**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 | User input should be checked to ensure it is correct and safe to run before it is processed to avoid issues like buffer overflows, injection attacks, and other forms of incorrect input. |
| 1. Heed Compiler Warnings | Compiler warnings should be reviewed to identify possible bugs in the code and vulnerabilities. |
| 1. Architect and Design for Security Policies | Security policies implemented prior to architect and design processes help mitigate security risk as a forethought rather than an afterthought |
| 1. Keep It Simple | Complexity in code can lead to unexpected issues such as bugs, memory issues, and make it difficult to edit later. |
| 1. Default Deny | Systems should deny access by default and only allow with the correct permissions to avoid unauthorized access. |
| 1. Adhere to the Principle of Least Privilege | Only give access to what is necessary to perform tasks and no more. This helps minimize risk. |
| 1. Sanitize Data Sent to Other Systems | Make sure that data sent to other systems is masked, encrypted, or removed to ensure that it cannot be accessed by unwanted means. |
| 1. Practice Defense in Depth | Having more than one layer of security to ensure if one fails others will provide protection. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques like code reviews and automated testing should be used in the process. |
| 1. Adopt a Secure Coding Standard | Follow secure coding standards to develop secure code and follow industry best practices to avoid problems. |

### 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 | Data type long long vs. int |

| **Noncompliant Code** |
| --- |
| This shows the incorrect use of the int data type that may cause an overflow resulting in an incorrect value |
| #include <iostream>  using namespace std;    int testNum1 = 1234567890123456789; //Using incorrect data type will cause an overflow  int main()  {  cout << testNum1 << endl;  } |

| **Compliant Code** |
| --- |
| Using the correct data type for the data used will avoid an overflow and allow the value to be accurate |
| #include <iostream>  using namespace std;    long long int testNum1 = 1234567890123456789ll; //Using the long long int data type will avoid an overflow  int main()  {  cout << testNum1 << endl;  } |

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

| **Principles(s):**  Validate input data – This standard follows this principal by ensuring correct data types are used to avoid overflows  Adopt a secure coding standard – This follows with best practices to ensure secure code that prevents common errors |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Low | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cpp Check | 2.14 | Variable Type | Detects possible misuse of data types |

Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Use of constants |

| **Noncompliant Code** |
| --- |
| The use of an arbitrary number for a limit can lead to risk of possible error and make it difficult to understand what is being represented. |
| #include <iostream>;  using namespace std;    int truckHeight;    void getTruckHeight()  {  cout << "Enter the truck height: ";  cin >> truckHeight;  }    int main()  {  getTruckHeight();    if (truckHeight > 12) // the use of an arbitrary number instead of constraints  {  cout << "Truck over max height" << endl;  }  else  {  cout << "Truck height within limit" << endl;  }  return 0;  } |

| **Compliant Code** |
| --- |
| The use of a constant value makes the code more readable and can reduce possible error |
| #include <iostream>;  using namespace std;    int truckHeight;  const int maxTruckHeight = 12; //making a constant value for max truck height    void getTruckHeight()  {  cout << "Enter the truck height in feet: ";  cin >> truckHeight;  }    int main()  {  getTruckHeight();    if (truckHeight > maxTruckHeight) //using a constant makes the code more readable and reuseable  {  cout << "Truck over max height" << endl;  }  else  {  cout << "Truck height within limit" << endl;  }  return 0;  } |

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

| **Principles(s):**  Architect and Design for security policies – The standard follows this principle by encouraging the use of constants in the design phase to reduce possible errors  Adopt a secure coding standard – The standard follows this principle by incorporating best practices to use constants for constant values to reduce risk that could lead to vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cpp Check | 2.14 | Variable can be declared const | Checks that constants are used when applicable |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Use of input validation |

| **Noncompliant Code** |
| --- |
| Incorrect input validation by verifying string correctness can lead to unexpected issues and open up to malicious attacks |
| #include <iostream>    using namespace std;    int main() {  char name[10]; //buffer is only 10 characters long    cout << "Enter your name: ";  cin >> name; //No bounds checking can lead to buffer overflow if greater than 10 characters    cout << "Hello, " << name << "!" << endl;    return 0;  } |

| **Compliant Code** |
| --- |
| The validation of input will help to stop possible buffer overflow when user input is taken. Note: there are many ways to validate user input this is only one example. |
| #include <iostream>    using namespace std;    int main() {  char name[10]; //buffer is only 10 characters long    cout << "Enter your name: ";  cin.get(name, 10) >> name; //Limits user input to buffer size validating input    cout << "Hello, " << name << "!" << endl;    return 0;  } |

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

| **Principles(s):**  Validate input data – This standard follows this principle by ensuring that user input is properly validated to prevent issues like buffer overflow  Sanitize data sent to other systems – This standard follows this principle by checking input data is safe |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cpp Check | 2.14 | Buffer Overrun | Detects potential for buffer overflow |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Concatenated SQL queries without validation |

| **Noncompliant Code** |
| --- |
| The failure to separate the user input and validate before inputting the SQL query can allow for malicious injection |
| #include <iostream>  using namespace std;    int main()  {  string userId;  cout << "Enter user ID: ";  cin >> userId;    string query = "SELECT \* FROM users WHERE id = " + userId; //concatenated string without validation    cout << query << endl;    return 0;  } |

| **Compliant Code** |
| --- |
| The compliant way of handling this would be to not concatenate the string but to use database functions to handle and validate the user input |
| #include <iostream>  using namespace std;    int main()  {  string userId;  cout << "Enter user ID: ";  cin >> userId;    std::string query = "SELECT \* FROM users WHERE id = ?";  prepStatement(query); //placeholder for database handling functions  setParam(1, userId); //placeholder for database handling functions    cout << query << endl;    return 0;  } |

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

| **Principles(s):**  Validate input data – This standard follows this principle by ensuring user input is validated to prevent SQL injection attacks  Sanitize data sent to other systems – This standard follows this principle by sanitizing input data before it is used in SQL queries |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.4 | SQLInjection | Identifies possible vulnerability for SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Free up memory |

| **Noncompliant Code** |
| --- |
| A failure to delete memory used after use can create memory leaks |
| #include <iostream>  using namespace std;    int main()  {  int\* array = new int[55];    //fails to delete the allocated memory    return 0;  } |

| **Compliant Code** |
| --- |
| Always free up dynamically allocated memory after use to avoid memory leaks |
| #include <iostream>  using namespace std;    int main()  {  int\* array = new int[55];    delete[] array; //frees up memory after use    return 0;  } |

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

| **Principles(s):**  Effective quality assurance techniques – This standard follows this principle by managing the memory which ensures the application is secure. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Low | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ValGrind | 3.23 | Memory Check | Detects memory leaks |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Use of assertions |

| **Noncompliant Code** |
| --- |
| Assertions are used for debugging purposes and should be removed before releasing code |
| #include <iostream>  #include <cassert>    using namespace std;      int main()  {  int x = 20;    assert(x < 10); //leaving assertion in after debugging    return 0;  } |

| **Compliant Code** |
| --- |
| Assertion has been removed |
| #include <iostream>  #include <cassert>    using namespace std;      int main()  {  int x = 20; //assertions have been removed  return 0;  } |

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

| **Principles(s):**  Heed Compiler warnings – this standard follows this principle by ensuring that assertions are removed in the production code  Keep it simple – This standard follows this principle by removing unnecessary code which can clean up the code and reduce risk. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cpp Check | 3.14 | Assert usage | Identifies assertions left in code |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Use of try and catch to handle exceptions |

| **Noncompliant Code** |
| --- |
| Failure to use a try and catch for exception handling can result in an error. In this example the program will run into a runtime error by trying to divide by zero which will crash the program. |
| #include <iostream>    using namespace std;      int main()  {  int numerator = 10;  int denominator = 0;  int result;    result = numerator / denominator; //without a try and catch this will result in a runtime error  cout << "Result: " << result << endl;    return 0;  } |

| **Compliant Code** |
| --- |
| Using the try and catch method to handle exceptions we can stop the program from crashing and produce an exception that shows the error. |
| #include <iostream>  #include <stdexcept>    int main()  {  int numerator = 10;  int denominator = 0;  int result;    try  {  if (denominator == 0)  {  throw std::runtime\_error("division by zero");  }  result = numerator / denominator;  std::cout << "Result: " << result << std::endl;  }  catch (const std::runtime\_error& e)  {  std::cerr << "Error: " << e.what() << std::endl;  }    return 0;  } |

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

| **Principles(s):**  Practice defense in depth – This standard follows this principle by incorporating proper exception handling to protect against unexpected failures  Use effective quality assurance techniques – This standard follows this principle by using proper exception handling. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.4 | Exception Handling | Checks that exceptions are handled properly |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Naming Conventions** | STD-008-CPP | Use of identifiable naming conventions |

| **Noncompliant Code** |
| --- |
| The naming conventions for the data types do not represent identifiable aspects of what they represent |
| #include <iostream>    using namespace std;    int main()  {  const int z = 21;  int x;  bool y = true;  } |

| **Compliant Code** |
| --- |
| The naming conventions of the data types are able to be understood easily |
| #include <iostream>    using namespace std;    int main()  {  const int minAge = 21;  int userAge;  bool isUserOfAge = false;    } |

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

| **Principles(s):**  Keep it simple – This standard follows this principle by ensuring clear naming conventions are used to simplify the code and maintain readability |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang tidy | 20.0 | Readability identifier naming | Enforces coding guidelines on naming conventions |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Declarations** | STD-009-CPP | Do not declare multiple variables in one line |

| **Noncompliant Code** |
| --- |
| There are multiple declarations on one line may cause confusion or difficulty reading the code leading to an error |
| int a = 5, b = 10, c = 15; |

| **Compliant Code** |
| --- |
| Each variable is declared on a separate line making the code more readable and easier to understand |
| int a = 5;  int b = 10;  int c = 15; |

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

| **Principles(s):**  Keep it simple – This standard follows the principle by declaring variables one at a time to improve the readability and making sure it's simple and easily maintainable |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 20.0 | Readability Multiple Declarations | Identifies multiple declarations on the same line |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Floating point numbers** | STD-010-CPP | Do not directly compare floating point numbers for equality |

| **Noncompliant Code** |
| --- |
| Using direct comparison of floating-point values will cause incorrect results due to small errors in calculation |
| #include <iostream>    using namespace std;    int main()  {  double a = 0.1 + 0.2;  if (a == 0.3)  {  cout << a << endl;  }  else  {  cout << "failed" << endl;  }    } |

| **Compliant Code** |
| --- |
| Using an epsilon value to define a small tolerance allows us to compare floating point numbers |
| #include <iostream>    using namespace std;    int main()  {  double a = 0.1 + 0.2;  double epsilonValue = 1.0e-9;  if (abs(a - 0.3) < epsilonValue)  {  cout << a << endl;  }  else  {  cout << "failed" << endl;  }  } |

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

| **Principles(s):**  Adopt a secure coding standard – This standard follows the principle by using secure coding practices to avoid subtle errors |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Low | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 20.0 | Floating Point Comparison | Identifies Floating point comparisons that don't use epsilon value |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | Medium | Likely | Low | Medium | 3 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Probable | Medium | High | 1 |
| STD-005-CPP | High | Probable | Low | High | 2 |
| STD-006-CPP | Low | Unlikely | Low | Low | 4 |
| STD-007-CPP | High | Likely | Medium | High | 1 |
| STD-008-CPP | Low | Likely | Low | Medium | 3 |
| STD-009-CPP | Low | Likely | Low | Medium | 3 |
| STD-010-CPP | Medium | Probable | Low | Medium | 2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | **Policy**: All stored data shall be encrypted.  **What**: Encryption at rest is the protection of data that is stored  **Applied**: Any time data is stored in a database, hard drive, CD, etc. The data should be encrypted  **Why use**: This policy should be used to help prevent data retrieval in the event of a breach. |
| Encryption in flight | **Policy**: All data being transferred shall be encrypted  **What**: Encryption in flight is the protection of data moving in a network  **Applied**: Any time data is being transferred from one location to another it needs to be protected by encryption  **Why use**: This policy should be used to prevent data retrieval in the event of interception of transfers. |
| Encryption in use | **Policy**: All data shall be encrypted while being processed by an application.  **What**: Encryption in use is the protection of data being used or processed by an application.  **Applied**: Any time data is being used by an application it should be protected by encryption  **Why use**: The use of this policy will protect the data being used by an application. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the verification of user data typically for user login to identify the user. This can be used to only let trusted or known users access certain applications. |
| Authorization | Authorization is the permission granted to users. This can be used to allow different users different levels of access to an application. |
| Accounting | Accounting is where the user activity is tracked in the system. This can show user activity that is unusual and that may cause a security risk. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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