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

Corley Adams cs 405 Professor Alhweiti



# 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 input data is considered to be the main strategy used in eliminating attack by malware in software applications. This procedure entails approving information before its submission or processed within the system. It is also crucial to block malicious information inputs as soon as it is obtained from the system. To have a flagging system setup would be one of the ways to solve this issue, they will analyze suspicious data and test them to certify whether the information will be harmless or not. There are many different types of input data that could be used to attack the system, but they are known to be primarily the fault of users untrusted external sources. On the other hand, the challenge is in preventing the invalid data from the unreliable sources. malicious actors will also certainly form part of measures taken to secure the whole system. |
| 1. Heed Compiler Warnings | The warning statements are of equal value and equally important for developers to read, detect and act on them, if needed. It is vital to perform this practice, because it helps developer see arising issues in early stages so that they do not rankle due to improper handling of warnings. The responsibility that comes with the suggestions as an outcome may not carry the burden of faulty code, but when they are not reviewed, they may result in clash of interoperability. Deciding on the right approach is the largest step skiing coders must take; which by the way will need energy & money to do so. This type of warnings is shown when code is compiled and in such case these warnings should be activated to make sure that the program runs correctly. Examples of the techniques used to produce compiler warnings that can stop the program when it discovers an error through dynamic and static analysis methods. |
| 1. Architect and Design for Security Policies | When deciding the architecture of the application, it is crucial to include security policies. Failing to consider and implement any security policies until the end can outcome to unforeseen issues and restructuring the majority of the project. Including security policies in each stage of the architecture design will lead to the development of a strong secure project. |
| 1. Keep It Simple | The complexity of a system has a straight effect on the legibility, maintainability and performance. Keeping a system straightforward and simple with the comments in the code will enhance the overall end results of the system and decrease the formation of mistakes like errors and bugs. |
| 1. Default Deny | It is highly important and valuable to have default deny as a security measure in the system. Default deny helps system security by denying access by default to the system database user accounts, unless a proper authentication is presented. Denying users from accessing valuable information in the system will secure the whole program from users that have malicious determination. |
| 1. Adhere to the Principle of Least Privilege | Reducing the accessibility of users in the system can stop the program from being exploited. This method does not only mitigate risks, but also contributes to accountability. Decreasing the accessibility to a handful of workers that need credentials to access the system enables a track record of what changes were made and who entered the system. Usage of this method is crucial in reducing risks and to maintain a track record of changes made. |
| 1. Sanitize Data Sent to Other Systems | Validating and checking data that is sent to external sources will stop malicious or compromised data from being transmitted. Making sure that data is checked and validated to be free from malware, will stop infecting external system. The procedure is important to maintain a trustworthy reputation. |
| 1. Practice Defense in Depth | Creating multiple layers of security to mitigate and stop any malware from accessing the system is referred to as defense in depth. Integrating this method will create layers of security as a safety net after the first layer has been bypassed. It is critical to have different layers of security to contain a sell secured system, but there are also some critical aspects to consider when using this method because it can get complicated to incorporate and maintain over time since it deals with different types of security layers that are different from each other. |
| 1. Use Effective Quality Assurance Techniques | Using effective quality assurance methods outcomes in improvements, testing, accountability, and delivering the best results. Implementing this method can be done in various ways, but the main practice to deliver the best outcomes are directly associated with testing and enhancing previous or current work. Testing code from user and developer stand point to gain performance and use experience information to enhance the system will deliver a solid well grounded project. |
| 1. Adopt a Secure Coding Standard | Integrating secure coding standards is desirable and applicable in every project. Without this practice, vulnerabilities would be left unattended which will lead to compromised data and system extortion. Some of the best practices to secure coding standards include applying coding consistency, vulnerability prevention, code review, and constant training. These standards may vary slightly depending on what company’s value, but almost all are composed of similar traits that all correlate with securing the system. |

### 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-005-MPR] | Safe Memory Management |

| **Noncompliant Code** |
| --- |
| The following code dynamically allocates memory for an integer array but fails to free it, leading to a memory leak. |
| int\* createArray(int size) {  int\* arr = (int\*)malloc(size \* sizeof(int));  // Initialization or usage of arr  return arr; // Memory allocated to arr is never freed  } |

| **Compliant Code** |
| --- |
| This version correctly frees the allocated memory once it is no longer needed, preventing a memory leak. |
| int\* createArray(int size) {  int\* arr = (int\*)malloc(size \* sizeof(int));  // Initialization or usage of arr  free(arr); // Memory is freed  return NULL;  } |

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

| **Principles(s):** Resource Management - Properly managing resources like memory is fundamental to preventing leaks and ensuring the efficient use of system resources. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Low | High | Critical |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.15.0 | Memcheck | Valgrind's memcheck tool can automatically detect memory management issues such as leaks, use of uninitialized memory, and improper freeing of memory. |
| Clang Static Analyzer | 10.0.1 | alpha.unix.cstring.BufferOverlap | The Clang Static Analyzer includes checks for potential buffer overflows and other memory-related issues, aiding in early detection during development. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-INV] | Robust Input Validation |

| **Noncompliant Code** |
| --- |
| This code directly uses external input without validation, potentially leading to buffer overflow if the input exceeds expected size. |
| void processInput() {  char buffer[256];  gets(buffer); // Unsafe use of gets() which does not check input size  // Process buffer  } |

| **Compliant Code** |
| --- |
| This example uses fgets() instead of gets(), limiting the number of characters read to prevent overflow. It also checks the input against expected conditions. |
| void processInput() {  char buffer[256];  if (fgets(buffer, sizeof(buffer), stdin) != NULL) {  // Validate input (e.g., length, format, etc.)  // Process buffer  } else {  // Handle reading error  }  } |

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

| **Principles(s):** Security - Robust input validation is a security cornerstone, preventing attackers from exploiting unexpected behavior through malicious input. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Medium | High | Critical |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2021.12 | TAINTED\_STRING | Coverity statically analyzes code to identify potential security vulnerabilities, including improper input validation, offering suggestions for mitigation. |
| SonarQube | 8.9LTS | S3649 | SonarQube analyzes code to detect security vulnerabilities, including those arising from unvalidated inputs, and suggests secure coding practices. |
| [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-003-STR] | Safe String Handling |

| **Noncompliant Code** |
| --- |
| The following code uses ‘strcpy’, which does not check for buffer overflow, potentially leading to a security vulnerability. |
| void copyString(char\* dest) {  char src[10] = "Overflow!";  strcpy(dest, src); // Potential buffer overflow  } |

| **Compliant Code** |
| --- |
| This version uses ‘strncpy’ to limit the number of copied characters, preventing buffer overflow. |
| void copyString(char\* dest, size\_t destSize) {  char src[10] = "Safe";  strncpy(dest, src, destSize - 1); // Prevents buffer overflow  dest[destSize - 1] = '\0'; // Ensures null termination  } |

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

| **Principles(s):** Secure Coding - Employing safe string manipulation functions to avoid vulnerabilities such as buffer overflows. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Low | High | Critical |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| GCC | 9.3 | -Wstringop-overflow | GCC's compiler warnings include -Wstringop-overflow, which warns about potential buffer overflows in string operations. |
| [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 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-SQL] | Preventing SQL Injection |

| **Noncompliant Code** |
| --- |
| The following code directly uses user input in a SQL query, creating a risk of SQL injection. |
| void queryDatabase(char\* userInput) {  char query[256];  sprintf(query, "SELECT \* FROM users WHERE name = '%s'", userInput); // Vulnerable to SQL injection  // Execute query  } |

| **Compliant Code** |
| --- |
| This version uses parameterized queries, which safely separate user input from the query logic. |
| void queryDatabase(char\* userInput) {  // Assume prepared statements and parameterized queries are used  char\* query = "SELECT \* FROM users WHERE name = ?";  // Bind userInput to the query parameter safely  // Execute query  } |

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

| **Principles(s):** Input Validation and Sanitization - Validating and sanitizing user inputs and employing parameterized queries are essential practices to prevent SQL injection. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Critical | High | Medium | Critical | Critical |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| OWASP ZAP | 2.9.0 | SQL Injection | OWASP ZAP can dynamically analyze web applications for vulnerabilities like SQL injection, helping developers identify and fix security issues. |
| [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 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-AST] | Proper Use of Assertions |

| **Noncompliant Code** |
| --- |
| Using assertions for error handling rather than for detecting programming errors. |
| void processFile(const char\* filename) {  FILE\* file = fopen(filename, "r");  assert(file != NULL); // Improper use of assertion for error handling  // Process the file  fclose(file);  } |

| **Compliant Code** |
| --- |
| Checking the file pointer and handling the error without relying on assertions. |
| void processFile(const char\* filename) {  FILE\* file = fopen(filename, "r");  if (file == NULL) {  // Handle error properly, e.g., logging and returning an error code  return;  }  // Process the file  fclose(file);  } |

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

| **Principles(s):** Error Handling - Assertions are for catching developer errors, not for handling runtime errors or user input errors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Medium | Low | Medium | Important |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | 10.0.1 | alpha.core.AssertSideEffect | Detects assertions that could have side effects, ensuring assertions do not become part of the application logic. |
| [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-EXC] | Exception Safety and Handling |

| **Noncompliant Code** |
| --- |
| This example shows a function that might throw an exception, potentially causing a resource leak if the exception is thrown before the resource is released. |
| void exampleFunction() {  int\* ptr = new int[10]; // Resource allocation  // Operations that might throw an exception  delete[] ptr; // Resource release  } |

| **Compliant Code** |
| --- |
| Using RAII (Resource Acquisition Is Initialization) to ensure that resources are properly released even if an exception is thrown. |
| void exampleFunction() {  std::vector<int> vec(10); // RAII manages resource allocation and deallocation  // Operations that might throw an exception  } |

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

| **Principles(s):** Resource Management and Error Handling - Managing resources and errors efficiently and safely, even in the face of exceptions. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Medium | High | Critical |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPcheck | 2.3 | exceptionSafety | Cppcheck can analyze code for potential exceptions safety issues, helping to ensure that resources are managed correctly. |
| [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 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-IVL] | Robust Input Validation |

| **Noncompliant Code** |
| --- |
| The following code does not validate the size of the input before using it, which can lead to buffer overflow if the input exceeds expected limits. |
| void copyInput(char\* input) {  char buffer[100];  strcpy(buffer, input); // Potential buffer overflow  } |

| **Compliant Code** |
| --- |
| This version includes validation to ensure the input size does not exceed the buffer's capacity, preventing buffer overflow. |
| void copyInput(char\* input) {  char buffer[100];  if (strlen(input) < sizeof(buffer)) {  strcpy(buffer, input);  } else {  // Handle error or truncate input  }  } |

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

| **Principles(s):** Security and Reliability - Input validation is a fundamental security measure that also contributes to the reliability of the application by preventing unexpected behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Medium | High | Critical |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| OWASP ZAP | 2.9.0 | N/A | While primarily a web application security scanner, OWASP ZAP can be used in a development environment to simulate attacks and test the robustness of input validation routines. |
| [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 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CDM] | Comprehensive Code Documentation |

| **Noncompliant Code** |
| --- |
| This code snippet lacks comments, making it difficult to understand the purpose and functionality of the function. |
| void process(int\* data, int size) {  for (int i = 0; i < size; i++) {  data[i] = data[i] \* 2;  }  } |

| **Compliant Code** |
| --- |
| The compliant code includes comments that explain the function's purpose and its parameters. |
| // Processes the array 'data' of length 'size' by doubling each element  void process(int\* data, int size) {  for (int i = 0; i < size; i++) {  data[i] = data[i] \* 2; // Double the element  }  } |

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

| **Principles(s):** Maintainability and Readability - Code documentation significantly improves the maintainability and readability of the codebase, ensuring that developers can efficiently work with and modify the code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Low | Low | Medium | Moderate |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Doxygen | 1.8.17 | N/A | Doxygen is a documentation generator that can create software documentation from annotated source code. It supports various programming languages, including C and C++. |
| [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** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CMG] | Effective Concurrency Management |

| **Noncompliant Code** |
| --- |
| This example accesses a shared resource without proper synchronization, leading to a potential race condition. |
| int counter = 0; // Shared resource  void incrementCounter() {  counter++; // Unsafe access in a multi-threaded environment  } |

| **Compliant Code** |
| --- |
| The compliant code uses a mutex to synchronize access to the shared resource, preventing race conditions. |
| int counter = 0; // Shared resource  pthread\_mutex\_t counterMutex = PTHREAD\_MUTEX\_INITIALIZER;  void incrementCounter() {  pthread\_mutex\_lock(&counterMutex);  counter++; // Safe access  pthread\_mutex\_unlock(&counterMutex);  } |

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

| **Principles(s):** Performance and Security - Effective concurrency management is key to maximizing performance in multi-threaded applications while also securing shared resources against concurrent access issues. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | High | High | Critical |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helgrind | 3.15.0 | N/A | Helgrind is a Valgrind tool for detecting synchronization errors in C, C++, and other languages. It can identify race conditions, deadlocks, and misuses of POSIX threads. |
| [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** |
| --- | --- | --- |
| [Student Choice] | [STD-010-DTC] | Use of Appropriate Data Types |

| **Noncompliant Code** |
| --- |
| Using an integer type (int) for a variable that is expected to store fractional numbers. |
| int calculatePercentage(int part, int whole) {  return (part / whole) \* 100; // Incorrect calculation due to integer division  } |

| **Compliant Code** |
| --- |
| Using a floating-point type (float or double) for fractional calculations to ensure accuracy. |
| double calculatePercentage(int part, int whole) {  return ((double)part / whole) \* 100; // Correct calculation with floating-point division  } |

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

| **Principles(s):** Precision and Accuracy - Ensuring that operations have the necessary precision and that data types accurately represent the values they are supposed to hold. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | High | Low | High | Important |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| GCC | 9.3 | -Wconversion | The GCC compiler with the -Wconversion flag warns about implicit conversions that may alter a value, helping to identify potential data type misuse. |
| Clang | 11.0.0 | -Wfloat-conversion | Clang's -Wfloat-conversion flag warns about implicit conversions from floating-point to integer types, highlighting areas where precision might be lost. |
| [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.]

### 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-MPR | High | Likely | Low | High | 4 |
| STD-002-CPP | High | Unlikely | Medium | High | 2 |
| STD-003-VAL | Medium | Likely | Low | Medium | 3 |
| STD-004-STR | Low | Possible | Low | Low | 2 |
| STD-005-SQL | High | possible | High | High | 4 |
| STD-006-ASS | Medium | Unlikely | Medium | Medium | 3 |
| STD-007-EXC | Low | Likely | Low | Low | 2 |
| STD-008-CDM | Low | Likely | Low | Medium | 2 |
| STD-009-CMG | Low | Possible | Medium | Medium | 2 |
| STD-010-DTC | Medium | Likely | Low | High | 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 is the process of encrypting data that is stored on a device or a storage service. It is designed to protect data from being accessed by unauthorized users, especially in the event of theft or unauthorized access to storage media. This policy applies to all stationary data, including files on servers, laptops, or databases. It is crucial for protecting sensitive information, such as personal data, trade secrets, and financial records.  Policy: All sensitive data stored within the organization's systems must be encrypted using industry-standard encryption protocols such as AES-256. Keys used for encryption should be managed securely, with regular rotations and strict access controls. |
| Encryption at flight | Encryption in flight is a measure that involves encrypting data while it travels through the network, ensuring that it is not intercepted or eavesdropped. The encryption method between systems, servers and applications in transit apply to data encryption. Data protection in a network of high security is especially important when the data representative is being sent via an unsafe network like the internet. Policy: Data can be encrypted with TLS or IPSec encryption protocols whenever it is sent using either internal or external networks. Credentials management includes getting rid of old ones and updating them to prevent illegal data transmission. |
| Encryption in use | Encryption in use involves enclaving processing applications which is taking place via the particular applications. The very nature of this kind of encryption enables computations even when the data environment is not trusted. But even so, the data processing remains secure and confidential. It's critical for cases where the moderate level of security is required, like for cloud services processing of personal data in shared environments. Policy: Any application that handles or distributes sensitive data needs to have its processing layers secured by homomorphic encryption or secure multiparty computation to protect data at all times. Developers ought to get to the encryption in a place where there is no chance of disrupting the performance and features of the app. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process that underlies the proper identification of a user or system. This mechanism verifies that the person or the entity which has requested the access is an actual author of the request in question. This usually entails authentication of credentials like ID and passphrase, or facial and fingerprint recognition, or tokens. Policy: Every user should confirm their identity by 2FA (two-factor authentication) to be subsequently get the access to any secure system and data. The Organization shall use robust authentication mechanisms, such that phishing and replay attack cannot bypass the systems. |
| Authorization | Authorization is the process of granting or denying access to resources based on an authenticated user's permissions. It ensures that users can only access resources appropriate for their role or status.  Policy: After authentication, users must be authorized to access only those resources necessary for their role within the organization. Access control lists, role-based access control (RBAC), or attribute-based access control (ABAC) should be employed to enforce least privilege principles. |
| Accounting | Accounting, often referred to as auditing, is the process of tracking user activities and recording security-relevant information. It is used for monitoring, billing, and for historical purposes to investigate security incidents.  Policy: All user activities that involve access to or modification of sensitive data must be logged and monitored. Logs should be stored securely for an appropriate period and reviewed regularly for signs of unauthorized or suspicious activities. |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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