**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 | This principle ensures that there is a validation on all input data received. This can avoid general vulnerabilities. |
| 1. Heed Compiler Warnings | This principle ensures that a user always uses the highest warning that can be chosen in a code editor, since it’s important to finish sans errors. This can be fixed by the user or by utilizing static and dynamic analysis tubes. Since a simple error can result in a security defect, it’s important to pay attention to compiler warnings. |
| 1. Architect and Design for Security Policies | This ensures that code is designed with security principles in mind to maintain safety. |
| 1. Keep It Simple | Keeping the mechanisms simple, since complex algorithms can result in more errors. |
| 1. Default Deny | By keeping the default setting as denying access to everyone makes the code access be on a permission basis. Special access should only by permission only. |
| 1. Adhere to the Principle of Least Privilege | All jobs that are required should be permitted by the execution of the code. To explain better, only give the least amount of privilege necessary for completion. If elevated permissions are given, it could be a security vulnerability. |
| 1. Sanitize Data Sent to Other Systems | When passing the data through the systems, especially complex ones, ensure that the data is sanitized. This will ensure that someone cannot use a simple injection attack to access and perform commands outside of the intended use. |
| 1. Practice Defense in Depth | This is a principle that understands that each line of defense has vulnerabilities. If there are multiple lines of defense, this can ensure that one of the lines of defense will stop a potential attack. |
| 1. Use Effective Quality Assurance Techniques | With Effective Quality Assurance techniques, the ability to identify and later remove bugs in code, such as security flaws, is increased. With multiple testing phases, and having multiple people check the code, having third parties check it over, the system can be ensured that it is as safe as possible. |
| 1. Adopt a Secure Coding Standard | With a strong and secure coding standard, this will ensure a baseline of security in the code, no matter how the code is written. |

### 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 cast to an out-of-range enumeration value. |

| **Noncompliant Code** |
| --- |
| This code attempts to check if a given value is within the enumeration value ranges that are acceptable. In this example, the code will do so after casting to the enumeration, which can hinder its representation of the given integer value. |
| enum EnumType {  One,  Two,  Three };  void func(intVariable) {  EnumTye enumVar = static\_cast<EnumType> (intVariable);  if (enumVar < One || enumVar > Three) {  }} |

| **Compliant Code** |
| --- |
| This code checks that a value has the ability to be represented by the enumeration type prior to the conversion. The ensures the conversion doesn’t represent an unspecified value. |
| enum EnumType {  One,  Two,  Three };  Void func(int intVariable) {  if (intVariable < One || intVariable > Three) {  }  EnumType enumVar = static\_case<EnumType> (intVariable); } |

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

| **Principles(s):** Values that are unspecified can result in an overflow of the buffer. This can result in the code that is not supposed to be ran by an attacker, when this is not the intention. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhais Suite | 6.9.0 | CertC++ - INT50 |  |
| PVS – Studio | 7.07 | V1016 |  |
| Parasoft C/C++ test | 2020.2 | CERT+CPP – INT50-a | An expression with enum type should only have values corresponding with the enumerators. |
| PRQA QA – C++ | 4.4 | 3013 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | For this standard, you will need to use the correct type of reference or pointers. |

| **Noncompliant Code** |
| --- |
| This example shows the loop with an error after the first iteration. |
| #include <deque>  void func(const double \*object, std::size\_t count) {  std::deque<double>num;  auto pos = num.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  num.insert(pos, object(i) + 41.0);  }  } |

| **Compliant Code** |
| --- |
| The pos doesn’t have an error because it has a valid iterator. |
| #include <deque>  void func(const double \*object, std::size\_t count) {  std::deque<double> num;  auto pos = num.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  num.insert(pos, object[i] + 41.0);  }  } |

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

| **Principles(s):** We can have unidentified behavior, if the incorrect pointers and references are used. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Overflow\_unpon\_derefence |  |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-CTR51-a | You can not modify the container while iterating over it. |
| PVS = Studio | 7.07 | V783 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not make a std::string from a null pointer. |

| **Noncompliant Code** |
| --- |
| In the following code we make a std::string object and store the output of the std:detenv() value in it. An error occurs, because it is a null pointer. |
| #include <cstdlib>  #include <string>    Void func() {  const char \*tempPointer = std::getenv("TMP");  if (!temp.empty()) {    } } |

| **Compliant Code** |
| --- |
| In this solution we check to see if std::getenv() is null and then we create the std::string if it is not null. |
| #include <cstdlib>  #include <string>    Void func() {  const char \*tempPointer = std::getenv("TMP"); std::string temp(tempPointer? tempPointer: "");  if (!temp.empty()) {    } } |

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

| **Principles(s):** If you dereference a null pointer, you will have unidentified behavior. This can cause the program to terminate randomly, which can allow attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Assert\_failure |  |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ | 2020.2 | CERT\_CPP-STR510a | Do not use null pointer dereferencing |
|  |  |  |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | If there is already-owner pointer, the smart pointer value should not be stored there. |

| **Noncompliant Code** |
| --- |
| Two smart pointers are created from the same pointer value. |
| #include <memory>  void func()  {  int \*i = new int;  std::shared\_ptr<int> ptrOne(i);  std::shared\_ptr<int> ptrTwo(i);  } |

| **Compliant Code** |
| --- |
| Both of the pointers are related, but they are not from the same value, which doesn’t cause one to delete. |
| #include <memory>  void  func() {  std::shared\_ptr<int> ptrOne = std::make\_shared<int>();  std::shared\_ptr<int> ptrTwo(ptrOne);  } |

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

| **Principles(s):** An attacker can breach when a deallocation function gets a pointer without being obtained by a alloc function. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Dangling\_pointer\_user |  |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-MEM56-a | Do not store an already owned pointer in an unrelated point |
| PVS-Studio | 7.01 | V1006 |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | All allocated resources are deallocated correctly. |

| **Noncompliant Code** |
| --- |
| The local variable is passed to new and then passed to the ::operator delete() with an error. |
| #include <iostream>  struct test {  test() {std::count <<”test::test()” <<std::endl; }  ~S() { std::count <<”test::~test()” << std::endl;}};  void func() {  alignas(struct test() char space[sizeof(struct test)];  test \*test1 = new(&space) test;  // …  Delete test1;  } |

| **Compliant Code** |
| --- |
| Remove the ::operator delete() and instead calls the destructor. |
| #include <iostream>  struct test {  test() {std::count <<”test::test()” <<std::endl; }  ~S() { std::count <<”test::~test()” << std::endl;}};  void func() {  alignas(struct test() char space[sizeof(struct test)];  test \*test1 = new(&space) test;} |

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

| **Principles(s):** If a deallocation functions receives a pointer that was not obtained by an alloc function will result in an undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Invalid\_dynamic\_memory\_allocation\_dangling\_pointer\_use |  |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ | 2020.2 | CERT\_CPP – MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d |  |
| PVS – Studio | 7.01 | V515, V554, V611, V701, V748, V773 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | A constant expressions needs to be tested with static inspection, when tested. |

| **Noncompliant Code** |
| --- |
| Use the assert() call to assert. |
| #include <assert.h>  Struct func time  Unsigned char MODE;  Unsigned int DATA;  Unsigned int COUNT;  };  Int func(void) {  Assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| A preprocessor conditional statement needs to be used when creating a static assertion. |
| Struct func timer  Unsigned char MODE;  Unsigned int DATA:  Unsigned int COUNT;  };  #if (sizeof(struct timer) != (sizeof(unsigned char) + size of (unsigned int) + sizeof(unsigned int)))  #error “Structure mused not have padding.”  #endif |

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

| **Principles(s):** Static assertion allow for errors and defects to be found in the code quickly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 |  | Users can create a check that uses the assert() function. |
| ÉCLAIR | 1.2 |  |  |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-LLL] | Before execution, resolve all exceptions. |

| **Noncompliant Code** |
| --- |
| The constructor throws an exception that is not caught. |
| Struct Test {  Test() noexcept(false);  };  Static Test globalTest; |

| **Compliant Code** |
| --- |
| GlobalTest is a local variable, which allows all exceptions. |
| Struct Test {  Test() noexcept(false);  };  Test &globalTest() {  Try {  Static Test test;  Return test;  } catch(…) {  //fix error  }  //unreachable  } |

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

| **Principles(s):** If an exception is thrown and not caught, there is a termination of the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Potentially throwing static initializing | Partially checked |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ | 2020.2 | CERT-CPERR58-a | Exceptions will be the only thing to show upon starting program |
|  |  |  |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [STD-nnn-LLL] | [Rationalize the standard.] |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| [Noncompliant code block; code should be indented using 12-point Courier New font.] |

| **Compliant Code** |
| --- |
| [Compliant description] |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**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 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Programming Object Oriented | [STD-009-CPP] | Virtual functions should not be invoked by constructors or destructors. |

| **Noncompliant Code** |
| --- |
| The base class attempts to take and release an object’s resources. |
| struct B {  B() { seize(); }  virtual ~B() { release(); }  protected:  virtual void seize();  virtual void release();  };  struct D : B {  virtual ~D() = default;  protected:  void seize() override {  B::seize();  // Get derived resources...  }  void release() override {  // Release derived resources...  B::release();  }  }; |

| **Compliant Code** |
| --- |
| The contructors call a private member function instead of a virtual function. |
|  |

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

| **Principles(s):** The constructors call for a private member function, instead of a virtual function. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axvion Bauhaus Suite | 6.9.0 | CERTC ++ OOOP50 |  |
| Clang | 3.9 | Clang-analyzer-aplha.cplusplus.VirtualCall | Checked by clang |
| LDRA tool suite | 9.7.1 | 467S, 92D | Implemented fully |
| PVS-Studio | 20.10 | Call – in customer | Checked fully |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| No positive value returned | [STD-010-CPP] | Value returning functionalities should all return a value, whether positive or negative. |

| **Noncompliant Code** |
| --- |
| There is no input value for positive input, so not all paths can return a value. |
| Int absolute\_value (int a) {  If (a<0) {  Return -a;  }} |

| **Compliant Code** |
| --- |
| All code paths return a value. |
| Int absolute\_value(int a) {  If (a < 0) {  Return -a;  }  Return a;  } |

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

| **Principles(s):** Failure to return value results in undefined behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probably | Mediume | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Bauhaus Suite |  |  |  |
| Clang | 3.9 | Wreturn type | Doesn’t capture all instances of the rule |
| CodeSonar | 6.0p0 | LANG.STRUCT.MRS | Missing return statment |
| Helix QAC | 2021.1 |  |  |

### 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 | L3 |
| STD-002-CPP | High | Probable | High | High | L2 |
| STD-003-CPP | High | Likely | Medium | Low | L1 |
| STD-004-CPP | High | Likely | Medium | Low | L1 |
| STD-005-CPP | High | Likely | Medium | Low | L1 |
| STD-006-CPP | Low | Unlikley | High | High | L3 |
| STD-007-CPP | Low | Likely | Low | Medium | L2 |
| STD-008-CPP | Low | Likely | Medium | Medium | L2 |
| STD-009-CPP | Low | Likely | Medium | High | L3 |
| STD-010-CPP | Medium | Probable | Medium | Medium | L2 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### 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 is designed to prevent an attacker from accessing the unencrypted data, because this ensures the data is encrypted on the disk. |
| Encryption in flight | Encrypting data while the data is being transmitted is this policy. |
| Encryption in use | Compromising data in use enables the access to encrypted data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process where the user is confirmed as having access to the system. This includes login and password. |
| Authorization | Authorization is the level of access a user has within said system. This can be admin, read only, delete, modify, etc. |
| Accounting | Accounting is the process where a user is monitored for what they are doing at their access level. |

**\***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 | 11/17/2024 | First Revision | Tasha Smith |  |
| 1.2 | 12/08/2024 | Final Revision | Tasha Smith |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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