally. |
| 1. Use Effective Quality Assurance Techniques | This ensures that the current software meets organizational requirements and is free of vulnerabilities. In return, this makes a system more reliable, being free of vulnerabilities. This also helps with ensuring the compliance of the code. |
| 1. Adopt a Secure Coding Standard | This is a standard where every developer follows to follow all of the rules and guidelines within an organization. Doing this adds another layer of security to ensure that the system is free from vulnerabilities. This, in return, can also stop hackers as well. |

### 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::string userInput;** | [STD-001-CPP] | This specific data type represents the validation of user input. For this I used an example of validating a string of user input. Within my label, the number part is “001” since it is the first principle and “CPP” represents the C++ language. |

| **Noncompliant Code** |
| --- |
| The example code down below, in the next block is punctuated all wrong. This code is telling us that when the user puts in the Correct userinput, that the system is to clear out the input and say that it was invalid. This is incorrect because if the user puts in the correct input, then it is valid. |
| STRING UserInPUT;  COUT << “Please Place your string in here” << endl;  While(cin >> UserInPUT)  {  CIN.Clear();  COUT << “Invalid input” << endl;  } |

| **Compliant Code** |
| --- |
| This block of code, down below, is compliant not only due to the punctuation being correct, but also because of the clarity of direction. This code lets our user know to clearly enter in your input. Once the user enters in anything other than the correct userInput, then the system will clear out and print out an error code that states “Invalid input”. |
| std::string userInput;  std::cout << “Please Place your string in here” << std::endl;  while(!(std::cin >> userInput))  {  std::cin.clear();  std::cout << “Invalid Input” << std::endl;  } |

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

| **Principles(s):** The principle is validate input data. With the coding standard being the to validate the string input, this is important since the user input needs to be validated to compile. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.MEM.BO | Buffer overrun, https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access |
| CodeSonar | 8.0p0 | LANG.MEM.BU | Buffer underrun, https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access |
| Code Sonar | 8.0p0 | IO.IOWOP | Input After Output, https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO50-CPP.+Do+not+alternately+input+and+output+from+a+file+stream+without+an+intervening+positioning+call |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value:**  **Very broad** | [STD-002-CPP] | In general, these principals let you know potential bugs, threats, or issues that could occur |

| **Noncompliant Code** |
| --- |
| This code is not popping up an error message which is not compiler by letting the compiler know about the potential threat. |
| int main() {  int t;  cout << t << “Print please” << endl;  return 0;  } |

| **Compliant Code** |
| --- |
| The warning it telling the compiler that ‘t’ has not been initiated yet. This could lead to an error and/or bug. |
| int main() {  int t; // ‘t’ is not initialized in this function  std::cout << t << “Print please” << std::endl;  return 0;  } |

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

| **Principles(s):** The principle is Heed Compiler Warnings and it relates to the coding standard since the coding standard relates to to bugs and warnings. The principle allows you to see the warnings. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.MEM.BO | Buffer overrun |
| CodeSonar | 8.0p0 | LANG.MEM.BU | Buffer underrun |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness:**  **std::string** | [STD-003-CPP] | If your string is not correct, this could lead to vulnerabilities, overall bugs and unwanted issues. |

| **Noncompliant Code** |
| --- |
| “std” is not added to the “string” or “cout” statements, which can leave room for errors. |
| #include <string>  int main() {  string bestString = “The Best string Ever!!”;  cout << bestString << endl;  return 0; |

| **Compliant Code** |
| --- |
| The string is being secured by “std”, as this avoids buffer overflows and errors. |
| #include <string>  int main() {  std::string bestString = “The Best string Ever!!”;  std::cout << bestString << std::endl;  return 0; |

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

| **Principles(s):** The principle is Architect and Design for Security Policies and it relates to the coding standard because you want to make sure that your strings are correct so you won’t have any vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft | 2023.1 | CERT\_CPP-STR53-a | Guarantees that containers are valid and within range, https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access |
| Polyspace Bug Finder | R2023b | CERT C++:STR53-CPP | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered,  https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | You must validate the user input to avoid SQL injection vulnerabilities. |

| **Noncompliant Code** |
| --- |
| This is demonstrating a 1=1, which means it will only spot a malicious code if there is a one to one function, but it won’t catch all of the other malicious code that may come through. |
| records.clear();  std::cout << sql.c\_str();  std::string subStringg = "1" = "1"; // spotted Malicious code  std::cout << "Malicious code" << std::endl; |

| **Compliant Code** |
| --- |
| This is a snippet of an SQL injection code that I did which displays if there is a sql injection code. This will output a Malicious code if there is a malicious code suspected. |
| records.clear();  std::cout << sql.c\_str();  std::string subStringg = """ = """; // spotted Malicious code  std::cout << "Malicious code" << std::endl; // letting the user know that there was Malicious Code that was spotted  char\* error\_message;  if (sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  } |

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

| **Principles(s):** The Principle is keeping it simple, and this relates to SQL Injection by validating the user input which is a simple process, in order to avoid the SQL Injection. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.MEM.BO | Buffer overrun |
| CodeSonar | 8.0p0 | LANG.MEM.BU | Buffer underrun |
| CodeSonar | 8.0p0 | LANG.MEM.TO | Type overrun |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Memory protection prevents most memory issues like leaks. |

| **Noncompliant Code** |
| --- |
| This is not the correct usage of ptr. |
| #include <memory>  Int main() {  ptr<int> smartPtr = shared<int>(37);  Return 0; |

| **Compliant Code** |
| --- |
| “std::shared\_ptr” manages memory |
| #include <memory>  Int main() {  Std::shared\_ptr<int> smartPtr = std::shared<int>(37);  Return 0; |

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

| **Principles(s):** The principle is default deny and it relates to this coding standard because they are both preventing either access or memory issues to protect the code |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | V7.5.0 | USE\_AFTER\_FREE | Able to detect the specific instances where memory is deallocatedmore than one time or read or written to the targer of freed pointer, https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Assertions check conditions that should always be true while the program is being executed. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| [Noncompliant code block; code should be indented using 12-point Courier New font.] |

| **Compliant Code** |
| --- |
| This section of code states that as long as the denominator is 0, you cannot divide by it the denominator; This will always be true. |
| #include <cassert>  Int divide(int numerator, int denominator) {  assert(denominator == 0);  return numerator / denominator;  std::cout << “Error, you cannot divide by 0” << std::end; |

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

| **Principles(s):** The principle is Adhere to the Principle of Least Privilege. This principle relates to the coding standard by checking the conditions, through assertions, as far as who has access to see what the code is outputting. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Default-new-overaligned-type | Fully Checked, https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM57-CPP.+Avoid+using+default+operator+new+for+over-aligned+types |
| RuleChecker | 22.10 | Default-new-overaligned-type | Fully checked, https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM57-CPP.+Avoid+using+default+operator+new+for+over-aligned+types. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Sometimes the program will throw an exception statement so the calling code can catch it in order to handle an error. |

| **Noncompliant Code** |
| --- |
| The exception is not being caught towards the end of the code block. |
| int main() {  try {  int result = divide(20, 10);  std::cout << “Answer: “ << result << std::endl; |

| **Compliant Code** |
| --- |
| This code uses the try, catch method to catch all of the exceptions. |
| int main() {  try {  int result = divide(20, 10);  std::cout << “Answer: “ << result << std::endl;  }  catch(const std::exception& e) {  //catches the exception  } |

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

| **Principles(s):** The principle is Sanitize Data Sent to Other Systems. This principle relates to the coding standard because both are dealing with handling errors, one by cleaning the code prior to submission and the standard by handling an error by throwing and catching. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Medium | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Exception-caught-by-earlier-handler | Fully checked, https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR54-CPP.+Catch+handlers+should+order+their+parameter+types+from+most+derived+to+least+derived |
| CodeSonar | 8.0p0 | LANG.STRUCT.UCTCH | Uncreachable Catch, https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR54-CPP.+Catch+handlers+should+order+their+parameter+types+from+most+derived+to+least+derived |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-ERR54-a | Multiple handlers are provided in a single try-catch statement or function for a try block for a derived class, https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR54-CPP.+Catch+handlers+should+order+their+parameter+types+from+most+derived+to+least+derived |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data Type:  long | [STD-008-CPP] | “long” data type is an integer type for longer integers |

| **Noncompliant Code** |
| --- |
| This is non-compliant because the integer type is not long and less than 8 bytes. |
| Long population = 10L |

| **Compliant Code** |
| --- |
| This is a long integer type that is longer than 8 bytes. |
| Long population = 1000000000000000L |

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

| **Principles(s):** The principle is Practice Defense in Depth. The principle relates to the standard since you sometimes need long (int) data types to implement an extra layer of security. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | ALLOC.LEAK | Prevent Leaks, https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR56-CPP.+Guarantee+exception+safety |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data Type: short | [STD-009-CPP] | This data type represents integers that are short, around 2 bytes. |

| **Noncompliant Code** |
| --- |
| This block of code is non-compliant because of the the integer represents a long integer data type. |
| short population = 320000000L; |

| **Compliant Code** |
| --- |
| This snippet of example code is compliant because it is and integer with 2 bites |
| short population = 32; |

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

| **Principles(s):** The principle is Use Effective Quality Assurance Techniques. This principle relates to the coding standard because it helps having short data types when ensuring the quality of the code is compliant with the organizational regulations. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2023b | CERT C++:STR50-CPP | Checks for the use of dangersous standard functions, missing null in strings arrays, buffer overflow, insufficient destination buffer size, rule partially covered, https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Arrays:  Integers | [STD-010-CPP] | This is a list (array) of numbers (integers). |

| **Noncompliant Code** |
| --- |
| This block of code is noncompliant because the beginning of the list states that there needs to be 6 numbers, but instead, it is displaying only 3 numbers. |
| int list[6] = {1, 5, 7}; |

| **Compliant Code** |
| --- |
| This block of code represents an array list of numbers that is punctuated properly. |
| int list[6] = {1, 2, 3, 4, 5, 7}; |

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

| **Principles(s):** The principle is Adopt a Secure Coding Standard. This principle relates to the coding standard by implementing integers to making sure the coding standard is met. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2023b | CERT C++:STR50-CPP | Checks for the use of dangersous standard functions, missing null in strings arrays, buffer overflow, insufficient destination buffer size, rule partially covered, https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

[Insert your written explanations here.]

With Automation, you access and plan the production. After this phase, you then design and build the production. You must verify and test, prior to the DevSecOps portion. After this phase you have been cleared to deploy the production while monitoring it. During this phase and process you are ensuring that you are blocking attacks and stabilizing the production, all while ensuring that you are using your security measures throughout.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Likely | High | High | 5 |
| STD-002-CPP | High | Likely | Medium | Medium | 2 |
| STD-003-CPP | High | Likely | High | Medium | 2 |
| STD-004-CPP | High | Likely | High | High | 3 |
| STD-005-CPP | High | Likely | High | High | 5 |
| STD-006-CPP | Medium | Unlikely | Medium | Medium | 2 |
| STD-007-CPP | Medium | Likely | Medium | Medium | 3 |
| STD-008-CPP | High | Likely | High | High | 5 |
| STD-009-CPP | High | Likely | High | High | 5 |
| STD-010-CPP | Medium | Unlikely | Medium | Low | 2 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 is data stored within a system, in which this system protects the secrecy of the data. This policy applies because when dealing with and handling security, you must ensure that all files are secure and not accessible to anyone who does not have access to it. |
| Encryption at flight | Encryption at flight is the process that the encryption takes on being made. This is the actual coding process when implementing the encryptions, which is very important and needed to secure code. |
| Encryption in use | Encryption in use is the encrypted code actually being is use. This is the tester to ensure that all of the encryption has been done properly. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | With the authentication, the user must login and add new users if needbe. |
| Authorization | Authorizing the users accounts and files so that the user may have access to it. Also any updates or changes to the database may be done at this time as well. |
| Accounting | The accounting is where the user has been authorized and now have access to their files. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### 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 a 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 |  |
| 1.0 | 12/17/2023 | Module 3-2 Milestone | Brianna Marsh | [Insert text.] |
| 1.0 | 12/22/2023 | Module 6 Project 1 | Brianna Marsh | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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