**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 | Properly validating input data ensures that only valid and expected inputs are accepted by the system, reducing the risk of unauthorized access, data corruption, or unintended disclosure of sensitive information. |
| 1. Heed Compiler Warnings | Compilers provide warnings about potential security vulnerabilities or weaknesses in the code, so it is essential to listen to them when warnings appear. Ignoring these warnings can leave the system vulnerable to various types of attacks. |
| 1. Architect and Design for Security Policies | It is important to consider security during the architectural and design phases of software development. By implementing security policies from the start, developers can integrate security controls, access controls, and protective measures into the system's design. |
| 1. Keep It Simple | Complex systems are more vulnerable to security issues as they are difficult to understand, test, and maintain. It is important for developers maintain simplicity to allow for enhanced security through the reduction of the amount of complex code and increase in overall organization. |
| 1. Default Deny | Default Deny refers to the practice of denying access by default and granting it only when explicitly authorized. This approach minimizes the risk of unauthorized access and ensures consistent access controls throughout the system. |
| 1. Adhere to the Principle of Least Privilege | The Principle of Least Privilege states that users or components should have only the minimum privileges they need for their specific tasks. By limiting individuals access to the minimum required, the potential damage from a compromised user or component is significantly less. |
| 1. Sanitize Data Sent to Other Systems | This principle emphasizes the importance of sanitizing data. Data sanitization involves validating and cleansing data to remove any potentially malicious content or unintended data structures. By sanitizing data, developers can prevent data corruption, injection attacks, and other security vulnerabilities that can arise from untrusted or malformed data. |
| 1. Practice Defense in Depth | Defense in Depth refers to using multiple layers of security controls to protect a system that includes measures like access controls, encryption, intrusion detection systems, and regular security assessments. By implementing multiple layers of defense systems have significantly increased security. |
| 1. Use Effective Quality Assurance Techniques | It is important to actively employ quality assurance (QA) techniques such as rigorous testing, code reviews, vulnerability assessments, and penetration testing throughout the software development lifecycle. By implementing these QA activities, developers can discover and fix issues prior to software deployment, improving the overall reliability and security of the system. |
| 1. Adopt a Secure Coding Standard | It is important to utilize established coding guidelines that promote secure programming. Employing these standards can help reduce the risk of common errors that can lead to significant vulnerabilities. |

### 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-DTC | **Data Type Coding Standard** - Using improper data types can lead to unpredictable program behavior and potential security vulnerabilities such as integer overflow or underflow, truncation errors, or precision loss. |

| **Noncompliant Code** |
| --- |
| The double value pi is being assigned to an integer variable truncatedPi. This could lead to truncation of the decimal part, causing a loss of precision. |
| double pi = 3.141592653589793;  int truncatedPi = pi; // potential loss of precision |

| **Compliant Code** |
| --- |
| The double value pi is assigned to another double variable correctPi, ensuring no loss of precision. |
| double pi = 3.141592653589793;  double correctPi = pi; // no loss of precision |

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

| **Principles(s):**  Adopt a Secure Coding Standard - Using proper data types is a basic coding practice that helps to promote security and reliability.  Heed Compiler Warnings – Compilers will typically provide warnings about data type mismatches along with other data type issues. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarLint | 6.16 | S5276 - Implicit casts should not lower precision | Detects when casting is being used that would lower the precision of the value being cast. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-DVC | **Data Value Coding Standard** - Incorrect or unsafe handling of data values can cause unexpected behavior or security vulnerabilities. It's important to validate and sanitize data inputs and respect the boundaries of data types. |

| **Noncompliant Code** |
| --- |
| An negative integer value is being assigned to an unsigned integer, which leads to a wrap-around, resulting in y having the maximum value that can be held by an unsigned integer. |
| int x = -1;  unsigned int y = x; // unsafe conversion, y now has maximum unsigned int value |

| **Compliant Code** |
| --- |
| Before assigning the value of x to y, we ensure that x is non-negative. This prevents wrap-around when converting to an unsigned integer. |
| int x = -1;  if (x >= 0) {  unsigned int y = x; // safe conversion  } |

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

| **Principles(s):** Validate Input Data – Validation of input data emphasizes the proper handling of data values. By properly handling data values within the boundaries of different data types and data inputs we can ensure that only valid inputs are processed by a given system. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarLint | 6.16 | S845 - Signed and unsigned types should not be mixed in expressions | Detects when signed and unsigned types converted implicitly. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-SCS | **String Correctness Coding Standard** - Misuse of string operations can lead to vulnerabilities such as buffer overflows, off-by-one errors, and other undefined behavior. |

| **Noncompliant Code** |
| --- |
| The string is too long to fit into the buffer, leading to a buffer overflow. |
| char buffer[10];  strcpy(buffer, "This string is too long for the buffer"); // buffer overflow |

| **Compliant Code** |
| --- |
| The strncpy function is used to safely copy the string into the buffer. The buffer is null-terminated to ensure it's a valid string. |
| char buffer[10];  strncpy(buffer, "This string is too long for the buffer", sizeof(buffer) - 1); // safe copying  buffer[sizeof(buffer) - 1] = '\0'; // null termination |

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

| **Principles(s):** Adopt a Secure Coding Standard – Following a standard to avoid misuse of string operations can help to mitigate vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarLint | 6.16 | S5801 - Using "strcpy" or "wcscpy" is security-sensitive | Warns that using “strcpy” or “wcspy” lead to vulnerabilities as buffer overflow can occur. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-SQL | **SQL Injection Coding Standard** - SQL injection vulnerabilities arise when user-supplied data is sent to a SQL server without adequate escaping or filtering. This can result in unauthorized access or modification of data. |

| **Noncompliant Code** |
| --- |
| User input is directly appended to a SQL query, which can allow a SQL injection if the user input is not properly sanitized. |
| std::string unsafe\_query = "SELECT \* FROM users WHERE name = '" + user\_input + "';"; // SQL Injection possible |

| **Compliant Code** |
| --- |
| User input is not directly inserted into the SQL query. Instead, a placeholder is used in the query, and the user input is securely bound to the placeholder. |
| std::string safe\_query = "SELECT \* FROM users WHERE name = ?"; // SQL query with placeholder  prepare\_statement(safe\_query);  bind\_parameter(1, user\_input); |

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

| **Principles(s):** Sanitize Data Sent to Other Systems – Sanitizing input data prior to it being sent to an SQL server will prevent potential SQL injection attacks. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Critical | Medium | High | Critical | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | CheckString() | Detects incorrect usage of strings and related functions. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-MPC | **Memory Protection Coding Standard** - Improper memory management can lead to security vulnerabilities such as buffer overflows, memory leaks, and double frees. |

| **Noncompliant Code** |
| --- |
| An array was allocated with new[], but is being deallocated with delete. This results in undefined behavior. |
| char \*buffer = new char[10];  // ...  delete buffer; // incorrect delete, should use delete[] |

| **Compliant Code** |
| --- |
| The array that was allocated with new[] is correctly deallocated with delete[]. |
| char \*buffer = new char[10];  // ...  delete[] buffer; // correct delete[] |

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

| **Principles(s):** Keep It Simple – Maintaining simple code also means managing memory properly. Properly managing memory will save time in the future and reduce the overall complexity of the system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarLint | 6.16 | S5025 – Memory should not be managed manually | Alerts the user when memory is managed manually and suggests utilizing C++ tools that automatically manage memory. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-ACS | **Assertions Coding Standard** - Assertions provide a way to test assumptions made by the program and catch unexpected conditions. |

| **Noncompliant Code** |
| --- |
| There is no assertion or check to validate the result of the calculate function. |
| int x = calculate();  // no check for the value of x |

| **Compliant Code** |
| --- |
| An assertion is used to validate the assumption that x should be greater than zero after the calculate function is called. |
| int x = calculate();  assert(x > 0); // asserting that x should be greater than zero |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – Assertions are a way to check a program for unexpected conditions and are thus part of assuring quality code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | CheckAssert() | Checks for unexpected effects from assert statements |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-ECS | **Exceptions Coding Standard** - Proper usage of exceptions in C++ is critical for writing robust and error-resistant code. |

| **Noncompliant Code** |
| --- |
| An exception is being caught, but no action is taken in the catch block. This can make debugging difficult, as the exception is silently ignored. |
| try {  // ...  } catch (...) {  // catch block is empty  } |

| **Compliant Code** |
| --- |
| The exception is caught and the error message is printed to std::cerr. This provides information about the error that occurred. |
| try {  // ...  } catch (const std::exception& e) {  std::cerr << "Caught exception: " << e.what() << '\n';  } |

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

| **Principles(s):** Architect and Design for Security Policies – Proper usage of exceptions is not only essential for writing error resistant code but it is also a major part of the architecture and design phase. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | CheckExceptionSafety() | Checks that exceptions are properly handled. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Loop Variable Scope** | STD-008-LVS | **Loop Variable Scope Coding Standard** - Proper scope of loop variables can prevent accidental misuse of values, leading to clearer, more readable code. |

| **Noncompliant Code** |
| --- |
| The loop variable i is still accessible after the loop, which may lead to unintentional reuse. |
| int i;  for(i = 0; i < 10; ++i)  {  // ...  }  // i is still accessible here |

| **Compliant Code** |
| --- |
| The loop variable i is scoped to the loop, preventing unintentional reuse. |
| for(int i = 0; i < 10; ++i)  {  // ...  }  // i is not accessible here |

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

| **Principles(s):** Keep It Simple – Proper scoping of loop variables can help prevent use of values outside of their intended scope, make code simpler, and make code more readable. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Low | Low | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarLint | 6.16 | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Function Complexity | STD-009-FNC | **Function Complexity Coding Standard** - Keeping functions simple and focused improves code readability and maintainability, and reduces the chance of bugs. |

| **Noncompliant Code** |
| --- |
| The function doManyThings may contain a lot of complex code which makes it harder to read, understand, and maintain. |
| void doManyThings()  {  // A lot of complex code here...  } |

| **Compliant Code** |
| --- |
| The function doManyThings is simplified by breaking its logic into smaller, more manageable functions. |
| void doOneThing1()  {  // Some code here...  }  void doOneThing2()  {  // Some code here...  }  void doOneThing3()  {  // Some code here...  }  void doManyThings()  {  doOneThing1();  doOneThing2();  doOneThing3();  } |

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

| **Principles(s):** Keep It Simple – Keeping functions simple and within a reasonable scope allows for more readable, maintainable, reusable, and bug free code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | High | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | CheckFunctions() | Checks for various issues within functions. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Implicit Type Conversion** | STD-010-IPC | **Implicit Type Conversion Coding Standard** - Implicit type conversions can cause unexpected behavior and bugs. It's better to make these conversions explicit for code readability and to avoid errors. |

| **Noncompliant Code** |
| --- |
| The double value x is implicitly converted to an integer when assigned to y, causing truncation of the fractional part. |
| double x = 10.5;  int y = x; // Implicit conversion from double to int |

| **Compliant Code** |
| --- |
| The double value x is explicitly converted to an integer when assigned to y. The use of static\_cast makes the conversion clear. |
| double x = 10.5;  int y = static\_cast<int>(x); // Explicit conversion from double to int |

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

| **Principles(s):** Heed Compiler Warnings – Compilers typically give warnings for implicit conversions, and following these warnings can help prevent bugs or unexpected behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarLint | 6.16 | S5276 - Implicit casts should not lower precision | Detects when casting is being used that would lower the precision of the value being cast. |

### 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.

Utilizing the DevSecOps diagram and our coding standards and tools we can implement a plan to uphold the code standards we have set. Starting with the Assess and plan stage we can utilize this stage to relay the standards we have set and the tools we have chosen to address and uphold those standards. Then during the design stage, we can actively utilize static code analyzers to enforce our standards. During the Building stage we can utilize again utilize tools like SonarLint to ensure the source code is compliant with our standards. During the Verification and Testing stage we will utilize automated unit testing to ensure the code does not have any unexpected behaviors, memory leaks, improperly handled inputs, etc. Then if any issues arise during the Transition and Health Check stage or the Monitor and Detect stage that cannot be directly addressed in the respond stage, we will repeat from the Assess and Plan stage to address any changes in standards or tools we might need.

### 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-DTC | High | High | Low | High | 4 |
| STD-002-DVC | High | High | Medium | High | 4 |
| STD-003-SCS | High | Medium | High | High | 3 |
| STD-004-SQL | Critical | Medium | High | Critical | 5 |
| STD-005-MPC | Critical | High | High | Critical | 5 |
| STD-006-ACS | Medium | Low | Low | Medium | 3 |
| STD-007-ECS | High | Medium | High | High | 3 |
| STD-008-LVS | Low | Low | Low | Low | 3 |
| STD-009-FNC | Medium | High | Medium | High | 4 |
| STD-010-IPC | High | High | Low | High | 4 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption in rest is the encryption of data stored on a physical device such as a hard drive.  This policy requires that all sensitive data being stored must be encrypted using algorithms that are sufficient to protect the data from attackers and to meet compliance standards. Encryption should take place prior to the data’s storage on any device.  Regular auditing should be performed to ensure this policy is being put into practice. |
| Encryption at flight | Encryption at flight is the encryption of data as it is being transmitted over a network.  This policy requires that data being transmitted over a network be encrypted using algorithms that are sufficient to protect the data from attackers and to meet compliance standards.  Regular auditing of code should be performed prior to the use of the code. Regular audits of existing code should also be performed to ensure that encryptions meet current standards and to ensure no errors were missed during the initial auditing. |
| Encryption in use | Encryption in use is the encryption of data while it is actively being used in an application.  This policy requires that sensitive information be encrypted in memory up until the information the information is needed and then encrypted again once it is no longer in immediate use.  Regular auditing of code is necessary to ensure sensitive information is being encrypted and the policy is being put into practice. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying a user’s identity prior to the use of a system.  This policy requires that authentication be used to verify users’ identities protecting from unauthorized access to a system.  Prior to a use accessing a system or sensitive information all users must be verified using login credentials, two-factor authentication, or another form of identity validation. Different levels of authentication can be used depending on the sensitivity of what is being accessed. Accounts that are no longer in use should be removed to prevent the risk of accounts being compromised. |
| Authorization | Authorization is the differentiation of permissions and access levels between different authenticated users within a system.  This policy requires that users be assigned a specific level of access which allows the user to access only the information and systems necessary to perform their related functions.  Implementation of an authorization system with different levels of access must be implemented to allow them access to information and systems. The default level of permission should be the lowest level of access or no access. Users access should be regularly maintained and users who no longer need access should be adjusted accordingly. |
| Accounting | Accounting is the logging of all user actions within a system.  This policy requires the logging of actions performed by users within the system. These actions should include user logins, changes to the database, addition of new users, changes to user access levels, and files being accessed. This allows for monitoring of actions and tracking of possible unauthorized access and compromised user accounts.  The system should keep logs of actions performed and flag actions that are deemed suspicious or high risk. These logs should be regularly reviewed. |

**\***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 | 05/27/2023 | Module 3 | Robert Kratzke |  |
| 1.2 | 06/11/2023 | Project 1 | Robert Kratzke |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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