**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 | Always validate and verify the input data of a system to ensure it is safe. Doing this can help prevent input vulnerabilities from being exploited. IT is also important to confirm the input data meets constraints before process. |
| 1. Heed Compiler Warnings | The warnings point to insecure code or bugs. Fixing these warning is pivotal to ensuring security of the application as well as ensuring expected behavior is present. |
| 1. Architect and Design for Security Policies | It is much harder to implement security after developing a product, so it is important to have built in a security plan from the beginning. This will allow for creation of the most secure application possible. |
| 1. Keep It Simple | The simpler the code/system, the easier it will be for developers to understand and maintain. This will also keep the vulnerabilities to a minimum based on the fact there is less code to attack. |
| 1. Default Deny | This ensures only those who have credentials can get into the system. Reducing the risk of unauthorized access will improve the security of an application greatly. |
| 1. Adhere to the Principle of Least Privilege | Allowing users to access and edit the data that is pertained to their job reduces the risk of having a security breach. |
| 1. Sanitize Data Sent to Other Systems | Ensuring that the data from outside databases and applications is sanitized and processed will prevent injection attacks. |
| 1. Practice Defense in Depth | Security should be more than one layer thick. The more layer to security the better. If one layer fails, others still provide protection. |
| 1. Use Effective Quality Assurance Techniques | Testing such as automated, regression and static analysis should be used to ensure a quality product makes it to the customers. |
| 1. Adopt a Secure Coding Standard | Following a recognized secure coding standard is both consistent and less apt to present vulnerabilities. Standards provide guidelines for secure programming practices and help impose security on development teams and others |

### 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** | INT31-CPP. | Do not assign a floating-point value to an integral type |

| **Noncompliant Code** |
| --- |
| If you assign a double value to int without casting problems with operation and security flaws appear |
| Double pi = 3.14;  Int value = pi; |

| **Compliant Code** |
| --- |
| Casting the double to int makes the conversion safe and secure. |
| double pi = 3.14;  int value = static\_cast<int>(pi); |

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

| **Principles(s):**   1. Least Privilege - By preventing data conversions you are guarded against unsafe operations. 2. Separation of duties - All conversions should hold separate values and those values should make sense for the numeric domians. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | medium | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 LTS | Cpp:S124 | Flags implicate type versions between floats and integer types. |
| Cling-tidy | 15.0 | Clang-analyzer-core | Gives warning for narrow conversions. |
| Coverity | 2024.4 | FLOATING\_POINT\_To\_INT | Flags illegal FLOAT\_TO\_INT as risky |
| Cppcheck | 2.10 | conversionLoss | See’s data loss in type conversions |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | EXP34-CPP | Validate inputs to avoid out-of-bounds array access. |

| **Noncompliant Code** |
| --- |
| Allowing user input without validation can cause unwanted failures. |
| int index = std::stoi(userInput);  array[index] = value; |

| **Compliant Code** |
| --- |
| Validaing this input can ensure the application is able to run smoothly without error. |
| int index = std::stoi(userInput);  if (index >= 0 && index < arraySize) {  array[index] = value;  } |

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

| **Principles(s):**  5) Validate all input – validate all ser inputs so you can make sure there are no unwanted security risks.  9) DiD – input validation is the frontmost layer of defense in depth. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 LTS | Cpp:S3518 | Senses the missing input validation. |
| Clang-Tidy | 15 | Clang-analyzer-core.NullDereference,core.builtin.BuiltinFunctions | Flags improper use of user input |
| Coverity | 2024.4 | ARRAY\_INDEX\_OUT\_OF\_BOUNDS | Finds access to invalid memory |
| CppCheck | 2.10 | arrayIndexOutOfBounds | Sees out of bounds access. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR31-CPP | Do not use unsafe string copy functions without bounds checking. |

| **Noncompliant Code** |
| --- |
| Using strncpy without checking the size of the buffer at the destination is unsafe. |
| char dest[10];  strcpy(dest, userInput); |

| **Compliant Code** |
| --- |
| Always use proper bounds when using strncpy |
| char dest[10];  strncpy(dest, userInput, sizeof(dest) - 1);  dest[sizeof(dest) - 1] = '\0'; |

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

| **Principles(s):**   1. Secure The Weakest Link – Legacy libraries like strcpy must be replaced with new and improved libraries.   9) Protects against buffer overflow |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 LTS | Cpp:S3518 | Detects unsafe use of strcpy |
| Clang-Tidy | 15.0 | Clang-analyzer-secuirty.insecureAPI.strcpy | Suggests replacements when strcpy is used. |
| Coverity | 2024.4 | Buffer\_size, STRING\_OVERFLOW | Flags buffer overflow and unsafe string functions. |
| Cppcheck | 2.10 | bufferAccessOutOfBounds | Reports buffer overrun risks from unsafe memory operations |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | **SQL Injection Prevention:** preventing injection will prevent unauthorized access to the system. |

| **Noncompliant Code** |
| --- |
| Concatenating user input directly into SQL queries can lead to SQL injection. |
| std::string query = "SELECT \* FROM users WHERE name = '" + userInput + "'";  executeQuery(query); |

| **Compliant Code** |
| --- |
| Using prepared statements will solve this vulnerability. |
| PreparedStatement stmt = connection.prepareStatement("SELECT \* FROM users WHERE name = ?");  stmt.setString(1, userInput);  stmt.executeQuery(); |

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

| **Principles(s):**  **5) Validate All Inputs** – Inputs are implicitly validated  7) Avoid Security by Obscurity – Relies on structured and proven security practices |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Critical | Very Likely | Low | Critical | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 LTS | cpp:S3649 | See’s unsafe SQL quarry and recommends alternatives. |
| Coverity | 2024.4 | SQL\_INJECTION | Identifies things vulnerable to injection. |
| Fortify SCA | 23.1 | SQL Injection: Poor Validation | Detects lack of input sanitization. |
| CodeQL | 13.5 | cpp/sql-injection | Query detects patterns where SQL injection is risky |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM52-CPP | Do not leak memory. Use RAII techniques or smart pointers. |

| **Noncompliant Code** |
| --- |
| Failing to deallocate dynamic allocated memory |
| int\* data = new int[100]; |

| **Compliant Code** |
| --- |
| Using smart pointers to automatically manage memory. |
| std::unique\_ptr<int[]> data(new int[100]); |

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

| **Principles(s):**  10) Simplicity - Smart pointers simplify code and reduce memory errors.  6) Separation of duties - Let smart pointers handle memory management |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 15.0 | clang-analyzer-cplusplus.NewDeleteLeaks | Detects unallocated memory. |
| SonarQube | 9.9 LTS | cpp:S1251 | Detects raw pointer allocation without deallocation |
| Cppcheck | 2.10 | memleak | Flags memory leaks |
| Coverity | 2024.4 | RESOURCE\_LEAK, UNCAUGHT\_EXCEPT | Identifies memory/resource leaks via exception paths. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | INT32-CPP | Do not perform arithmetic operations that overflow. |

| **Noncompliant Code** |
| --- |
| Arithmetic operations can silently wrap |
| int total = INT\_MAX;  total += 1; |

| **Compliant Code** |
| --- |
| Using safe integer operations or overflow checks |
| #include <limits>  if (total < std::numeric\_limits<int>::max()) {  total += 1;  } else {  } |

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

| **Principles(s):**  8) Keep Security Simple - Avoiding overflow with straightforward checks  6) Separation of duties - Ensures operations that manipulate data types are explicitly checked |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 15.0 | cert-int32-c | Flags operations that cause wraparound |
| SonarQube | 9.9 LTS | cpp:S2185, cpp:S2755 | Detects arithmetic overflow vulnerabilities |
| Cppcheck | 2.10 | integerOverflow | Identifies risky operations |
| Coverity | 2024.4 | INTEGER\_OVERFLOW | Analysis to detect arithmetic overflow in various contexts. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | MEM34-CPP | Use delete[] to deallocate memory allocated with new[] |

| **Noncompliant Code** |
| --- |
| Delete use with arrays can lead to unexpected behavior. |
| int\* data = new int[5];  delete data; |

| **Compliant Code** |
| --- |
| Fixing the use of delete will solve this issue. |
| int\* data = new int[5];  delete[] data; |

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

| **Principles(s):**  1) Least Privilege – Properly deleting memory ensure proper allocation  2) Defense in Depth – Prevents memory mismanagement |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 15.0 | clang-analyzer-cplusplus.NewDeleteLeaks | Flags misuse of delete functions |
| SonarQube | 9.9 LTS | cpp:S5843 | Warns about unsafe delete operations |
| Cppcheck | 2.10 | memleak | Detects memory mismanagement |
| Coverity | 2024.4 | DELETE\_MISMATCH | Flags New while paired with delete |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Input Validation** | FIO50-CPP | Do not use untrusted input to construct file paths. |

| **Noncompliant Code** |
| --- |
| Concatenating user input directly into a file path allows directory traversal. |
| std::ifstream file("/home/data/" + userInput); |

| **Compliant Code** |
| --- |
| Sanitizing all of the input will mitigate this issue. |
| if (userInput.find("..") == std::string::npos) {  std::ifstream file("/home/data/" + userInput);  } |

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

| **Principles(s):** [  5) Validate All Inputs - Ensures input cannot be used to access unintended areas of SW  3) Secure Defaults - The default behavior should prevent path traversal. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Critical | Very Likely | Low | Critical | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 LTS | cpp:S2076, cpp:S2674 | Detects unwanted input in file paths |
| Coverity | 2024.4 | TAINTED\_STRING | Detects bad input in filesystem |
| Fortify SCA | 23.1 | Path Manipulation | Flags risky path construction |
| CodeQL | 13.5 | cpp/path-traversal | Detects file path construction from user input. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Use Approved Libraries** | MSC50-CPP | Do not use std::rand() for generating pseudorandom numbers |

| **Noncompliant Code** |
| --- |
| Using outdated libraries with known vulnerabilities. |
| #include <cstdlib>  int value = std::rand(); |

| **Compliant Code** |
| --- |
| Use modern, maintained, and secure libraries |
| #include <random>  std::random\_device rd;  std::mt19937 gen(rd());  std::uniform\_int\_distribution<> dis(1, 100);  int value = dis(gen); |

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

| **Principles(s):**  10) Simplicity – Modern libraries simplify random functions  8) Keep Security Simple – Approved libraries provide better security |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 LTS | cpp:S2245 | Flags unsafe random generators |
| Clang-tidy | 15 | cert-msc50-cpp | Warns about bad rand() usage |
| Coverity | 2024.4 | INSECURE\_RANDOM | Flags weak RNG |
| Cppcheck | 2.10 | insecureRandom | Detects use of rand() in cryptographic |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exception Safety** | ERR60-CPP | Exception objects must be no throw copy constructible |

| **Noncompliant Code** |
| --- |
| Failing to handle exceptions can cause program crashes |
| void riskyOperation() {  throw std::runtime\_error("Failure");  } |

| **Compliant Code** |
| --- |
| Catch exceptions and perform cleanup |
| void process() {  try {  riskyOperation();  } catch (const std::exception& e) {  std::cerr << "Error: " << e.what() << std::endl;  }  } |

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

| **Principles(s):**  **2) Fail-Safe Defaults -** Failures should be caught before they crash the program.  **10) Simplicity - Structured error handling simplifies logic** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 15 | cert-err60-cpp | Detects throwable objects that aren’t no except copyable. |
| SonarQube | 9.9 LTS | cpp:S3626 | Flags bad exception handling patterns |
| Coverity | 2024.4 | UNCAUGHT\_EXCEPT | Looks for missing exception handling |
| Cppcheck | 2.10 | uncaughtException | Flags places where exceptions may go uncaught. |

### 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.

*Green Pace uses automation throughout its DevSecOps pipeline, to ensure that the standards of coding are maintained, and security is integrated from day one. During the planning and design phases, tools are ran that check coding for known vulnerabilities to help identify threats as early as possible. During development, static analyzers, such as SonarQube or Clang-Tidy, are used within an IDE so they can report common problems like memory leaks or improper input handling before the code ever leaves the developer's machine.*

*Moving to build and test stages, automation tools run security hazard inspections every push and merge; tools such as SonarQube, Coverity, or Cppcheck are fit for this task. We then further unit test and fuzz test to catch edge cases and to assure handling exceptions and input validation are functioning from the edge.*

*The intention is to certify at the epic stage that something encrypts correctly, enforces access control, and cleans up memory. Real-time threat detection and quick responses are handled by monitoring tools. Such an automated setup will keep us compliant but adds with the layers of security that we built with a defense-in-depth strategy.*

### 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 |
| --- | --- | --- | --- | --- | --- |
| INT31-CPP. | Medium | Likely | Medium | High | 2 |
| EXP34-CPP | High | Likely | Low | High | 1 |
| STR31-CPP | High | Likely | Low | High | 1 |
| STD-004-CPP | Critical | Very Likely | Low | Critical | 1 |
| MEM52-CPP | Medium | Likely | Low | High | 2 |
| INT32-CPP | Medium | Likely | Low | Medium | 2 |
| MEM34-CPP | High | Likely | Low | High | 1 |
| FIO50-CPP | Critical | Very Likely | Low | Critical | 1 |
| MSC50-CPP | Medium | Likely | Low | Medium | 2 |
| ERR60-CPP | Medium | Unlikely | Medium | Medium | 3 |

### 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 | All sensitive data stored on any disk or long-term storage device must be encrypted by industry best standards. |
| Encryption in flight | All data that is transferred or started via network must be encrypted by industry best standards. |
| Encryption in use | All data that is sensitive must be protected from unauthorized access on the system. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | All systems must include user credentials. |
| Authorization | The principle of least privilege must be followed. |
| Accounting | System must be monitored and audited. |

**\***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 |  |
| 2.1 | 06/15/2025 | Project 1 | Chandler Duhamel | Chandler Duhamel |
| [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 |