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



# Green Pace Secure Development Policy

## Contents

[Green Pace Secure Development Policy 0](#_Toc169460691)

[Contents 1](#_Toc169460692)

[Overview 3](#_Toc169460693)

[Purpose 3](#_Toc169460694)

[Scope 3](#_Toc169460695)

[Module Three Milestone 3](#_Toc169460696)

[Ten Core Security Principles 3](#_Toc169460697)

[C/C++ Ten Coding Standards 4](#_Toc169460698)

[Coding Standard 1 5](#_Toc169460699)

[Coding Standard 2 6](#_Toc169460700)

[Coding Standard 3 8](#_Toc169460701)

[Coding Standard 4 9](#_Toc169460702)

[Coding Standard 5 11](#_Toc169460703)

[Coding Standard 6 13](#_Toc169460704)

[Coding Standard 7 15](#_Toc169460705)

[Coding Standard 8 17](#_Toc169460706)

[Coding Standard 9 19](#_Toc169460707)

[Coding Standard 10 21](#_Toc169460708)

[Defense-in-Depth Illustration 23](#_Toc169460709)

[Project One 23](#_Toc169460710)

[Revise the C/C++ Standards 23](#_Toc169460711)

[Risk Assessment 23](#_Toc169460712)

[Automated Detection 23](#_Toc169460713)

[Automation 23](#_Toc169460714)

[Summary of Risk Assessments 24](#_Toc169460715)

[Create Policies for Encryption and Triple A 24](#_Toc169460716)

[Map the Principles 26](#_Toc169460717)

[Audit Controls and Management 27](#_Toc169460718)

[Enforcement 27](#_Toc169460719)

[Exceptions Process 27](#_Toc169460720)

[Distribution 28](#_Toc169460721)

[Policy Change Control 28](#_Toc169460722)

[Policy Version History 28](#_Toc169460723)

[Appendix A Lookups 28](#_Toc169460724)

[Approved C/C++ Language Acronyms 28](#_Toc169460725)

## 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 | Input data can not only be erroneous, but it can also be malicious. The first thing that should be done to defend against this is to validate input data. This helps with accidental user inputs and mistyped entries as well as help keep attackers at bay. No input should be trusted until it has been validated. |
| 1. Heed Compiler Warnings | The compiler should be set to verbose, and all warnings must be heeded. These warnings help automatically point out security vulnerabilities and bugs. All warnings must be verified safe or remedied. |
| 1. Architect and Design for Security Policies | Security should be woven into the very fabric of the software. At every step of the process, security must be considered. This means that during the earliest stages of design throughout the last stages of life, security needs to baked in. |
| 1. Keep It Simple | Complexity serves only to confuse the next person and open the door for malicious users. Simple solutions are more robust, easier to maintain, easier to read and understand, and are usually more secure. Ingenious simplicity is far better than unnecessary complexity. |
| 1. Default Deny | The default answer to any permission must be to deny. When the unexpected happens, denying by default can be the thing between a costly breach and a denied attacker. Never allow a user access or permission by default. |
| 1. Adhere to the Principle of Least Privilege | A user only ever needs to have the minimum amount of privilege to accomplish a task. Any more than that will always be a security risk. This applies not only to allowing privilege, but also to removing privilege as soon as it is no longer required. |
| 1. Sanitize Data Sent to Other Systems | Raw data should never be sent straight to other systems. This is for similar reasons to input validation. Data cannot be trusted until it has been validated, and other systems do not need more data than is required. |
| 1. Practice Defense in Depth | Defense in depth requires that several layers of security be used to defend against attack vectors. This is necessary to ensure that a single failure does not allow an attacker into the system. It is inevitable that the security of a system will, at some point, fail. One of the best ways to guard against this is to accept it from the beginning and use layers of defense. |
| 1. Use Effective Quality Assurance Techniques | Quality code is safe code. Ensuring that quality assurance is followed in development and maintenance of code ensures security. One of the greatest sources of vulnerabilities are bugs in the code. |
| 1. Adopt a Secure Coding Standard | It is paramount that a secure coding standard is adopted. Following this coding standard will give the development team the necessary tools to secure the code against the would-be attacker. The standards of this organization will be laid out in the following section. |

### 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** | **Ensure that type conversions do not result in a loss of data** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | When converting data types, ensure that data is not lost. For example, converting from a double to an integer will result in some of the data being lost. |

| **Noncompliant Code** |
| --- |
| The initial data is stored as a double. After adding 67, it is stored in an integer. This has resulted in the decimal portion of the data being lost. |
| double data = 123.45;  int too\_small = data + 67; |

| **Compliant Code** |
| --- |
| The correct way to handle this data is to ensure that the accuracy is preserved. |
| double data = 123.45;  double just\_right = data + 67; |

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

| **Principles(s):**  3. Architect and Design for Security Policies  Following this principle means that, during design, the correct variable types are decided, conversions minimized, and protection guaranteed from data loss.  9. Use Effective Quality Assurance Techniques  Quality code includes well used data types and careful conversions. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that division by zero cannot occur** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Do not allow division or remainder operations to cause a value to be divided by zero. This will cause a divide-by-zero error. |

| **Noncompliant Code** |
| --- |
| The below function does not check if num2 is zero before performing division. This would result in a divide-by-zero error if num2 was passed 0. |
| void divide(int num1, int num2) {  int result = num1 / num2;  } |

| **Compliant Code** |
| --- |
| This divide function ensures that num2 is not equal to zero before performing division. |
| void divide(int num1, int num2) {  int result;  if (num2 != 0) {  result = num1 / num2;  }  } |

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

| **Principles(s):**  1. Validate Input Data  Ensure that user input cannot cause a division by zero error.  2. Heed Compiler Warnings  With the compiler set to verbose, it will likely warn the developer that there is a risk of division by zero. Heeding this warning would eliminate the issue.  9. Use Effective Quality Assurance Techniques  Quality code does not include division by zero. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 1.66 | zerodiv  zerodivcond | Context sensitive analysis of division by zero Not detected for division by struct member / array element / pointer data that is 0 Detected when there is unsafe division by variable before/after test if variable is zero |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Ensure sufficient storage for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Make sure that there is always enough space for the character data and the null terminator. Trying to store more characters than the string has space for will result in a buffer overflow. |

| **Noncompliant Code** |
| --- |
| This code fails to ensure that the user cannot input more than the 20 characters this array has the space for. |
| char[20] input;  std::cin >> input; |

| **Compliant Code** |
| --- |
| By using the getline member function, we can limit the input size that the user can enter. This eliminates the risk of a buffer overflow. |
| char[20] input;  std::cin.getline(input, 20, ‘\n’); |

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

| **Principles(s):**  1. Validate Input Data  Do not allow input data to cause an overflow related to strings. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Validate user input before using SQL queries** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | User input should never be put directly into a SQL query. It should always be validated before using the data. |

| **Noncompliant Code** |
| --- |
| The user is allowed to inject SQL here because the input is not validated before the query string is modified. |
| std::string username;  std::string sql = “SELECT ID, NAME, PASSWORD FROM DB WHERE NAME=”;  std::cin >> username;  sql = sql + “’” + username + “’”; |

| **Compliant Code** |
| --- |
| This code checks for the injection of ‘or 1=1’ injections. This verification ensures that this kind of SQL injection cannot occur. |
| std::string username;  std::string sql = “SELECT ID, NAME, PASSWORD FROM DB WHERE NAME=”;  std::cin >> username;  if(username.find(“or”) == std::string::npos) {  sql = sql + “’” + username + “’”;  } |

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

| **Principles(s):**  1. Validate Input Data  Input data must be validated in order to preclude SQL injection attacks.  5. Default Deny  Deny access to the user by default can help ensure that an SQL related attack does not allow a breach.  7. Sanitize Data Sent to Other Systems  Raw SQL query strings should never be sent directly to other systems. Always sanitize these queries to stop attacks before they happen.  8. Practice Defense in Depth  Layers of defense when dealing with SQL queries are absolutely necessary to stop attacks from gaining significant access. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Manual | N/A | Human verification | SQL queries must me manually tested for security |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Accessing deallocated memory can result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| This memory has been deallocated. This could result in errors that allow arbitrary code execution. |
| struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| This code waits to deallocate the memory until it is no longer needed. |
| struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):**  2. Heed Compiler Warnings  Verbose compilers may warn the developer about attempting to access freed memory.  3. Architect and Design for Security Policies  Design systems such that memory is carefully allocated and deallocated. There should never be a need to access memory that has been deallocated in a well-designed system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Do not rely on assertions to verify condition** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Assertions can be removed in later production code. Never rely on assertions to verify the condition of data. |

| **Noncompliant Code** |
| --- |
| This code uses an assertion to verify that b is not zero. If assertions are later removed, this check would no longer function. |
| int func(int a, int b) {  assert(b != 0);  return a / b;  } |

| **Compliant Code** |
| --- |
| Now b is verified by the function. |
| int func(int a, int b) {  assert(b != 0);  int result = 0;  if(b != 0) {  result = a / b;  }  return result;  } |

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

| **Principles(s):**  10. Adopt a Secure Coding Standard  The standards must be followed in development to ensure that assertions do not cause side effects, variables are not validated in assertions, and error checking does not depend on assertions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2023.9.0 | ASSERT\_SIDE\_EFFECTS | Finds side effects caused by assertions |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | All exceptions must be caught by the nearest handler possible. No exception can cause an abrupt termination of the program. |

| **Noncompliant Code** |
| --- |
| This code does not catch the exception thrown by throwing\_func, and therefore causes the program to terminate. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| This code fixes this by using a try…catch block to handle the exception. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

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

| **Principles(s):**  2. Heed Compiler Warnings  The compiler can warn the developer of uncaught exceptions. Always heed these warnings and fix uncaught exceptions  10. Adopt a Secure Coding Standard  Follow the standard for secure coding so that all exceptions are caught. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.FUNCS.IRV  LANG.ERRCODE.NOTEST  LANG.ERRCODE.NZ | Ignored return value Missing Test of Error Code Non-zero Error Code |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Close files when they are no longer needed** |
| --- | --- | --- |
| File Management | STD-008-CPP | Files must be closed when they are no longer needed. Keeping them open only increases the chances that a vulnerability will be found and that it can cause a lot of damage. This keeps in line with the principle of privilege. |

| **Noncompliant Code** |
| --- |
| This code opens a file stream and calls the termination handler without closing the file. |
| #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** |
| --- |
| This code ensures that the close function is called before termination. |
| #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):**  6. Adhere to the Principle of Least Privilege  As soon as the file is no longer needed, closing the file ensures that the least privilege is given to the user with regards to file access. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.LEAK | Leak |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do not delete a polymorphic object without a virtual destructor** |
| --- | --- | --- |
| Objects | STD-009-CPP | A polymorphic object must have a virtual destructor to be deleted without cause undefined behavior. |

| **Noncompliant Code** |
| --- |
| The below polymorphic object lacks a virtual destructor. When it is later deleted, undefined behavior results. |
| struct Base {  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

| **Compliant Code** |
| --- |
| The polymorphic object now has a virtual destructor to ensure well-defined behavior upon destruction. |
| struct Base {  virtual ~Base() = default;  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

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

| **Principles(s):**  2. Heed Compiler Warning  The compiler may warn the developer about the lack of a destructor. Always fix this issue.  10. Adopt a Secure Coding Standard  The standard requires that these objects include a destructor so that this is never an issue. Following this standard should eliminate this threat. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 1.66 | leakReturnValNotUsed | Doesn't use return value of memory allocation function |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do not read uninitialized memory** |
| --- | --- | --- |
| Expressions | STD-010-CPP | Variables that have been declared but not initialized have the potential to cause undefined behavior should they be read. |

| **Noncompliant Code** |
| --- |
| The following code prints the variable to the console before it has been initialized. This results in undefined behavior. |
| #include <iostream>    void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| The variable has now been initialized before being printed to the console. |
| #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):**  2. Heed Compiler Warnings  The compiler will likely warn the developer in the case of uninitialized memory. Fixing the issues reported by the compiler will stop issues like this often.  3. Architect and Design for Security Policies  Design the software such that variables are always initialized. If there are no uninitialized variables, this issue will never happen.  10. Adopt a Secure Coding Standard  The developers should always initialize variables in the same line that they are defined. This will keep uninitialized memory from ever being accessed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 1.66 | uninitvar  uninitdata  uninitstring  uninitMemberVar  uninitStructMember | Detects uninitialized variables, uninitialized pointers, uninitialized struct members, and uninitialized array elements (However, if one element is initialized, then cppcheck assumes the array is initialized.) There are FN compared to some other tools because Cppcheck tries to avoid FP in impossible paths. |

### 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.

Automation will be used to ensure that the standards are followed in several stages of the process. In the design, build, verify and test, transition and health check, monitor and detect, and maintain and stabilize phases automation can be used. Using automation throughout the entire process ensures that security is woven into the software from the very beginning and throughout its life. Considering security as an intrinsic part of the software development process is one of the best ways to build more robust and secure systems.

### 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 | Probable | High | Medium | 2 |
| STD-002-CPP | Low | Likely | Medium | Medium | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Likely | High | Medium | 2 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Unlikely | Medium | Low | 3 |
| STD-007-CPP | High | Likely | Medium | High | 1 |
| STD-008-CPP | Medium | Unlikely | Medium | Low | 3 |
| STD-009-CPP | High | Probable | High | Medium | 2 |
| STD-010-CPP | High | Probable | Medium | High | 1 |

### 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 at rest | Data is at rest when it is stored for later use. This data tends to be the most valuable data, but also the least at risk. To protect data at rest, we will use encryption at rest by enforcing full disk encryption. This will ensure that any lost memory hardware will be inaccessible. |
| Encryption in flight | Data is in flight when it is being transmitted between points. This data is at more risk than data at rest due to the nature of transmitted data. Encryption in flight will be accomplished by ensuring that data is encrypted before transmission. This will cause data lost or procured without permission to be unreadable. |
| Encryption in use | Data is in use when the user is actively interacting with the data. This data is at the most risk due to its proximity to the user. Encryption in use will be accomplished through the Triple-A policies below. This will ensure that the right user has the right amount of access and it monitored as long as this access is held. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of identifying the user. This will be accomplished through the use of two-factor authentication. These factors will be strong passwords and USB access keys. Authentication verifies the identity of the user so that attackers cannot gain access to the system through false identities. Addition of new users must be done in accordance with authentication policies to ensure that the user data is valid, the user is not malicious, and the user has the correct level of access rights. |
| Authorization | Authorization is given after authentication, and it is the granting of access and/or permissions to the necessary areas of the system. Only the minimum level of access and permissions will be granted to the user based on their needs and prior authorization. This ensures that users have only the minimum trust necessary to accomplish the tasks they need to perform. User logins are granted through the use of authorization. |
| Accounting | Accounting is the monitoring of user activity while they are logged into the system. If all else fails, monitoring user activity can allow the system to flag suspicious activity and put a stop to attackers before significant damage is done. Changes to the database and files access by users are monitored during a user’s session. |

**\***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 |  |
| 1.1 | 05/26/2024 | Principles and Standards Defined | Shawn Way |  |
| 1.2 | 06/16/2024 | Full Policy Developed | Shawn Way |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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