**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 | Test inputs to make sure the data has been entered correctly into a system, and incorrect data is eliminated and/or prevented. |
| 1. Heed Compiler Warnings | These warnings exist to notify you of errors or issues in code. The code may still execute as errors will stop the code from running, however, a warning will not do this. |
| 1. Architect and Design for Security Policies | Take architecture and design into consideration when building software as it relates to implementing security policies, such as separating a system into subsystems. |
| 1. Keep It Simple | Keeping the scope of the design simple reduces errors in the software. This will allow you to minimize the complexity of security required of the system. |
| 1. Default Deny | The default is to deny access and only be permitted through the correct correlation of conditions within the schema. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege says that users are only given the least amount of privilege needed to perform their functions adequately. This reduces the possibility of accidental or malicious actions that could occur. |
| 1. Sanitize Data Sent to Other Systems | As data is sent from one system to another, it should be sanitized which will remove any negative or unneeded content. This helps to ensure the prevention of attacks like SQL injection. Enabling only clean data to be sent helps to maintain security of the systems. |
| 1. Practice Defense in Depth | Defense in depth is a strategy that has multiple layers of defense to protect the given systems and/or data. This strategy facilitates preventive and responsive measures, that can enable beefed up security. If a layer is passed, the next layers will continue to protect the system. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques can be testing, or reviewing code that aid in identifying security vulnerabilities, ideally, early in the development lifecycle. This process can help to ensure coding is of the quality required. |
| 1. Adopt a Secure Coding Standard | By adopting a secure coding standard, we are able to follow laid out guidelines and ensure coding is performed to the necessary standards. This can also include adhering to security principles and standards throughout the development lifecycle. |

### 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] | Do not write syntactically ambiguous declarations |

| **Noncompliant Code** |
| --- |
| Using ambiguous declarations can lead to misunderstandings within the code. This can result in different interpretations. |
| Int num1, num2; |

| **Compliant Code** |
| --- |
| Declarations should be easy to understand and ensure the data types are clear. This will lead to the prevention of errors and will make the code more usable. |
| Int num1;  Int num1; |

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

| **Principles(s):** Adopt a Secure Coding Standard, this will prevent unnecessary errors, |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion | 7.2.0 | CertC++ DCL52 |  |
| Astrée | 20.10 | Type file spreading | Partial Checked |
| LDRA tool suite | 9.7.1 | 286, 287 S | Fully Implemented |
| Polyspacc Bug Finder | R2024a | CERT C++:DCL52-CPP | Const-qualified reference types  Modification of const-qualified reference types |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not access an object outside of its lifetime |

| **Noncompliant Code** |
| --- |
| Accessing an object outside of its lifetime can cause inconsistencies or security vulnerabilities. Caused by an object being used after it has been destroyed, leading to invalid memory access. |
| void code() {  int\* a = b(10);  delete a;  std::cout << a << std::endl;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the pointer a is set to nullptr after the object is deleted to make sure it is not accessed again, preventing undefined behavior. |
| void code() {  int\* a = b(10);  delete a;  a = nullptr;  if (a) {  std::cout << a << std::endl;  }  } |

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

| **Principles(s):** Validate Input Data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Uninitialized read | Partially Checked |
| LDRA tool suite | 9.7.1 | 2 D, 36 S | Fully Implemented |
| Polyspace Bug Finder | R2024a | CERT C++:MSC52-CPP | Missing return statements |
| RuleChecker | 23.10 | Return-implicit | Fully checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CCP] | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| The object is created from the results of a call to std::getenv(). Since std::getenv() returns a null pointer on failure, this code can lead to errors. |
| void a() {  std::string str(std::getenv("STR"));  if (!str.empty()) |

| **Compliant Code** |
| --- |
| The results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| void a() {  const char \*strPtrVal = std::getenv("STR");  std::string str(strPtrVal ? strPtrVal : "");  if (!str.empty()) |

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

| **Principles(s):** Heed complier Warnings, as the complier should inform the dev of this issue |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.10 | Assert\_failure |  |
| Helix QAC | 2024.1 | C++3162, C++3164 |  |
| Parasoft c++ test | 2021.1 | CERT\_CPP-ST51a | Avoid null pointer |
| Polyspace Bug Finder | R2024a | CERT C++:STR53-CPP | Array access  Tainted index  Pointer Deference |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL Injection |

| **Noncompliant Code** |
| --- |
| The user input is in a SQL query, making it susceptible to SQL injection attacks. |
| std::string username = getRequestString("username");  std::string password = getRequestString("password");  std::string query = "SELECT \* FROM Users WHERE Name = “ + username + “ AND Pass = “ + password + “; |

| **Compliant Code** |
| --- |
| With a prepared statement user can safely input into the SQL query, aiding in the prevention of SQL injection attacks. |
| PreparedStatement\* stmt = conn->prepareStatement("SELECT \* FROM Users WHERE Name = ? AND Pass  = ?");  stmt->setString(1, username);  stmt->setString(2, password);  stmt->executeQuery(); |

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

| **Principles(s):** Validate Input Data, All inputs should be validated to prevent unwanted or unexpected behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 |  | Stubbing analysis |
| LDRA tool suite | 9.7.1 | 108 D, 109 D | Partially Implemented |
| Polyspace Bug Finder | R2024a | Cert C: STR02-C | Command execution  Library loaded from controlled path  Externally controlled command |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Detect and handle memory allocation errors |

| **Noncompliant Code** |
| --- |
| Without checking for memory allocation errors, the program may access the value stored a null pointer, leading to  undefined behavior. |
| int\* data = new int[10];  memset(data, 0, 10 \* sizeof(int)); |

| **Compliant Code** |
| --- |
| This checks for successful memory allocation and handles any errors. |
| int\* data = new (std::nothrow) int[10];  if (!data) {  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques, This will allow for testing and help ensure the catching of memory problems. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Invalid dynamic memory allocation  Dangling pointer use |  |
| Clang | 3.9 | Clang analyzer  Cplusplus NewDelteLeaks | Does not catch all violations |
| LDRA tool suite | 9.7.1 | 232 S, 236 S, 239 S | Partially Implemented |
| Parasoft Insure ++ |  |  | Runtime Detection |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| Without using static assertions, the code might fail if a constant expression that may not meet certain  conditions, that could inflict undefined behavior at runtime. |
| ARRAY\_SIZE 10;  int arr[ARRAY\_SIZE];  if (ARRAY\_SIZE <= 0) {  } |

| **Compliant Code** |
| --- |
| This uses a static assertion to check that a constant expression meets specific conditions  at compile time. |
| ARRAY\_SIZE 10;  static\_assert(ARRAY\_SIZE > 0, "Array to be greater than 0");  int arr[ARRAY\_SIZE]; |

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

| **Principles(s):** Heed Complier Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CERTC-DCL03 |  |
| Clang | 3.9 | Misc-static-assert | Checked by clang tidy |
| LDRA too suite | 9.7.1 | 44 S | Fully Implemented |
| Parasoft C/C++test | 2023.1 | CERT\_CERROR a | Do not use assertions |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| Abruptly terminating a program using exit can lead to resource leaks, inconsistent states, and loss of data. |
| void processInput(int input) {  if (input < 0) {  std::cerr << "Error: Negative input\n";  std::exit(EXIT\_FAILURE); |

| **Compliant Code** |
| --- |
| This handles errors correctly by cleaning up resources and allowing for an error messages before terminating. |
| void processInput(int input) {  if (input < 0) {  throw std::invalid\_argument("Negative input");  }  //Process input  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques, Testing would allow uncaught exceptions to be realized. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Main-function-catch-all | Partially checked |
| CodeSonar | 7.4p0 | Lang.sturct.uctch | Unreachable catch |
| LDRA tool suite | 9.7.1 | 527 S | Partially Implemented |
| Polyspace Bug Finder | R2023a | CERT C++:ERR50-CPP | Checks for implicit call to terminate() |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-008-CCP] | Write constructor member initializers in the predefined order |

| **Noncompliant Code** |
| --- |
| The member initializer list attempts to initialize aVal then initialize usesAVal to a value that is dependent. The declaration order of the member variables does not match the member initializer order, which causes an unspecified value being stored into usesAVal. |
| int usesAVal;    int aVal;    public:    C(int val) : aVal(val), usesAVal(aVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| Change the declaration order of the class member variables so that the dependency can be ordered properly in the constructor's member initializer list. |
| int aVal;    int usesAVal;    public:    C(int val) : aVal(val), usesAVal(aVal + 1) {}  }; |

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

| **Principles(s):** Keep it simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Initializer-list-order | Fully Checked |
| LDRA tool Suite | 9.7.1 | 206 S | Fully Implemented |
| Helix QAC | 2023.1 | C++3180, C++3181 |  |
| Polyspace Bug Finder | R2023a | CERT C++:DCL58-CPP | Checks for mod of standard namespaces |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous | [STD-009-CPP] | Do not use std::rand() for generating pseudorandom numbers |

| **Noncompliant Code** |
| --- |
| Using std::rand() for generating pseudorandom numbers can lead to issues such as lack of standard distribution. |
| int main() {  int random\_number = std::rand() % 100;  std::cout << "Random number: " << random\_number << "\n";  return 0;  } |

| **Compliant Code** |
| --- |
| Being specific can lead to uniform distribution for generating random numbers when needed. |
| int main() {  std::random\_device rd;  std::mt19937 gen(rd());  std::uniform\_int\_distribution<> dis(1, 100);  int random\_number = dis(gen);  std::cout << "Random number: " << random\_number << "\n";  return 0;  } |

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

| **Principles(s):** Adopt a Secure Coding Standards |
| --- |

**Threat Level**

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

**Automation**

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

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-010-CCP] | Use valid iterator ranges |

| **Noncompliant Code** |
| --- |
| Each iteration of the internal loop compares the first with the second checking for equality. If they are not equal it will increment the first. |
| void a (const std::vector<int> &c) {    std::for\_each(c.end(), c.begin(), [](int i) { std::cout << i; }); |

| **Compliant Code** |
| --- |
| Iterator value passed will be done in the correct order. |
| void a(const std::vector<int> &c) {    std::for\_each(c.begin(), c.end(), [](int i) { std::cout << i; });  } |

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

| **Principles(s):** Architect and design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Overflow upon dereference |  |
| Helix QAC | 2021.2 | C++ 3802 |  |
| Parasoft | 2021.1 | CERT CPP CT53 a | Do not use range that is not really a range |
| PVS-Studio | 7.14 | V539, V662, V789 |  |

### 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 | low | Unlikely | Low | P3 | 3 |
| STD-002-CPP | High | Probable | Medium | P12 | 1 |
| STD-003-CPP | High | Unlikely | Medium | P6 | 2 |
| STD-004-CPP | High | Likely | Medium | P18 | 1 |
| STD-005-CPP | High | Likely | Medium | P18 | 1 |
| STD-006-CPP | Low | Unlikely | High | P1 | 3 |
| STD-007-CPP | Low | Probable | Medium | P4 | 3 |
| STD-008-CPP | Medium | Unlikely | Medium | P4 | 3 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | 3 |
| STD-010-CPP | High | Probable | High | P6 | 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 | This refers to the process of securing data by encrypting it and ensuring it is stored in the storage system. It protects against unauthorized access, and data breaches. Allows for encryption algorithms and keys. |
| Encryption in flight | Encrypting data while it a transmitted over a network. This is understood to protect data from unauthorized access, or interception while it is being transmitted. It can protect the data and can be part of a compliance requirement. It uses key exchange for the sender and receiver. Decryption must also be utilized on the receiving end to see and restore the data. |
| Encryption in use | This means encrypting the data as it is being used or processed. This protects data as it is currently being used either in memory or on an application. It allows for usage of the data while it remains encrypted, so it is not vulnerable. It will provide enhanced security and reduce the risk of data breaches. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This process of verifying a user and ensuring they have access. Can be utilized with password authentication, MFA, and/or biometric authentication. Providing better security and greater access control. |
| Authorization | After the user has been authenticated it will assess what that user has permission to access. This will assign specific permissions and rights to certain resources based on what the user can or should be able to access. |
| Accounting | Accounting involves the tracking of a user and their activity to include what data they have or should access, and how long. It can track and audit a user’s actions, detect suspect activity, and log this information. This can enhance security analysis to better understand system history. |

**\***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 |