**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 | Input validation is just a way to ensure we are only getting the correct input, and sanitizing anything that can be malicious. This helps us prevent attacks such as injection attacks by validating characters, numbers, and length. |
| 1. Heed Compiler Warnings | Watch your compiler warnings, this is the basis of this principle. These warnings can help you with finding vulnerabilities before they even become one. |
| 1. Architect and Design for Security Policies | Integrating security into the architectural and design phases of development ensures that security policies are considered from the beginning. This approach helps in creating a robust and secure system by design rather than as an afterthought. |
| 1. Keep It Simple | Simplicity keeps things simple, and simple things are usually easier to keep secure and that is in anything in life. The more complex something is the more things we must consider. |
| 1. Default Deny | Default deny means that we are going to start out with no authorization. We will gain access by asking regardless of how small it is. This will help protect from unwanted access. |
| 1. Adhere to the Principle of Least Privilege | Granting the minimum level of access or permissions required for a user or system to perform its tasks reduces the potential impact of security incidents. This principle minimizes the privileges granted to users or processes. |
| 1. Sanitize Data Sent to Other Systems | Before sending data to other systems or components, it's essential to sanitize the data to remove any potentially harmful content. This practice helps prevent injection attacks and ensures the integrity of data transmitted between different parts of a system. |
| 1. Practice Defense in Depth | Implementing multiple layers of security defenses is key to protecting a system. Defense in depth involves having redundant security measures to compensate for failures in one layer and to provide a more comprehensive security posture. |
| 1. Use Effective Quality Assurance Techniques | Incorporating quality assurance techniques specific to security helps identify and address vulnerabilities during the development life cycle. Rigorous testing and code reviews are essential components of effective quality assurance. |
| 1. Adopt a Secure Coding Standard | Following a secure coding standard provides guidelines and best practices for writing secure code. It helps developers avoid common security pitfalls and ensures consistency in secure coding practices across a development team. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | This standard requires explicit definition and rational use of data types in the code to enhance clarity, maintainability, and prevent unintended type-related issues. |

| **Noncompliant Code** |
| --- |
| We cannot add these two different data types as the results will not be correct, we will not get 10 as the char ‘5’ is not actually equal to 5. |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  char first = ‘5’;  int second = 5;  int third = first + second; |
|  |

| **Compliant Code** |
| --- |
| We can use the same data type such as long and we will get our intended value of 10 as a result. |
| [Compliant code block; code should be indented using 12-point Courier New font.]  int first = 5;  int second = 5;  int third = first + second; |

**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** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | This standard focuses on validating data values to prevent issues such as overflow or unexpected behavior due to incorrect data. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  In the example we will cause overflow as a short data type can hold a maximum value of 32,767 |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  short myShort = 32,767;  short myShortTwo = 10;  Short addShorts = myShort + myShortTwo; |

| **Compliant Code** |
| --- |
| [Compliant description]  In this example we will not cause overflow as we will not pass our maximum value a short can hold |
| [Compliant code block; code should be indented using 12-point Courier New font.]  short myShort = 32,757;  short myShortTwo = 10;  Short addShorts = myShort + myShortTwo; |

**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** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2021.1 | CERT\_C-INT30-a  CERT\_C-INT30-b  CERT\_C-INT30-c | Prevent integer overflows:    Ensure there are no integer overflows or underflows in constant expressions involving the '+', '-', '\*' operators.    Guard against integer overflow or underflow in constant expressions when using the '<<' operator. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | This standard emphasizes proper handling and validation of strings to prevent common vulnerabilities like buffer overflows. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  This code could possibly cause a buffer overflow due to not being sanitized to ensure we do not pas our char arrays maximum of 10 |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  char buffer[10];  strcpy(buffer, userInput); |

| **Compliant Code** |
| --- |
| [Compliant description]  This updated version of the code will ensure we sanitize the users input so we do not exceed our maximum |
| [Compliant code block; code should be indented using 12-point Courier New font.]  char buffer[10];  strncpy(buffer, userInput, sizeof(buffer)); |

**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** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | LANG.MEM.BO  LANG.MEM.TO  MISC.MEM.NTERM  BADFUNC.BO.\* | Address buffer overrun issues:  Detect and handle type overruns and ensure sufficient space for null terminators.  Implement a set of warning classes to report instances of library function usage that may lead to internal buffer overflows. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | This standard aims to prevent SQL injection vulnerabilities by promoting the use of parameterized queries and other secure coding practices when interacting with databases. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] In this example we will just run the query without parameterizing it and this can lead to the SQL injections |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  std::string userInput = "John Doe";  std::string query = "SELECT \* FROM Users WHERE Name = '" + userInput + "';";  executeQuery(query); |

| **Compliant Code** |
| --- |
| [Compliant description] In this example we will use a function that will parameterize the query and this will help ensure that we sanitize any unwanted input that can leave us vunerable |
| [Compliant code block; code should be indented using 12-point Courier New font.]  std::string userInput = "John Doe";  std::string query = "SELECT \* FROM Users WHERE Name = ?";  executeParameterizedQuery(query, userInput); |

**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** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| TrustInSoft Analyzer | 1.38 | Mem\_access | Exhaustively verified |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | This standard focuses on preventing memory-related vulnerabilities such as memory leaks, and buffer overflows. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  This example we will forget to deallocate our memory and potentially causing a vulnerability. |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  void processData() {  int\* dynamicArray = new int[10];  } |

| **Compliant Code** |
| --- |
| [Compliant description] We will ensure after we are done using the array to deallocate the memory to prevent a leak |
| [Compliant code block; code should be indented using 12-point Courier New font.]  void processData() {  int\* dynamicArray = new int[10];  //anytype of use for the array  delete[] dynamicArray;  } |

**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** |
| --- | --- | --- | --- | --- |
| High | Liekly | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Identify inconsistencies in the handling of function call return values. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | This standard encourages assertions to check conditions expected to be true during program execution. Assertions help identify unexpected situations and facilitate debugging. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  This code does not ensure int b is not 0 thus causing problems |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  int divide(int a, int b) {  return a / b;  } |

| **Compliant Code** |
| --- |
| [Compliant description]  This code does ensure that int b is not 0 and will help with debugging. |
| [Compliant code block; code should be indented using 12-point Courier New font.]  int divide(int a, int b) {  assert(b != 0);  return a / b;  } |

**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** |
| --- | --- | --- | --- | --- |
| Low | Low | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | LANG.FUNCS.ASSERTS | Insufficient use of assertions. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | This standard guides the proper use of exceptions for error handling, promoting a consistent and effective approach to handle exceptional situations. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  This code fails to catch any exceptions that may be caused and will end in the program crashing |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  void processFile(std::string filename) {  std::ifstream file(filename);  } |

| **Compliant Code** |
| --- |
| [Compliant description]  This code properly sets up a try/catch to ensure we open the files and if not, it will catch the exception and not cause the program to crash. |
| [Compliant code block; code should be indented using 12-point Courier New font.]  void processFile(std::string filename) {  try {  std::ifstream file(filename);  if (!file.is\_open()) {  throw std::runtime\_error("Failed to open file");  }  // ...  } catch (const std::exception& e) {  // Handle the exception (log, rethrow, or other appropriate action)  }  } |

**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** |
| --- | --- | --- | --- | --- |
| Low | Probably | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Ensure that every exception explicitly thrown in the code has a corresponding handler of a compatible type in all call paths that could potentially reach that specific point in the code. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Naming Conventions | STD-008-CPP | This standard defines naming conventions for variables, functions, classes, and other identifiers to ensure consistency, readability, and maintainability of the codebase. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  Not using this standard can make this very hard to read. This is not algebra it's okay to make it easier to read |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  int calculateResult(int x, int y) {  //Poor naming conventions as x and y can get confusing when many things are happening in code  } |

| **Compliant Code** |
| --- |
| [Compliant description] |
| [Compliant code block; code should be indented using 12-point Courier New font.]  int calculateResult(int operand1, int operand2) {  //Great naming convention as I won't confused operand1 for anything else in my code  } |

**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** |
| --- | --- | --- | --- | --- |
| Low | Probably | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Static Analysis Tool | 2022,1 | NameConventionChecker | NameWizard's NamingConventionChecker enforces consistent and readable naming conventions for variables, functions, and classes. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Code Documentation | STD-009-CPP | This standard outlines guidelines for documenting code, including comments, Doxygen-style documentation, and other documentation practices to enhance code readability and understanding. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] This code will not have any notes which could make understanding what is happening hard on a large scaled |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  int addNumbers(int a, int b) {  return a + b; //Lack of documentation  } |

| **Compliant Code** |
| --- |
| [Compliant description] This code will have great comments that will help you understand what the purpose of the code it |
| [Compliant code block; code should be indented using 12-point Courier New font.]  /\*\*  \* @brief Adds two numbers.  \*  \* This function adds two integers and returns the result.  \*  \* @param a The first operand.  \* @param b The second operand.  \* @return The sum of a and b.  \*/  int addNumbers(int a, int b) {  return a + b;  } |

**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** |
| --- | --- | --- | --- | --- |
| Medium | Possible | Low | P3 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Doxygen | 1.8.20 | DocumentationCheck | DocSense's DocumentationChecker ensures proper code documentation, including comments and Doxygen-style documentation for enhanced readability. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Indentation | STD-010-CPP | Indentation is one of the most important standards as reading code that is not indented gives the code no hierarchy and makes readability hard. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  This example shows the code without indentation and how hard it can be to read this and understand this |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  void nonCompliantFunction() {  int x = 5; // Noncompliant: Inconsistent indentation  if (x > 0) {  cout << "Positive" << endl;  } else {  cout << "Non-positive" << endl;  }  } |

| **Compliant Code** |
| --- |
| [Compliant description]  This code has indentation that show you the flow, and helps with the readability of the code |
| [Compliant code block; code should be indented using 12-point Courier New font.]  void CompliantFunction() {  int x = 5;  if (x > 0) {  cout << "Positive" << endl;  }  else {  cout << "Non-positive" << endl;  }  } |

**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** |
| --- | --- | --- | --- | --- |
| Low | Probably | Low | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-format | 12.0.1 | IndetationChecker | IndentGuard's IndentationChecker enforces proper indentation for improved code hierarchy and readability. |

### 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 can be initiated during the Build phase by transforming manual processes into a cohesive CI/CD pipeline. This not only streamlines workflows within the team but also integrates tools like Docker for managing container instances, GitLab for version control, and Jenkins for continuous integration.

To ensure robust security measures, the SecOps phase involves automating virtualized container deployment, enabling the implementation of automated security tests and regression tests within the QA environment.

In monitoring and detection, a proactive approach is taken by automating static application security tests within nightly builds, focusing on critical sections of the code. Real-time vulnerability assessment is achieved by seamlessly embedding dynamic application security testing into the Software Development Life Cycle (SDLC).

Enhancing the security posture further, tools such as OWASP Dependency-Check are employed to scrutinize code dependencies, effectively identifying potential vulnerabilities. This comprehensive automation strategy reinforces security throughout the development lifecycle, promoting efficiency and reliability.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Likely | High | P9 | L2 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Low | High | P1 | L3 |
| STD-007-CPP | Low | Probably | Medium | P4 | L3 |
| STD-008-CPP | Low | Probably | Medium | P4 | L3 |
| STD-009-CPP | Medium | Possible | Low | P3 | L2 |
| STD-010-CPP | Low | Probable | Low | P4 | L3 |

### 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 refers to the encryption applied to data that is stored or at rest. In this state, the policy stipulates the use of robust encryption algorithms to safeguard data integrity and prevent unauthorized access in storage environments. |
| Encryption at flight | Encryption during transmission or in flight is crucial for securing data as it traverses networks. The policy mandates the use of encryption protocols during data transfer to prevent interception or eavesdropping, ensuring the confidentiality and integrity of information in transit. |
| Encryption in use | This pertains to encrypting data while it is being processed or used by applications. The policy emphasizes the implementation of encryption mechanisms to protect data during active processing, mitigating the risk of unauthorized access or manipulation. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication involves verifying the identity of users or systems attempting to access resources. The policy dictates the implementation of robust authentication mechanisms, such as multi-factor authentication, to ensure that only authorized entities gain access to sensitive information. |
| Authorization | Authorization determines the permissions and access rights granted to authenticated users. The policy specifies the need for well-defined authorization policies, restricting access based on roles and responsibilities, thereby preventing unauthorized actions or data access. |
| Accounting | Accounting involves tracking and monitoring user activities to maintain an audit trail. The policy underscores the importance of comprehensive logging and monitoring practices to facilitate accountability, forensic analysis, and compliance adherence. |

**\***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 |  |
| 1.5 | 02/16/2024 | Update for all current policies | David Waid |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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