**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 | This is the first layer of defense that protects a system against external threats. Controlling input data allows us to prevent many common forms of attacks, such as SQL injections. It allows us to safely make assumptions when storing and using data, therefore preventing crashes and errors, and unexpected system behavior. |
| 1. Heed Compiler Warnings | While they are not fatal errors, compiler warnings can reveal important information about potential errors and vulnerabilities in the code, such as overflows or accessing null pointers. They can help us to fix and prevent potential bugs in the system. |
| 1. Architect and Design for Security Policies | A secure and sound architecture is the foundation for a system impervious to threats, vulnerabilities, and risks. The architecture is a blueprint that incorporates security requirements, the principle of defense in depth, authentication, and authorization mechanisms, etc.. |
| 1. Keep It Simple | The solution to a problem should be as simple and straightforward as possible because intricacies and unnecessary complications introduce errors. Simplicity promotes maintainability and therefore security because we can understand and test the code better. |
| 1. Default Deny | By default, deny access to system resources to all but authorized users. This helps reducing the risk of unauthorized access, data breaches, or malicious activities. |
| 1. Adhere to the Principle of Least Privilege | In order to carry out tasks, users should have the least necessary privileges in the system. This helps contain damage in case of unauthorized access or security breaches. Users should have access only to the resources essential for their specific tasks, nothing more. |
| 1. Sanitize Data Sent to Other Systems | This is a form of validation for our input supplied to other systems. We ensure data follows the expected format and requirements of the receiving system and is free of malicious content. |
| 1. Practice Defense in Depth | Implementing multiple, independent, and complementing layers of security so that, in case of potential attacks, if one layer fails, the others can still prevent or reduce the damages caused. |
| 1. Use Effective Quality Assurance Techniques | Concurrent QA during development allows us to detect bugs and errors as early as possible, thereby improving system security, reducing costs overall, and yielding defect-free products that satisfy the user experience. |
| 1. Adopt a Secure Coding Standard | Usually, there is no one right way to code something. But coding standards provide guidelines and best practices for developer to write code that secure, consistent, maintainable, and compliant. |

### 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** | EXP39-C | “Do not access a variable through a pointer of an incompatible type.”  Modifying a variable via a pointer of an incompatible type may lead to unexpected system behavior. |

| **Noncompliant Code** |
| --- |
| The following function declares a float variable. Then, it captures and casts the address of that variable to an integer pointer. Finally, it attempts to dereference, modify, and print the value of the pointer: 1.4013e-45 |
| void Foo() {  float f = 0.0f;  int\* ip = (int\*)&f;  (\*ip)++;  std::cout << f << std::endl;  } |

| **Compliant Code** |
| --- |
| This function uses the nextafterf() function to calculate the next “representable floating-point” value that comes after the given value, 0.0f. FLT\_MAX is a predefined variable that represents the maximum possible that can be represented by float. As an argument for nextafterf, it represents the direction of the next value. |
| void Foo() {  float f = 0.0f;  f = nextafterf(f, FLT\_MAX);  std::cout << f << std::endl;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard. In this case, the compiler does not issue any warnings or errors. However, we can avoid this error by following the C++ secure coding guidelines. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | N/A | Casting from float\* to signed int\* is not portable due to different binary data representation on different platforms. |

**Reference Links**

<https://wiki.sei.cmu.edu/confluence/display/c/EXP39-C.+Do+not+access+a+variable+through+a+pointer+of+an+incompatible+type>

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT33-C | “Ensure that division and remainder operations do not result in divide-by-zero errors.”  Division by 0 is undefined and will either throw an exception or cause unexpected system behavior. |

| **Noncompliant Code** |
| --- |
| In the following code snippet, the divide function performs a simple division, but does not perform any validation on the arguments. |
| double divide(double a, double b) {  return a / b;  } |

| **Compliant Code** |
| --- |
| Here, divide will check if the divisor is 0. If so, an exception is thrown. The caller should do input validation before calling this function. |
| double divide(double a, double b) {  if (b == 0) {  throw std::invalid\_argument(“Divisor cannot be 0”);  return a / b;  } |

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

| **Principles(s):** Validate input data. By checking for invalid arguments, we prevent potential errors within our program. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | zerodiv | Context sensitive analysis of division by zero. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR30-C | Do not attempt to modify string literals. In C/C++ (as well as other other static languages), string literals are immutable, and attempting to change them will result in errors and unexpected system behavior. |

| **Noncompliant Code** |
| --- |
| The following initializes a string literal and attempts to modify it, which is not allowed because string literals are immutable. |
| const char\* stringLiteral = “Hello world!”;  stringLiteral[11] = ‘.’; |

| **Compliant Code** |
| --- |
| In the following code snippet, we first copy the string literal into a character array (std::string is also valid), then modify the variable by value. |
| Char charArray[] = “Hello world!”;  charArray[11] = ‘.’; |

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

| **Principles(s):** Heed compiler warnings. The compiler will not allow modifying an immutable type. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio IDE | Community 2022 |  | Expression must be a modifiable lvalue |
| Cppcheck | 2.10 |  | Modifying string literal directly or indirectly is undefined behavior. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-nnn-LLL] | [Rationalize the standard.] |

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

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

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | INT32-C | Ensure that operations on signed integers do not result in overflow |

| **Noncompliant Code** |
| --- |
| We attempt to increment the greatest supported integer on the system, which we retrieve using std::numeric\_limits<int>:max(), which causes a numeric overflow. |
| #include <numeric>  int add(int a, int b) {  return a + b;  }  int main() {  add(std::numeric\_limits<int>::max(), 1);  } |

| **Compliant Code** |
| --- |
| We introduce input validation to the add function that would throw an exception if the condition (a + b <= std::numeric\_limits<int>::max()) is false. |
| #include <numeric>  int add(int a, int b) {  if (a > std::numeric\_limits<int>::max() - b) {  throw std::invalid\_argument(“Arguments cause overflow”);  return a + b;  }  int main() {  add(std::numeric\_limits<int>::max(), 1);  } |

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

| **Principles(s):** ValidateInput Data. We check the arguments and avoid potential numeric overflows, which could inadvertently lead to release of sensitive data (as error messages, console logs, etc..) |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS-Studio | 7.25 | V1026, V1070, V1081, V1083, V1085, V5010 | [Insert text.] |
| Astrée | 23.04 | Integer-overflow | Fully checked |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | MSC11-C | Incorporate diagnostic tests using assertions.  Assertions are ordinarily used for debugging during development and are turned off for release. |

| **Noncompliant Code** |
| --- |
| The following function creates a smart pointer to an integer vector, uses it to add a few elements, then proceeds to deallocate the memory. Then, it makes the assertion the pointer is null. However, if something goes wrong and reset fails, the assertion aborts the program, making it susceptible to denial-of-service attacks and its alike. |
| void Foo() {  std::unique\_ptr<std::vector<int>> vecPtr = std::make\_unique<std::vector<int>>();  vecPtr->push\_back(1);  vecPtr->push\_back(2);  vecPtr->push\_back(3);  for (auto element : \*vecPtr)  std::cout << element << std::endl;  vecPtr.reset();  assert(vecPtr == NULL);  } |

| **Compliant Code** |
| --- |
| In this revised function, we handle memory errors without aborting the program. |
| void Foo() {  std::unique\_ptr<std::vector<int>> vecPtr = std::make\_unique<std::vector<int>>();  vecPtr->push\_back(1);  vecPtr->push\_back(2);  vecPtr->push\_back(3);  for (auto element : \*vecPtr)  std::cout << element << std::endl;  vecPtr.reset();  if (vecPtr == NULL) {  return;  }  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques. Assertions allow us to set up preconditions and make certain assumptions when testing the program for errors or for expected behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| Coverity | [Insert text.] | ASSERT\_SIDE\_EFFECT | Can detect the specific instance where assertion contains an operation / function call that may have a side effect |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR54-CPP | Catch handlers should order their parameter types from most derived to least derived |

| **Noncompliant Code** |
| --- |
| In the following program, we create two custom exception classes meant to deal with specific exceptions. Both classes extend std::exception, and ExceptionB extends ExceptionA. Catch statements are executed in order of appearance. Since exception is the base for the 2 other classes, it handles those exceptions as well thanks to polymorphism, meaning the remaining catch statements are never executed. |
| class ExceptionA : public std::exception {};  class ExceptionB : public ExceptionA {};  int main(){  try { /\*some code\*/ }  catch (std::exception& e) {}  catch (ExceptionB& e) {}  catch (ExceptionA& e) {}  } |

| **Compliant Code** |
| --- |
| This example places the catch statements in the correct order. The most derived (i.e. the most specialized exception type) comes before the general exception types. |
| class ExceptionA : public std::exception {};  class ExceptionB : public ExceptionA {};  int main(){  try { /\*some code\*/ }  catch (ExceptionB& e) {}  catch (ExceptionA& e) {}  catch (std::exception& e) {}  } |

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

| **Principles(s):** Adopt a Secure Coding Standard. The noncompliant code from this section is bad practice that could result in low program quality. We can avoid such mistakes by following standard and secure coding practices. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Astrée | 22.10 | exception-caught-by-earlier-handler | Fully checked |

#### 

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory Protection | FIO51-CPP | Close files when they are no longer needed.  Not properly closing files would occupy resources in the system, puts the integrity of data at risk of loss or leak. |

| **Noncompliant Code** |
| --- |
| Even though the following calls the terminate function to end the program, the function does not properly call the destructor for the file stream object, indicating file resources are not properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  std::terminate();  } |

| **Compliant Code** |
| --- |
| Closing the file before calling the terminate function ensures the file resources are properly released. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    file.close();    if (file.fail()) {      // Handle error    }    std::terminate();  } |

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

| **Principles(s):** Adopt a Secure Coding Standard. When working with files, we should follow best practices and handle system resources responsibly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3P0 | ALLOC.LEAK | Leak |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions | EXP53-CPP | Do not read uninitialized memory.  Reading uninitialized memory leads to undefined system behavior and often throws exceptions. |

| **Noncompliant Code** |
| --- |
| In this non-compliant example, we attempt to read data from an uninitialized variable, which will cause undefined behavior, leading to errors. |
| #include <iostream>    void f() {    int i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| This example correctly initializes a variable before attempting to read. |
| #include <iostream>    void f() {  int i = 0;  std::cout << i;  } |

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

| **Principles(s):** Heed compiler warnings. The compiler will warn of uninitialized variables, and would not compile the code if attempting to read from those variables. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | uninitialized-read | Partially checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | MSC50-CPP | Do not use std::rand() for generating pseudorandom numbers.  The std::rand() function produces consistent and predictable results, even with random seeds. |

| **Noncompliant Code** |
| --- |
| Noncompliant code uses the rand function to generate a number in the [0-10000) range, but does not even use a custom seed. |
| void f() {  std::string id("ID");  id += std::to\_string(std::rand() % 10000);  // ...  } |

| **Compliant Code** |
| --- |
| The following uses an algorithm provided by the C++ standard library “for fine-grained control over pseudorandom number generation. It breaks random number generation into two parts: one is the algorithm responsible for providing random values (the engine), and the other is responsible for distribution of the random values via a density function (the distribution)" |
| #include <random>  #include <string>    void f() {  std::string id("ID"); // Holds the ID, starting with the characters "ID" followed  // by a random integer in the range [0-10000].  std::uniform\_int\_distribution<int> distribution(0, 10000);  std::random\_device rd;  std::mt19937 engine(rd());  id += std::to\_string(distribution(engine));  // ...  } |

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

| **Principles(s):** Adopt a Secure Coding Standard. Use more robust libraries (such as random by C++ Standard Library) for generating pseudo-random numbers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | bad-function (AUTOSAR.26.5.1A) | Fully checked |
| CodeSonar | 7.3p0 | BADFUNC.RANDOM.RAND | Use of rand |

### 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.

We begin by reviewing the coding guidelines and best practices, as well as the security standards, such as those by SEI CERT C++, to come up with a comprehensive policy. In addition, we can integrate a static code analysis tool that automatically scans the code for potential security vulnerabilities. An example is incorporating Cppcheck into the development pipeline. The pipeline also benefits from testing tools that check against different threats. Lastly, we should implement auditing tools that perform regular security scans on the system, so that potential vulnerabilities or policy violations may be identified and fixed.

### 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 | Unlikely | Medium | High | 2 |
| EPX39-C | Medium | Unlikely | High | Low | 3 |
| ERR54-CPP | Medium | Likely | Low | High | 1 |
| EXP53-CPP | High | Probable | Medium | High | 1 |
| FIO51-CPP | Medium | Unlikely | Medium | Low | 3 |
| INT32-C | High | Likely | High | Medium | 2 |
| INT33-C | High | Likely | High | High | 2 |
| STR30-C | Low | Likely | Low | High | 2 |
| MSC11-C | Low | Likely | High | Low | 3 |
| MSC50-CPP | Medium | Unlikely | Low | Medium | 2 |

### 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 | This involves encrypting data when it is stored in databases or files, so that even if the storage is compromised, the data remains secure against unauthorized access. Therefore, per policy, all confidential system data should be encrypted before storage. |
| Encryption at flight | This involves encrypting data while it is being transmitted over networks, eliminating threats of eavesdropping, or otherwise intercepting the connection. Thus, it is policy to transport data via encryption protocols such as Transport Layer Security (TLS) or Secure Sockets Layer (SS) and use strong encryption algorithms, such as SHA-256. |
| Encryption in use | This involves encrypting data while it is being processed or otherwise used by the system components; it aims to provide protection against unauthorized access to sensitive data during its processing. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is verifying the identity of a user before granting access to the system or its resources. The policy will require password-based authentication, where users must use a unique username and password to validate their identity. |
| Authorization | Once a user is authenticated, authorization will determine what permissions or privileges that user has in the system. Per policy, the users are assigned roles that define their abilities, such as reading from or writing to specific resources, and executing certain functions. |
| Accounting | Accounting is auditing the system. It involves tracking and recording the actions, activities, and usage of resources by the users or components of the system. Per policy, the system logs will be analyzed daily to ensure compliance and detect unauthorized system activity. |

**\***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 | 06/15/2023 | Initial Template | Afshin E. Ahvazi |  |
| [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 |