**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 | Validating input data ensures that only expected and safe data enters the system, preventing injection attacks, buffer overflows, and other vulnerabilities caused by malicious or malformed input. This principle is fundamental to maintaining the integrity and security of applications by rejecting invalid data early in the process. |
| 1. Heed Compiler Warnings | Compiler warnings often highlight potential issues such as uninitialized variables or type mismatches, which can lead to undefined behavior. Addressing them improves code reliability and security by catching errors that could otherwise become exploitable vulnerabilities. |
| 1. Architect and Design for Security Policies | Incorporating security considerations into the architecture and design phase ensures that security is built in rather than bolted on, leading to more robust systems that adhere to policies like least privilege and defense in depth from the outset. |
| 1. Keep It Simple | Complex code is harder to understand, maintain, and secure. Simplicity reduces the likelihood of errors and makes it easier to spot vulnerabilities, thereby enhancing overall code quality and reducing the attack surface. |
| 1. Default Deny | By denying access or operations by default and only allowing what is explicitly permitted, this principle minimizes the attack surface and prevents unauthorized actions, ensuring that only verified and necessary operations proceed. |
| 1. Adhere to the Principle of Least Privilege | Granting only the minimal permissions necessary for tasks reduces the potential damage from compromised components or users, limiting the scope of any breach and enhancing system security. |
| 1. Sanitize Data Sent to Other Systems | Cleaning data before sending it to external systems prevents injection attacks and ensures compatibility, protecting against exploitation in interconnected environments by removing potentially harmful elements. |
| 1. Practice Defense in Depth | Implementing multiple layers of security controls ensures that if one layer fails, others can still protect the system, providing comprehensive protection against a wide range of threats. |
| 1. Use Effective Quality Assurance Techniques | Employing thorough testing, code reviews, and static analysis helps identify and fix vulnerabilities early in the development cycle, improving the overall security and reliability of the software. |
| 1. Adopt a Secure Coding Standard | Following established secure coding standards like SEI CERT ensures consistent application of best practices to avoid common pitfalls and vulnerabilities, promoting secure development across the team. |

### 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 access a cv-qualified object through a cv unqualified type |

| **Noncompliant Code** |
| --- |
| Modifying a const-qualified object through a non-const reference leads to undefined behavior. |
| void g(const int &ci) { int &ir = const\_cast |

| **Compliant Code** |
| --- |
| Pass a modifiable reference instead of casting away const. |
| void g(int &i) { i = 42; } |

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

| **Principles(s):** 1 (Validate Input Data) - Ensures types are respected to prevent invalid data access; 2 (Heed Compiler Warnings) - Compilers often warn about type mismatches; 8 (Practice Defense in Depth) - Proper type handling adds a layer of protection against misuse. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.6 | EXP55-CPP | Detects access to cv-qualified objects through unqualified types. |
| Coverity | 2023.12 | MISRA C++:2008 | Checks for type qualifier mismatches. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| Reading from uninitialized variables can lead to undefined behavior. |
| int x; use(x); |

| **Compliant Code** |
| --- |
| Ensure variables are initialized before use. |
| int x = 0; use(x); |

**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** |
| --- | --- | --- | --- | --- |
| High | Probable | Low | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.6 | EXP53-CPP | Detects reads from uninitialized memory. |
| Clang Static Analyzer | 16.0 | core.uninitialized | Identifies uninitialized value usage. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| Guarantee that storage for strings has sufficient space for character data and the null terminator |
| char buf[10]; strcpy(buf, "longerstring"); |

| **Compliant Code** |
| --- |
| Ensure buffers are large enough including null terminator. |
| char buf[11]; strcpy(buf, "longstring"); |

**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** |
| --- | --- | --- | --- | --- |
| High | Probable | Low | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.6 | STR50-CPP | Checks string storage sufficiency. |
| Coverity | 2023.12 | BUFFER\_SIZE | Detects potential buffer overflows in strings. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Sanitize untrusted data passed across a trust boundary to prevent injection |

| **Noncompliant Code** |
| --- |
| Concatenating user input directly into SQL queries allows injection. |
| std::string query = "SELECT \* FROM users WHERE name = '" + user\_input + "'"; |

| **Compliant Code** |
| --- |
| Use parameterized queries to sanitize input. |
| // Using a library like SOCI or ODBC with prepared statements:  stmt << "SELECT \* FROM users WHERE name = ?", use(user\_input); |

**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** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.6 | S2077 | Detects SQL injection vulnerabilities. |
| Coverity | 2023.12 | SQL\_INJECTION | Identifies unsanitized inputs in queries. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Provide placement new with properly aligned pointers to sufficient storage capacity |

| **Noncompliant Code** |
| --- |
| Passing misaligned or insufficient storage to placement new results in undefined behavior. |
| char c; unsigned char buffer[sizeof(long)]; long \*lp = ::new (buffer) long; |

| **Compliant Code** |
| --- |
| Use alignas for proper alignment. |
| alignas(long) unsigned char buffer[sizeof(long)]; long \*lp = ::new (buffer) long; |

**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** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.6 | MEM54-CPP | Checks placement new alignment. |
| Clang | 16.0 | alignment | Warns on misaligned allocations. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Never use assertions to validate method arguments |

| **Noncompliant Code** |
| --- |
| Using assert for runtime input validation, which may be disabled in release builds. |
| void func(int x) { assert(x > 0); /\* use x \*/ } |

| **Compliant Code** |
| --- |
| Use explicit checks or exceptions for validation. |
| void func(int x) { if (x <= 0) throw std::invalid\_argument("x must be positive"); /\* use x \*/ } |

**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** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Low | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.6 | S5685 | Detects assertions on arguments. |
| Coverity | 2023.12 | ASSERT\_SIDE\_EFFECT | Checks for improper assert usage. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Catch exceptions by lvalue reference |

| **Noncompliant Code** |
| --- |
| Catching by value causes object slicing. |
| try { throw S(); } catch (std::exception e) { std::cout << e.what() << std::endl; } |

| **Compliant Code** |
| --- |
| Catch by reference to avoid slicing. |
| try { throw S(); } catch (std::exception &e) { std::cout << e.what() << std::endl; } |

**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** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.6 | ERR61-CPP | Detects catch by value. |
| Clang | 16.0 | performance-move-const-arg | Warns on inefficient catches. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Do not use std::rand() for generating pseudorandom numbers |

| **Noncompliant Code** |
| --- |
| std::rand() has poor quality and bias. |
| std::string id("ID"); id += std::to\_string(std::rand() % 10000); |

| **Compliant Code** |
| --- |
| Use std::mt19937 for better randomness. |
| std::uniform\_int\_distribution |

**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** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.6 | MSC50-CPP | Detects use of std::rand(). |
| Coverity | 2023.12 | SECURITY.RAND | Flags insecure random functions. |
| [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** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | Use valid iterator ranges |

| **Noncompliant Code** |
| --- |
| Using invalid iterator ranges can lead to overflows. |
| // Assume invalid range: std::vector  std::vector v; auto it = v.begin() + 10; \*  it = 5 |

| **Compliant Code** |
| --- |
| Ensure ranges are valid. |
| std::vector v(10);  auto it = v.begin() + 5;  \*it = 5; |

**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** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.6 | CTR53-CPP | Checks iterator validity. |
| Valgrind | 3.22 | Memcheck | Detects invalid accesses at runtime. |
| [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** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Avoid cycles during initialization of static objects |

| **Noncompliant Code** |
| --- |
| Recursive static object initialization causes undefined behavior. |
| static A a;  static B b(a);  // Assuming B depends on A and vice versa |

| **Compliant Code** |
| --- |
| Ensure no circular dependencies. |
| static A a = initA();  static B b = initB(a);  // Non-circular |

**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** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.6 | DCL56-CPP | Detects static init cycles. |
| GCC | 14.1 | -Winit-self | Warns on self-init, related to cycles. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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.

To integrate automation into the existing DevOps process, we will modify it to incorporate DevSecOps practices as illustrated in the toolchain diagram. In the "Create" phase, IDE plugins like SonarLint will provide real time feedback on coding standards during development. During "Verify," static analysis tools such as SonarQube and Coverity will be run in the CI pipeline to scan code for compliance with the defined standards, halting builds if violations are found. In "Preprod," dynamic testing tools like Valgrind will check for runtime issues such as memory leaks. For "Prevent" and "Detect," runtime application self protection and monitoring tools will ensure ongoing compliance. This continuous loop ensures security is enforced automatically at every stage, reducing manual effort and catching issues early.

### 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 | Medium | Probable | Medium | P8 | L2 |
| STD-002-CPP | High | Probable | Low | P18 | L1 |
| STD-003-CPP | High | Probable | Low | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Medium | Probable | Low | P12 | L1 |
| STD-007-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-008-CPP | Low | Unlikely | Low | P6 | L2 |
| STD-009-CPP | High | Probable | High | P6 | L2 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 | L3 |

### 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 | Encryption at rest protects stored data from unauthorized access by encrypting it on disks or databases using algorithms like AES-256. It should be applied to all sensitive data such as user information in databases or files, and is required whenever data is persisted to prevent breaches if storage media is stolen or accessed improperly. This policy applies to comply with data protection regulations and as part of defense in depth. |
| Encryption in flight | Encryption in flight secures data during transmission over networks using protocols like TLS 1.3. It must be used for all network communications involving sensitive data, such as API calls or database connections, to prevent interception by attackers. The policy applies to all external and internal transmissions to ensure confidentiality and integrity. |
| Encryption in use | Encryption in use safeguards data while it is being processed in memory, using techniques like homomorphic encryption or secure enclaves (e.g., Intel SGX). It is applied when processing highly sensitive data in untrusted environments, such as cloud computations, to prevent exposure during runtime. This policy is enforced for critical applications handling confidential information to maintain protection throughout the data lifecycle. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication verifies the identity of users or systems using methods like multi-factor authentication (MFA) and passwords. It should be applied to all user logins and API accesses, logging attempts for auditing. The policy applies to prevent unauthorized access, ensuring only verified entities interact with the system. |
| Authorization | Authorization determines what authenticated entities can do, using role-based access control (RBAC) to check user levels of access before operations like database changes or file access. It is enforced after authentication for all actions, to adhere to least privilege and prevent privilege escalation. |
| Accounting | Accounting logs all activities, including user logins, changes to the database, addition of new users, user level of access changes, and files accessed by users. It uses centralized logging tools for auditing and is applied continuously to provide traceability, detect anomalies, and support incident response. |

**\***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.] | 08/10/2025 | Completed all of the coding standards, encryption, Triple A, and Automation. | Seth Porter | CISO and CIO |
| [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 |