**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 | Explain the importance of input validation to prevent unexpected inputs like SQL injection, XSS, buffer overflows. |
| 1. Heed Compiler Warnings | Describe how compiler warnings help identify risky behavior or deprecated features early. |
| 1. Architect and Design for Security Policies | Emphasize planning security from the start of the design, not just during implementation. |
| 1. Keep It Simple | Security benefits from simplicity. Fewer bugs, easier maintenance. |
| 1. Default Deny | Access should be denied unless explicitly allowed. |
| 1. Adhere to the Principle of Least Privilege | Programs/users should operate using the least set of privileges needed. |
| 1. Sanitize Data Sent to Other Systems | Protect outbound data to prevent injection attacks in interconnected systems. |
| 1. Practice Defense in Depth | Use multiple layers of security (e.g., validation, firewalls, permissions). |
| 1. Use Effective Quality Assurance Techniques | Explain testing methods like static analysis, fuzzing, penetration testing. |
| 1. Adopt a Secure Coding Standard | State that coding must follow vetted standards like SEI CERT C++ for consistency and security. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Data Type |

| **Noncompliant Code** |
| --- |
| Improper use of data type leading to truncation. |
| int main() {  short x = 32767;  x = x + 1;  printf("%d\n", x); // May overflow  return 0;  } |

| **Compliant Code** |
| --- |
| Use of appropriate data type prevents overflow. |
| int main() {  int x = 32767;  x = x + 1;  printf("%d\n", x);  return 0;  } |

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

| **Principles(s):** 2. Heed Compiler Warnings, 4. Keep It Simple, 10. Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | |  | | --- | | 17.0+ |  |  | | --- | |  | | |  | | --- | | modernize-use-auto, cert-str34-c |  |  | | --- | |  | | |  | | --- | |  |   Detects unsafe type usage. |
| [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 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Validating data values prevents logic errors, crashes, and misuse of functions expecting specific value ranges. |

| **Noncompliant Code** |
| --- |
| No validation on user input range. |
| void setAge(int age) {  printf("Age set to %d\n", age);  } |

| **Compliant Code** |
| --- |
| Validates that age is in a realistic range. |
| void setAge(int age) {  if (age > 0 && age < 130)  printf("Age set to %d\n", age);  else  printf("Invalid age\n");  } |

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

| **Principles(s):** 1. Validate Input Data, 3. Architect and Design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.4 | cpp:S5701 | |  | | --- | | Warns about lack of input validation. |  |  | | --- | |  | |
| [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 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Safe string handling prevents buffer overflows and memory corruption. |

| **Noncompliant Code** |
| --- |
| Buffer overflow due to unsafe string copy. |
| char buf[10];  strcpy(buf, "thisisverylong"); |

| **Compliant Code** |
| --- |
| Safe copy using strncpy and buffer size limit. |
| char buf[10];  strncpy(buf, "thisisverylong", sizeof(buf) - 1);  buf[9] = '\0'; |

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

| **Principles(s):** 1. Validate Input Data, 8. Practice Defense in Depth |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Fortify SCA | 23.1 | Buffer Overflow rule | Flags unsafe strcpy/strncpy use. |
| [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 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Parameterized queries prevent malicious data from being interpreted as SQL code. |

| **Noncompliant Code** |
| --- |
| SQL query built with unsanitized input. |
| sprintf(query, "SELECT \* FROM users WHERE name='%s';", input); |

| **Compliant Code** |
| --- |
| Uses parameterized query to avoid injection. |
| sqlite3\_prepare\_v2(db, "SELECT \* FROM users WHERE name = ?", -1, &stmt, NULL);  sqlite3\_bind\_text(stmt, 1, input, -1, SQLITE\_STATIC); |

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

| **Principles(s):** 1. Validate Input Data, 7. Sanitize Data Sent to Other Systems |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeQL | Latest | cpp/sql-injection | Detects SQLi vulnerabilities via taint tracking. |
| [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 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Proper memory handling prevents leaks and access violations. |

| **Noncompliant Code** |
| --- |
| Memory allocated but not freed. |
| char \*data = malloc(100);  strcpy(data, "Hello");  // no free |

| **Compliant Code** |
| --- |
| Memory properly deallocated. |
| char \*data = malloc(100);  strcpy(data, "Hello");  free(data); |

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

| **Principles(s):** 6. Least Privilege, 8. Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.21.0 | N/A | Runtime tool for detecting memory leaks and unfreed memory. |
| [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 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Assertions verify assumptions and help detect logic errors during development. |

| **Noncompliant Code** |
| --- |
| No check on divide-by-zero. |
| void process(int val) {  int result = 100 / val;  } |

| **Compliant Code** |
| --- |
| Assertion ensures non-zero value. |
| #include <assert.h>  void process(int val) {  assert(val != 0);  int result = 100 / val;  } |

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

| **Principles(s):** 2. Heed Compiler Warnings, 9. Use Effective QA Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.13 | missingAssert | Detects missing or unsafe assertions. |
| [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 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Proper error handling prevents crashes and unexpected behavior. |

| **Noncompliant Code** |
| --- |
| Divide-by-zero not handled. |
| int divide(int a, int b) {  return a / b;  } |

| **Compliant Code** |
| --- |
| Exception thrown for invalid input. |
| int divide(int a, int b) {  if (b == 0) throw "Divide by zero error";  return a / b;  } |

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

| **Principles(s):** 5. Default Deny, 10. Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.4 | cpp:S3516 | Catches code with missing exception handling. |
| [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 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Pointer Safety | STD-008-CPP | Safe pointer operations prevent segmentation faults and memory corruption. |

| **Noncompliant Code** |
| --- |
| Null pointer dereference. |
| int \*ptr = NULL;  \*ptr = 10; |

| **Compliant Code** |
| --- |
| Checks for null before dereferencing. |
| int \*ptr = NULL;  if (ptr != NULL) {  \*ptr = 10;  } |

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

| **Principles(s):** 4. Keep It Simple, 1. Validate Input Data |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Possible | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | 17+ | core.NullDereference | |  | | --- | | Detects null pointer dereferencing. |  |  | | --- | |  | |
| [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** |
| --- | --- | --- |
| Integer Overflow | STD-009-CPP | Prevents arithmetic operations from exceeding type limits. |

| **Noncompliant Code** |
| --- |
| Possible overflow in multiplication. |
| int a = 1000000;  int b = 1000000;  int c = a \* b; |

| **Compliant Code** |
| --- |
| Checks before performing multiplication. |
| #include <limits.h>  if (a <= INT\_MAX / b) {  int c = a \* b;  } |

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

| **Principles(s):** 2. Heed Compiler Warnings, 10. Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2024.3 | Integer Overflow checker | Identifies risky multiplication, addition operations. |
| [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** |
| --- | --- | --- |
| File Handling | STD-010-CPP | Ensures file access is secure and no resources are leaked. |

| **Noncompliant Code** |
| --- |
| File pointer used without checking. |
| FILE \*fp = fopen("data.txt", "r");  fscanf(fp, "%s", buffer); |

| **Compliant Code** |
| --- |
| Checks for NULL before accessing file. |
| FILE \*fp = fopen("data.txt", "r");  if (fp != NULL) {  fscanf(fp, "%s", buffer);  fclose(fp);  } |

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

| **Principles(s):** 3. Architect for Security, 6. Least Privilege |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.2 | FILE\_NULL\_DEREF | Warns if file handles aren't checked for NULL. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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.

Automation will be implemented throughout the secure development lifecycle at Green Pace to enforce compliance and minimize manual intervention. Key points of integration include:

* **Plan Stage:** Security requirements and static code scanning rules will be integrated into development plans. Security debt will be tracked as part of sprint deliverables.
* **Create Stage:** Developers will install IDE plugins (like SonarLint, Clangd extensions) to catch issues early.
* **Verify Stage:** CI/CD pipelines will run tools like SonarQube, CodeQL, and Clang-Tidy to automatically detect compliance violations. Pull requests will be blocked if any critical rule is violated.
* **Pre-Production:** Fuzzing tools and dynamic scanners (like OWASP ZAP) will test application stability and surface zero-day logic issues.
* **Release:** Software will be signed, and integrity verified before being deployed to staging or production.
* **Detect & Respond:** RASP tools and monitoring systems will detect abnormal behaviors and log them for auditing and incident response.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | Medium | Possible | Low | Medium | 3 |
| STD-003-CPP | High | Likely | Medium | High | 5 |
| STD-004-CPP | High | Likely | Low | High | 5 |
| STD-005-CPP | Medium | Possible | Medium | Medium | 3 |
| STD-006-CPP | Low | Unlikely | Low | Low | 1 |
| STD-007-CPP | Medium | Possible | Medium | Medium | 3 |
| STD-008-CPP | High | Possible | Medium | High | 4 |
| STD-009-CPP | High | Likely | Low | High | 5 |
| STD-010-CPP | Medium | Possible | Low | Medium | 3 |
| [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.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 | **What it is:** Encryption applied to data that is stored on disk (e.g., databases, file systems).  **How it’s used:** Encrypting sensitive data using AES-256 before writing it to disk. Disk encryption tools like BitLocker, or database-level encryption like TDE (Transparent Data Encryption), are commonly used.  **Why/When policy applies:** Prevents unauthorized access if physical storage is compromised. Required by compliance frameworks (e.g., HIPAA, GDPR). Applies during system shutdown, theft, or improper decommissioning of hardware. |
| Encryption in flight | **What it is:** Protects data being transmitted over networks from interception and tampering.  **How it’s used:** TLS 1.2+ is used to encrypt HTTP traffic (HTTPS). VPNs and SSH are used for administrative connections.  **Why/When policy applies:** Ensures confidentiality and integrity during transmission between clients, servers, APIs, and external services. Prevents man-in-the-middle attacks. |
| Encryption in use | **What it is:** Protects data while it is actively being processed in memory.  **How it’s used:** Techniques include memory isolation, hardware-based encryption (Intel SGX), or secure enclaves. Access control and zero-trust execution contexts are also used.  **Why/When policy applies:** Prevents attacks such as memory scraping or cold boot attacks. Essential when sensitive data is handled in multi-tenant or cloud environments. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | **What it is:** Verifying the identity of a user or system before granting access.  **How it’s used:** MFA (Multi-Factor Authentication), secure password policies, biometric login, OAuth2.0 for services.  **Why/When policy applies:** Ensures only authorized users access systems. Required for logins, access to APIs, or admin panels. |
| Authorization | **What it is:** Defining what an authenticated user is allowed to do.  **How it’s used:** Role-based access control (RBAC), ACLs (Access Control Lists), and permission groups.  **Why/When policy applies:** Enforced when users attempt to perform actions. Prevents privilege escalation. Example: Admins can modify config, users cannot. |
| Accounting | **What it is:** Tracking and logging user actions in the system.  **How it’s used:** System logs, database logs, SIEM tools track changes to files, logins, configuration, etc.  **Why/When policy applies:** Required for forensic analysis, compliance, and monitoring insider threats. Applies during user logins, file access, system changes. |

**\***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 |  |
| 2.0 | 06/15/2025 | Completed Project One Security Policy | Musab Sagri |  |
| [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 |