**Kenneth Dandrow 10/8/25**

**Prof. Aaron Demory CS-405**

**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_w0d19b8h1i9w)

[Purpose 2](#_m5inlr9acl4l)

[Scope 2](#_icjjk6arhclc)

[Module Three Milestone 2](#_8r7jq8iyenr1)

[Ten Core Security Principles 2](#_bqacakc51y23)

[C/C++ Ten Coding Standards 3](#_wfcwpqw71okp)

[Coding Standard 1 4](#_r29s9imdgjft)

[Coding Standard 2 5](#_98ybxvociq4x)

[Coding Standard 3 6](#_voj4q1ejjmwk)

[Coding Standard 4 7](#_lyixfl65bvve)

[Coding Standard 5 8](#_huwtnoywhc8m)

[Coding Standard 6 9](#_4k84wdllh094)

[Coding Standard 7 10](#_si2k21juz77y)

[Coding Standard 8 11](#_5wu6e62f74pn)

[Coding Standard 9 13](#_rin8jag321hh)

[Coding Standard 10 14](#_44p2g7ubpklu)

[Defense-in-Depth Illustration 15](#_q3oqj7gqdc73)

[Project One 15](#_ubtec64skour)

[1.](#_ogyjeejfxcct) Revise the C/C++ Standards 15

[2.](#_e4bos2mmp1zg) Risk Assessment 15

[3.](#_yslvl5nrtdzq) Automated Detection 15

[4.](#_s7dbtezjc42) Automation 15

[5.](#_qh8e2tz9tynp) Summary of Risk Assessments 16

[6.](#_mh8i6rej7wla) Create Policies for Encryption and Triple A 16

[7.](#_wph6ku8kuz46) Map the Principles 17

[Audit Controls and Management 18](#_cd99xfmbb1we)

[Enforcement 18](#_ujy62tzhv1no)

[Exceptions Process 18](#_1n71i4swcrn5)

[Distribution 19](#_6i2kbkz88sah)

[Policy Change Control 19](#_3niw1gj1l5os)

[Policy Version History 19](#_3pp2n2hgexd5)

[Appendix A Lookups 19](#_nl13199xpcwk)

[Approved C/C++ Language Acronyms 19](#_c6yi226di1tl)

## 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 treat all input as untrusted. Check type, size, range, and format before use. Reject or sanitize anything that does not match the expected pattern. |
| 1. Heed Compiler Warnings | Turn warnings up to the highest level and fix them. Warnings often point to bugs like truncation, sign errors, and uninitialized use. |
| 1. Architect and Design for Security Policies | Plan for security from the start. Pick designs that support access control, encryption, and logging instead of trying to bolt them on later. |
| 1. Keep It Simple | Prefer simple code and simple designs. Complex logic is hard to reason about and easier to break. |
| 1. Default Deny | Deny by default and allow only what is needed. Use allowlists instead of blocklists. |
| 1. Adhere to the Principle of Least Privilege | Give code and users the smallest set of permissions needed to do the job. Drop privileges as soon as possible. |
| 1. Sanitize Data Sent to Other Systems | Before you build SQL, shell commands, or file paths, encode or parameterize values so user input cannot change intent. |
| 1. Practice Defense in Depth | Use layers: input validation, parameterized queries, memory safety, code reviews, SAST/DAST, and runtime protections. |
| 1. Use Effective Quality Assurance Techniques | Write unit tests, integration tests, fuzz tests, and regression tests. Add security tests to CI so they run on every change. |
| 1. Adopt a Secure Coding Standard | Follow a known standard (SEI CERT for C/C++) and enforce it with automated tools and code review. |

### 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 of Standard:** Use explicit, correct-width integer types and avoid implicit narrowing.  **Rationalize the standard:** Implicit conversions and mixed signedness can truncate values or flip comparisons. Use fixed-width types and validate before casting. |

| **Noncompliant Code** |
| --- |
| **Implicit narrowing from a larger unsigned type to a smaller signed type.** |
| uint32\_t size = getSize();  int8\_t small = size; // truncation risk |

| **Compliant Code** |
| --- |
| **Check the range, then cast on success.** |
| uint32\_t size = getSize();  if (size > INT8\_MAX) { throw std::range\_error("size too large"); }  int8\_t small = static\_cast<int8\_t>(size); |

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

| **Principles(s):**  Validate input data — check ranges before casting.  Keep it simple — prefer clear, fixed-width types.  Adopt a secure coding standard — follow SEI CERT and tool rules. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| clang-tidy | 17 | cppcoreguidelines-narrowing-conversions, bugprone-signed-char-misuse | Flags implicit narrowing and risky signed/unsigned use |
| SonarQube (C/C++) | 10.x | Rules related to dangerous casts and type conversions | Highlights truncation and sign problems in PRs and CI |
| Cppcheck | 2.x | --enable=warning (narrowing/sign) | Finds narrowing and sign mix issues |
| Code Review Gate | - | Require explicit fixed-width types in diffs | Manual policy enforcement in reviews |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | **Name of Standard:** Validate numeric ranges and protect against divide by zero.  **Rationalize the standard:** Out-of-range inputs and zero divisors cause crashes and logic errors. Always check bounds and preconditions. |

| **Noncompliant Code** |
| --- |
| **Missing precondition checks.** |
| int avg = sum / count; // count may be 0 |

| **Compliant Code** |
| --- |
| **Enforce valid ranges first.** |
| if (count <= 0) { throw std::invalid\_argument("count must be positive"); }  int avg = sum / count; |

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

| **Principles(s):**  Validate input data — confirm ranges and nonzero divisors.  Default deny — reject invalid values.  Use effective QA — add unit tests for edge cases. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| clang-analyzer | 17 | core.DivideZero | Static detection of divide-by-zero paths |
| SonarQube (C/C++) | 10.x | Rule for possible divide by zero | Reports risky arithmetic |
| Cppcheck | 2.x | divide-by-zero diagnostics | Warns on unchecked divisors |
| Unit Tests in CI | - | gtest cases for zero/negative inputs | Prevents regressions in pipelines |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | **Name of Standard:** Range-check string and array access. Prefer std::string over C strings.  **Rationalize the standard:** Out-of-bounds access can corrupt memory. Use bounds-checked access and safer string types. |

| **Noncompliant Code** |
| --- |
| **Unchecked index on a string.** |
| std::string name = input;  char c = name[100]; // may be out of range |

| **Compliant Code** |
| --- |
| **Check the size and use at() for bounds checking.** |
| std::string name = input;  if (name.size() > 100) { /\* handle \*/ }  char c = name.at(99); // throws on out-of-range |

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

| **Principles(s):**  Validate input data.  Defense in depth.  Adopt a secure coding standard. *(SEI CERT STR53-CPP: Range check element access.)* |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| clang-tidy | 17 | cppcoreguidelines-pro-bounds-constant-array-index, readability-container-size-empty | Finds unsafe indexing and common string misuse |
| SonarQube (C/C++) | 10.x | Rules on bounds and unsafe access | Detects possible out-of-bounds reads |
| Cppcheck | 2.x | --enable=warning (bounds) | Flags risky index expressions |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | **Name of Standard:** Use parameterized queries. Do not build SQL by string concatenation.  **Rationalize the standard:** Concatenated SQL lets attackers change the query. Bind parameters instead. |

| **Noncompliant Code** |
| --- |
| **SQL made from string parts.** |
| std::string sql = "SELECT \* FROM users WHERE name='" + user + "'";  db.exec(sql); |

| **Compliant Code** |
| --- |
| **Use placeholders and bind values.** |
| auto stmt = db.prepare("SELECT \* FROM users WHERE name=?");  stmt.bind(1, user);  stmt.exec(); |

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

| **Principles(s):**  Validate input data.  Default deny.  Sanitize data sent to other systems.  Defense in depth. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (C/C++) | 10.x | SQL injection rules | Flags string-built queries |
| Semgrep | Latest | c/sql-injection ruleset | Pattern match on unsafe SQL building |
| CodeQL (C/C++) | Latest | Data flow to SQL sinks | Tracks tainted input into query APIs |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | **Name of Standard:** Prefer RAII and smart pointers. Avoid raw new and delete.  **Rationalize the standard:** RAII prevents leaks and use-after-free by tying lifetime to scope. |

| **Noncompliant Code** |
| --- |
| **Manual memory management.** |
| Widget\* w = new Widget();  use(w);  delete w; // easy to forget on error paths |

| **Compliant Code** |
| --- |
| **Use ownership types.** |
| auto w = std::make\_unique<Widget>();  use(w.get()); // lifetime managed by unique\_ptr |

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

| **Principles(s):**  Keep it simple.  Defense in depth.  Adopt a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| clang-tidy | 17 | modernize-make-unique, cppcoreguidelines-owning-memory | Recommends smart pointers/RAII and flags owning raw pointers |
| SonarQube (C/C++) | 10.x | memory leak/invalid free rules | Detects leaks, double frees, and missing deletes |
| Cppcheck | 2.x | resourceLeak, memleak | Finds leaks and forgotten frees |
| Code Review Gate | - | ban new/delete in app code | Manual policy: require RAII/smart pointers |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | **Name of Standard:** Do not use assert for input validation or control flow.  **Rationalize the standard:** Asserts may be compiled out in release builds. They do not protect users at runtime. |

| **Noncompliant Code** |
| --- |
| **Assert used to guard a null pointer.** |
| assert(ptr != nullptr);  \*ptr = 3; // unsafe in release builds |

| **Compliant Code** |
| --- |
| **Validate and handle failure.** |
| if (!ptr) { throw std::invalid\_argument("null pointer"); }  \*ptr = 3; |

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

| **Principles(s):**  Validate input data.  Keep it simple.  Use effective QA techniques. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Semgrep | Latest | Pattern: assert(...) on public paths | Flags misuse of asserts |
| SonarQube (C/C++) | 10.x | Rules discouraging debug-only checks | Reports reliance on asserts |
| Code Review Gate | - | Checklist item | Review rejects asserts for validation |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | **Name of Standard:** Throw by value, catch by const reference. Destructors must not throw.  **Rationalize the standard:** Bad exception practices lead to leaks and program termination. |

| **Noncompliant Code** |
| --- |
| **Catch by value and throwing in destructor.** |
| try { work(); }  catch (MyError e) { handle(e); } // copy by value  ~Resource() noexcept(false) { throw; } // destructor throws |

| **Compliant Code** |
| --- |
| **Correct exception usage.** |
| try { work(); }  catch (const MyError& e) { handle(e); }  ~Resource() noexcept { /\* cleanup only \*/ } |

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

| **Principles(s):**  Keep it simple.  Defense in depth.  Use effective QA techniques.  Adopt a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| clang-tidy | 17 | misc-throw-by-value-catch-by-reference, cppcoreguidelines-noexcept-destructor | Enforces correct throw/catch and noexcept destructors |
| SonarQube (C/C++) | 10.x | Exception handling rules | Flags risky throw patterns |
| Cppcheck | 2.x | Exception safety checks | Detects problematic throws |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input and Output Handling | STD-008-CPP | **Name of Standard:** Normalize and validate file paths. Prevent directory traversal.  **Rationalize the standard:** Attackers can escape intended folders using “..” or alternate separators. |

| **Noncompliant Code** |
| --- |
| **Directly joining user input to a base path.** |
| std::ifstream f(base + "/" + userPath); // userPath may be "../secret" |

| **Compliant Code** |
| --- |
| **Canonicalize and check location.** |
| auto path = std::filesystem::weakly\_canonical(base / userPath);  if (path.native().rfind(base.native(), 0) != 0) { throw std::runtime\_error("bad path"); }  std::ifstream f(path); |

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

| **Principles(s):**  Validate input data.  Default deny.  Sanitize data sent to other systems.  Defense in depth. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (C/C++) | 10.x | S2083 Path traversal | Detects unvalidated file paths |
| Semgrep | Latest | path-traversal rules | Flags unsafe path joins |
| CodeQL (C/C++) | Latest | Path injection queries | Traces tainted data into file APIs |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Cryptography | STD-009-CPP | **Name of Standard:** Use approved crypto libraries and strong randomness.  **Rationalize the standard:** Custom crypto and weak RNGs are easy to break. |

| **Noncompliant Code** |
| --- |
| **Weak algorithms and predictable random.** |
| int token = rand(); // predictable  std::string h = sha1(data); // weak hash for new uses |

| **Compliant Code** |
| --- |
| **Use vetted libraries and proper APIs.** |
| // Non-crypto random:  std::random\_device rd; std::mt19937\_64 gen(rd());  // Crypto operations:  auto token = os\_secure\_random\_bytes(32); // platform or vetted lib  // Use AES-GCM, ChaCha20-Poly1305, and SHA-256 or better |

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

| **Principles(s):**  Architect and design for security policies.  Defense in depth.  Adopt a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (C/C++) | 10.x | Weak crypto and RNG rules | Flags SHA-1, MD5, and rand() misuse |
| Semgrep | Latest | crypto and weak-crypto rules | Pattern check for banned algorithms |
| CodeQL (C/C++) | Latest | Weak crypto queries | Finds use of broken or weak algos |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Logging and Secrets | STD-010-CPP | **Name of Standard:** Do not log secrets. Protect logs and include safe context only.  **Rationalize the standard:** Logs often leak credentials and PII. Keep sensitive data out and restrict access to log files. |

| **Noncompliant Code** |
| --- |
| **Logging passwords.** |
| LOG("Login failed for " << user << " with password " << pwd); |

| **Compliant Code** |
| --- |
| **Redact secrets and lock down logs.** |
| LOG\_SECURE("Login failed", {{"user", userId}}); // no secrets  // Set permissions, rotate logs, and ship to central store |

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

| **Principles(s):**  Default deny.  Least privilege.  Defense in depth.  Use effective QA techniques. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (C/C++) | 10.x | S2068 hardcoded credentials and sensitive sinks | Detects secrets and risky logging |
| Semgrep | Latest | logs-leak, generic.secrets | Finds prints of passwords, tokens, keys |
| Gitleaks or TruffleHog | Latest | Secret scanner | Blocks commits that include secrets |
| CI Policy Gate | - | Fail build on secret findings | Enforces rule in the pipeline |

### 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.

**How we enforce this policy with DevSecOps**

* **Create/Plan:** Developers enable IDE plug-ins and pre-commit hooks. clang-tidy, Cppcheck, and Semgrep run locally and block commits on critical issues.
* **Verify (CI):** On every push, the pipeline runs SAST (SonarQube/CodeQL/Semgrep/Cppcheck) and unit + fuzz tests. Security gates must pass or the build fails.
* **Preprod/Release:** We sign artifacts, scan containers and dependencies, and run DAST against the staging app. Only signed, passing builds can release.
* **Operations:** Logs flow to a central system with alerts. RASP/WAF rules add a protective layer. Incidents open tickets automatically for response and tracking.
* **Adapt:** Trend dashboards show recurring issues so we can update rules, training, and secure defaults.

### 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 | Medium | Low | High | 4 |
| STD-002-CPP | High | Medium | Low | High | 4 |
| STD-003-CPP | High | Medium | Medium | High | 4 |
| STD-004-CPP | High | High | Low | High | 5 |
| STD-005-CPP | High | Medium | Medium | High | 4 |
| STD-006-CPP | Medium | Medium | Low | Medium | 3 |
| STD-007-CPP | Medium | Medium | Medium | Medium | 3 |
| STD-008-CPP | High | Medium | Medium | High | 4 |
| STD-009-CPP | High | Medium | Medium | High | 4 |
| STD-010-CPP | High | Medium | Low | High | 4 |

### 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 | Data on disks, databases, and backups is encrypted with approved algorithms (for example, AES-256-GCM). Keys live in a managed KMS. Limit key access by role, rotate keys, and encrypt both full volumes and sensitive fields. Applies to servers, laptops, mobile, cloud storage, and backups. |
| Encryption in flight | All sensitive traffic uses TLS 1.2+ with strong ciphers. Disable insecure protocols. Use mutual TLS for service-to-service calls. Enforce HSTS on web apps and avoid sending secrets over plain HTTP. |
| Encryption in use | Keep secrets in memory only as long as needed, clear buffers after use, and restrict debug dumps. For high-risk data, use HSMs or hardware enclaves where available for crypto operations. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Centralized identity with MFA for users and service identities for apps. Use SSO (OIDC/SAML). Strong password policy, secure session cookies, and session timeouts. Disable default accounts. |
| Authorization | Least privilege by design with RBAC/ABAC. Deny by default and check permissions on every request at the resource layer. Version-control role definitions and review changes. |
| Accounting | Log authentication events, role/permission changes, sensitive data access, admin actions, and config changes. Protect logs with access controls, sync time, make logs tamper-evident, alert on suspicious patterns, and retain per policy. |

**\***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

**STD-001-CPP:** 1, 4, 10 - validate ranges; simple, fixed-width types; follow a standard.

**STD-002-CPP:** 1, 5, 9 - validate values; default deny invalid inputs; test edge cases.

**STD-003-CPP:** 1, 8, 10 - bounds checks; layered protection; SEI CERT guidance.

**STD-004-CPP:** 1, 5, 7, 8 - validate, default deny, sanitize to SQL, layered controls.

**STD-005-CPP:** 4, 8, 10 - simple RAII ownership; defense-in-depth; coding standard.

**STD-006-CPP:** 1, 4, 9 - real runtime checks; simple control flow; unit tests enforce it.

**STD-007-CPP:** 4, 8, 9, 10 - clear exception rules; layered recovery; tested; standard.

**STD-008-CPP:** 1, 5, 7, 8 - validate and normalize paths; deny by default; sanitize; layers.

**STD-009-CPP:** 3, 8, 10 - design for crypto; multiple layers; approved algorithms.

**STD-010-CPP:** 5, 6, 8, 9 - deny leaking secrets; least-privilege logs; layers; tests.

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 |
| --- | --- | --- | --- | --- |
| 1.1 | 2025-10-10 | Converted template to Green Pace policy; added 10 coding standards with examples, risk tables, automation plan, and encryption + Triple-A policies. | Kenneth Dandrow | Security Manager, Green Pace |
| 1.1.1 | 2025-10-11 | Fixed labels and typos; standardized STD-001…STD-010 names; aligned severity wording; added short references. | Kenneth Dandrow | DevSecOps Lead |
| 1.2 | 2025-10-12 | Expanded tooling (clang-tidy, SonarQube, Semgrep, CodeQL); tightened SQL-injection standard; added log-redaction and secret-scanning requirements; clarified “encryption in use.” | Kenneth Dandrow | Security Manager, Green Pace |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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