**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 | Never trust external input. Check type, length, range, and format before you use it so it can’t blow up memory or logic. |
| 1. Heed Compiler Warnings | Turn warnings up, treat them like errors. They catch real bugs (narrowing, uninit vars, UB) early instead of in prod. |
| 1. Architect and Design for Security Policies | Build auth, authz, and data validation into the design—not bolted on later when it’s 10x harder. |
| 1. Keep It Simple | Simpler code > fewer surprises. Avoid clever magic that future,you won’t understand (attackers love complexity). |
| 1. Default Deny | Block by default, allow only what’s explicitly safe (inputs, permissions, network). Least surprise, fewer gaps. |
| 1. Adhere to the Principle of Least Privilege | Processes, users, and tokens only get what they need. If something gets popped, damage stays contained. |
| 1. Sanitize Data Sent to Other Systems | Encode/escape by *context* (SQL, HTML, shell) so your output isn’t a payload. |
| 1. Practice Defense in Depth | Layer controls-validation, escaping, auth, rate limits, logging. One control fails, others still help. |
| 1. Use Effective Quality Assurance Techniques | Code reviews, static analysis, unit/integration tests, fuzzing where it makes sense. Catch it before release. |
| 1. Adopt a Secure Coding Standard | Use SEI CERT C++ rules as guardrails so the whole team writes safer, consistent code.––– |

### 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   | Name: Use correct, explicit types (no accidental narrowing; prefer fixed-width/size types)  Rationale: Mismatched or vague types cause overflows, UB, and security bugs—especially at boundaries (I/O, sizes, indexes). |

| **Noncompliant Code** |
| --- |
| Narrowing a large value into short and mixing signed/unsigned. |
| int len = getUserLen(); // could be large or negative  short sz = len; // narrowing!  for (int i = 0; i < len; i++) { /\* ... \*/ } |

| **Compliant Code** |
| --- |
| Use precise, safe types and cast intentionally; use size\_t for sizes. |
| #include <cstdint>  #include <vector>  std::size\_t len = getUserLenSafe(); // validated non-negative  std::vector<int> v(len);  for (std::size\_t i = 0; i < v.size(); ++i) { /\* ... \*/ } |

**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 | Unlikely | Medium | 6 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2024.2 | BAD\_CONVERSION / INTEGER\_OVERFLOW | Flags risky casts/overflows |
| CodeSonar | 8.1p0 | LANG.TYPE.CONVERT | Detects narrowing conversions |
| Parasoft C/C++test | 2023.1 | MISRA-C++ 5-0-1 | Requires explicit conversions |
| Clang-Tidy | 17.0 | bugprone-narrowing-conversions | Warns on narrowing |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Name: Validate value ranges before use  Rationale: Out-of-range values become OOB reads/writes or logic bombs. |

| **Noncompliant Code** |
| --- |
| Uses user index directly |
| std::vector<int> scores = load();  int idx = std::stoi(userInput); // unchecked  int val = scores[idx]; // OOB risk |

| **Compliant Code** |
| --- |
| Check bounds or use .at() which throws if invalid. |
| int idx = parseInt(userInput);  if (idx < 0 || static\_cast<size\_t>(idx) >= scores.size()) return error();  int val = scores.at(static\_cast<size\_t>(idx)); |

**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 | Medium | 12 | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2024.2 | BUFFER\_OVERRUN | Out-of-range access |
| Polyspace Bug Finder | R2024a | NRS-INDX | Index/size checks |
| LDRA Tool Suite | 9.7.1 | 53 D / 69 D | Range validation rules |
| Astrée | 24.04 | Range checks | Detects OOB access |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Name: Avoid unsafe C-strings; use std::string (and bounds-checked ops)  Rationale: Raw buffers + strcpy/strcat = classic overflows. |

| **Noncompliant Code** |
| --- |
| Fixed buffer + unbounded copy |
| char buf[8];  strcpy(buf, user.c\_str()); // overflow risk |

| **Compliant Code** |
| --- |
| std::string and controlled concatenation |
| std::string s = user;  if (s.size() > 256) return error();  auto msg = "Hello " + s; |

**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 | 18 | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.MEM.BUF | Buffer misuse |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR51-a | String safety violations |
| Clang Static Analyzer | 17.0 | core.NullDereference | Null/buffer issues |
| Polyspace Bug Finder | R2024a | STR-OVR | String overflow |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Name: Use parameterized queries (never string-build SQL)  Rationale: Concatenating user input into SQL is injection 101. |

| **Noncompliant Code** |
| --- |
| Builds SQL with user input. |
| std::string sql = "SELECT \* FROM users WHERE name = '" + name + "'";  db.exec(sql); // injection risk |

| **Compliant Code** |
| --- |
| Prepared statement + bind. |
| sqlite3\_stmt\* stmt = nullptr;  sqlite3\_prepare\_v2(db, "SELECT \* FROM users WHERE name = ?", -1, &stmt, nullptr);  sqlite3\_bind\_text(stmt, 1, name.c\_str(), -1, SQLITE\_TRANSIENT);  while (sqlite3\_step(stmt) == SQLITE\_ROW) { /\* ... \*/ }  sqlite3\_finalize(stmt); |

**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 | 18 | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2024.2 | TAINTED\_STRING | Tainted concatenation |
| Semgrep | 1.64+ | generic.sql-injection | String-built queries |
| Parasoft C/C++test | 2023.1 | CERT\_C STR02 | Injection patterns |
| SonarQube C/C++ | 10.x | cpp:S3649 | SQL string construction risks |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Name: Prefer RAII/smart pointers; avoid UAF, double-free, leaks  Rationale: Manual new/delete across paths is fragile and unsafe. |

| **Noncompliant Code** |
| --- |
| Early return leaks memory. |
| int\* p = new int[100];  if (!ok()) return; // leak  delete[] p; |

| **Compliant Code** |
| --- |
| RAII containers/smart pointers. |
| #include <vector>  std::vector<int> buf(100); // auto-frees |

**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 | 18 | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2024.2 | USE\_AFTER\_FREE / RESOURCE\_LEAK | Lifetime issues |
| ASan/UBSan | LLVM 17 | sanitizers | Detect leaks/UB at runtime |
| CodeSonar | 8.1p0 | MEM.LEAK | Leak detection |
| Valgrind | 3.22 | memcheck | Heap errors/leaks |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Name: Use assert for internal invariants, not for user input  Rationale: Asserts can be compiled out; never rely on them for runtime validation. |

| **Noncompliant Code** |
| --- |
| Using assert to validate untrusted data |
| assert(userLen > 0 && userLen < 1024);  process(userLen); // in release, assert may vanish |

| **Compliant Code** |
| --- |
| Check and handle at runtime. |
| if (userLen <= 0 || userLen >= 1024) return error();  process(userLen); |

**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 | High | 1 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 17.0 | misc-static-assert | Misuse of asserts |
| LDRA Tool Suite | 9.7.1 | 44 S | Assertion usage rules |
| ECLAIR | 1.2 | CC2.DCL03 | Assertion compliance |
| Cppcheck | 2.13 | warning/assert | Suspicious asserts |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Name: Throw/catch by type; RAII for cleanup; no throwing destructors  Rationale: Bad exception use causes leaks, crashes, or swallowed errors. |

| **Noncompliant Code** |
| --- |
| Throwing strings; catching by value; destructor throws. |
| ~Worker() { throw "nope"; } // UB territory  try { doWork(); } catch (std::exception e) { /\* slicing \*/ } |

| **Compliant Code** |
| --- |
| Throw std types; catch by const&; RAII manages resources. |
| ~Worker() noexcept = default;  try { doWork(); }  catch (const std::exception& e) { log(e.what()); } |

**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 | Probable | Medium | 4 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-ERR50-a..m | Exception-safety rules |
| CodeSonar | 8.1p0 | BADFUNC.ABORT/EXIT | Unsafe termination/throws |
| Clang-Tidy | 17.0 | performance-noexcept-move | noexcept patterns |
| SonarQube C/C++ | 10.x | cpp:S112 | Exception misuse |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-008-CPP | Name: Parse and validate at the boundary (CLI, file, network)  Rationale: Bad input becomes bad state; validate where it enters. |

| **Noncompliant Code** |
| --- |
| Blind parse then use. |
| int port = std::stoi(argv[1]); // could be negative/huge  startServer(port); |

| **Compliant Code** |
| --- |
| Validate format + range. |
| int port = parseInt(argv[1]);  if (port < 1024 || port > 65535) return error(); // pick policy  startServer(port); |

**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 | Possible | Medium | 10 | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2024.2 | UNVALIDATED\_INPUT | Missing boundary checks |
| Semgrep | 1.64+ | security.input-validation | Boundary validation |
| Parasoft C/C++test | 2023.1 | CERT\_C MSC24 | Validate all inputs |
| Clang Static Analyzer | 17.0 | core.CallAndMessage | Unchecked results |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-009-CPP | Name: Guard shared state; avoid data races  Rationale: Races = undefined behavior + security bugs. |

| **Noncompliant Code** |
| --- |
| Shared vector with no lock. |
| std::vector<int> q;  void push(int v){ q.push\_back(v); } // not thread-safe |

| **Compliant Code** |
| --- |
| std::mutex with RAII locking. |
| std::vector<int> q;  std::mutex m;  void push(int v){ std::lock\_guard<std::mutex> lk(m); q.push\_back(v); } |

**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 | 6 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang ThreadSanitizer | 17.0 | tsan | Data races at runtime |
| Helgrind (Valgrind) | 3.22 | helgrind | Thread synchronization issues |
| CodeSonar | 8.1p0 | CONCURRENCY | Race conditions |
| Polyspace Bug Finder | R2024a | CERT\_CPP-CTR53 | Iterator/race checks |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-010-CPP | Name: Log safely (no secrets, no raw payloads)  Rationale: Logs leak data all the time; attackers read them too. |

| **Noncompliant Code** |
| --- |
| Logs secrets and raw SQL. |
| log("pw=" + password + " sql=" + sql); |

| **Compliant Code** |
| --- |
| Redact/structure logs; use IDs, not secrets. |
| logInfo("user\_login\_attempt", {{"userId", userId}, {"result", "fail"}}); |

**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 | Medium | 2 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.SEC.LOG | Sensitive data in logs |
| Semgrep | 1.64+ | security.log-leak | Secrets in logs |
| SonarQube C/C++ | 10.x | cpp:S4829 | Hard-coded creds/logging |
| Parasoft C/C++test | 2023.1 | MISRA-21.6 | Log content rules |

### 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 | 6 | 2 |
| STD-002-CPP | High | Probable | Medium | 12 | 1 |
| STD-003-CPP | High | Likely | Medium | 18 | 1 |
| STD-004-CPP | High | Likely | Medium | 18 | 1 |
| STD-005-CPP | High | Likely | Medium | 18 | 1 |
| STD-006-CPP | Low | Unlikely | High | 1 | 3 |
| STD-007-CPP | Low | Probable | Medium | 4 | 3 |
| STD-008-CPP | High | Possible | Medium | 10 | 1 |
| STD-009-CPP | High | Probable | High | 6 | 2 |
| STD-010-CPP | Low | Unlikely | Medium | 2 | 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 | Encrypt stored data (databases, object storage, volumes, and backups) with AES‑256 or stronger. Keys must be generated, rotated, and stored in a managed KMS/HSM. Applies to all environments holding sensitive data (prod, staging, developer laptops with local copies). Reason: if disks or snapshots are copied or stolen, data is unreadable without keys. |
| Encryption in flight | Use TLS 1.3 for every network connection—external and internal (APIs, logins, file uploads, service‑to‑service). Disable plaintext protocols. Enforce modern ciphers, certificate pinning where practical, and HSTS for web apps. Reason: prevents sniffing and man‑in‑the‑middle attacks during transit. |
| Encryption in use | Limit exposure of secrets while processed in memory. Prefer secure enclaves or OS keyrings for crypto operations; zeroize buffers after use; avoid swapping secrets to disk. Reason: reduces risk from memory scraping, crash dumps, and side‑channel leaks. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Confirm user and service identity using strong credentials and MFA where possible. Use SSO/OIDC for users and short‑lived tokens for services (no long‑lived static keys). Apply lockouts and anomaly detection. Reason: blocks unauthorized access at the front door. |
| Authorization | Grant the least privilege required via role‑based access control (RBAC). Use fine‑grained, resource‑scoped permissions and time‑bound elevation for admin tasks. Review roles quarterly. Reason: limits blast radius if an account or token is abused. |
| Accounting | Record immutable, centralized logs for logins, permission changes, data changes, and file access. Include who/what/when/where; sync clocks (NTP), protect logs from tampering, and retain per policy. Reason: supports audits, forensics, and 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 | 09/21/2025 | Module Three Milestone – Draft Standards | Kain Mason |  |
| 1.1 | 10/05/2025 | Pre‑final Revisions – Added Automation, Encryption & AAA | Kain Mason |  |
| 1.2 | 10/12/2025 | Final Revision – CS 405 Project One | Kain Mason |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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