**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 | Assume all inputs are hostile until proven otherwise. Check type, length, format, encoding, and range before use; reject or sanitize early and log cleanly. |
| 1. Heed Compiler Warnings | Treat warnings as errors. Turn on strict flags and static analysis to surface UB, narrowing, and lifetime bugs before they ship. |
| 1. Architect and Design for Security Policies | Bake authN/Z, logging, and encryption into design so the secure path is the default, not an afterthought. |
| 1. Keep It Simple | Simple APIs and clear ownership make mistakes less likely and code easier to review and test. |
| 1. Default Deny | Close everything by default; permissions, inputs, network egress, and only open what’s explicitly required. |
| 1. Adhere to the Principle of Least Privilege | Minimize capabilities for code and users. Use process isolation, sandboxing, and least-capability interfaces. |
| 1. Sanitize Data Sent to Other Systems | Clean what you output, not just what you take in. When crossing a boundary (SQL, shell, HTML, logs), encode/parameterize so the receiver can’t interpret data as commands. |
| 1. Practice Defense in Depth | Layer controls so if one fails, others still protect you. Combine validation, least privilege, sandboxing, rate limits, and monitoring rather than betting on a single control. |
| 1. Use Effective Quality Assurance Techniques | Mix unit tests, fuzzing, property tests, and code review. Automate what you can in CI so security regressions are caught before release. |
| 1. Adopt a Secure Coding Standard | Follow a recognized standard and make it enforceable via tooling and review checklists so “secure” is the norm, not the exception. |

### 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 Correctness** | [STD-001-CPP] | Rule: Use fixed-width types (std::int32\_t, std::uint64\_t) for externally visible or serialized data.  Rationale: Ensures consistent size across platforms, preventing data truncation and ABI mismatch; mitigates numeric overflow and portability issues (SEI CERT, 2025). |

| **Noncompliant Code** |
| --- |
| Uses int for wire/on-disk ID; size varies by platform. |
| struct Record { int id; };  void write(std::ostream& os, const Record& r) {  os.write(reinterpret\_cast<const char\*>(&r.id), sizeof(r.id));  } |

| **Compliant Code** |
| --- |
| Fixed-width type; explicit network order makes layout stable. |
| #include <cstdint>  struct Record { std::int32\_t id; };  void write(std::ostream& os, const Record& r) {  std::int32\_t net = htonl(r.id);  os.write(reinterpret\_cast<const char\*>(&net), sizeof(net));  } |

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

| **Principles(s):** Principle(s): 1 (Validate Input Data), 2 (Heed Compiler Warnings), 9 (Use Effective QA Techniques) |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (CFamily / C++ analyzer) | 10.x (server) + CFamily 7.x | Integer/narrowing conversions; platform-dependent integer use (Rules Explorer – C/C++) | Static analysis in CI. Flags implicit narrowing (int→int16\_t), signed/unsigned conversions, and use of platform-dependent types in serialized/ABI-relevant structs. Fails PR via Quality Gate. |
| clang-tidy | 18.x | bugprone-narrowing-conversions, cppcoreguidelines-narrowing-conversions, google-runtime-int | Warns on value loss during integer conversions and discourages long/unsigned long portability hazards; suggests fixed-width types (std::int32\_t, std::uint64\_t). |
| Cppcheck | 2.14+ | Category: portability, type, warning (narrowing, sign/size mismatches) | Detects overflow-prone arithmetic, size/sign mismatches (size\_t vs fixed-width), and nonportable integer usage in data models and file/network I/O. |
| Polyspace Bug Finder | R2025b | CERT C++ “INT” family, integer conversion loss of data | Finds risky integer conversions affecting serialized layouts; integrates with reports for audit evidence. |

Reference:

SEI CERT C Coding Standard. INT36-C. Converting a pointer to integer or integer to pointer.

Retrieved October 2025 from https://wiki.sei.cmu.edu/confluence/display/c/INT36-C.+Converting+a+pointer+to+integer+or+integer+to+pointer

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value Validation** | [STD-002-CPP] | Rule: Validate all numeric operations for overflow and range prior to conversion.  Rationale: Prevents wraparound and numeric overflow that can trigger buffer overruns or denial-of-service (SEI CERT, 2025). |

| **Noncompliant Code** |
| --- |
| Length addition can wrap size\_t. |
| size\_t need = header\_len + payload\_len;  std::vector<char> buf(need); |

| **Compliant Code** |
| --- |
| Detects overflow and caps max size. |
| bool add\_overflows(size\_t a, size\_t b){ return a > SIZE\_MAX - b; }  size\_t safe\_len(size\_t h,size\_t p){  if(add\_overflows(h,p) || h+p > (1u<<24)) throw std::length\_error("bad");  return h+p;  }  std::vector<char> buf(safe\_len(header\_len,payload\_len)); |

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

| **Principles(s):** 1 (Validate Input Data), 4 (Keep It Simple), 8 (Defense in Depth) |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (CFamily / C++ analyzer) | 10.x (server) + CFamily 7.x | Integer overflow, narrowing conversions (Rules Explorer, C/C++) | Static analysis in CI; flags overflow-prone math (a+b on size\_t, shifts, signed & unsigned mixing) and implicit narrowing before allocation or I/O. Blocks merge via Quality Gate. |
| clang-tidy | 18.x | cppcoreguidelines-narrowing-conversions, bugprone-narrowing-conversions, clang-analyzer-core.\* (undefined result), clang-analyzer-security.insecureAPI.\* (when conversions feed APIs) | Warns when arithmetic may lose value or overflow and when conversions occur prior to bounds checks; integrates with compile commands and PR comments. |
| Cppcheck | 2.14+ | Categories warning, portability, performance (integer overflow, negative/too-large shift, sign/size mismatch) | Detects risky arithmetic and size/length calculations used for buffer sizes; CLI-friendly for pipelines and artifact reports. |
| Polyspace Bug Finder | R2025b | CERT C/C++ “INT” family — integer conversion/overflow | Proves/flags overflow paths and tainted length calculations; exports audit-ready findings. |

Reference:

SEI CERT. (2025). INT32-C: Ensure that operations on signed integers do not result in overflow.

Software Engineering Institute, Carnegie Mellon University.

Archived version retrieved October 2025 from

https://web.archive.org/web/20230419082719/https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Rule: Use std::string or std::string\_view instead of raw C-strings; bounds-check all indexing operations (prefer .at()).  Rationale: Prevents buffer overflow and undefined behavior from unchecked access (SEI CERT, 2025). |

| **Noncompliant Code** |
| --- |
| Unbounded copy into fixed buffer. |
| char name[16];  strcpy(name, user\_input.c\_str()); |

| **Compliant Code** |
| --- |
| Use dynamic string or bounded copy. |
| std::string name = user\_input;  if(name.size()>15) throw std::runtime\_error("name is too long");  // or fixed buffer with truncation  std::array<char,16> buf{};  std::snprintf(buf.data(), buf.size(), "%.\*s", 15, name.c\_str()); |

**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), 8 (Defense in Depth) |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (CFamily) | 10.x (server) + CFamily 7.x | Rule S2228 - Out-of-bounds access Rule S3514 - Untrusted data used in array access | Scans C++ and C code for string or array index violations and unsafe functions (strcpy, sprintf). Integrates into CI/CD quality gates. |
| clang-tidy | 18.x | bugprone-string-literal-unchecked-length, cppcoreguidelines-pro-bounds-array-to-pointer-decay | Analyzes string/array bounds issues and warns when .at() should be used over [ ]. Highlights potential out-of-range std::string access. |
| Cppcheck | 2.14x | Category “warning/security” – buffer overrun (unsafe C string API use) | Detects use of functions like strcpy, gets, and flagged string manipulations likely to overflow. CLI-friendly for pipeline automation. |
| Polyspace Bug Finder | R2025b | CERT C++ STR53-CPP rule set | Formally verifies that std::string access is within bounds; detects tainted index values used in string operations and reports violations for audits. |

Reference:

SEI CERT C++ Coding Standard. STR53-CPP. Range check element access.

Archived version retrieved October 2025 from

https://web.archive.org/web/20230401034733/https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Rule: Use parameterized queries or prepared statements; never concatenate user input into SQL commands.  Rationale: Prevents SQL injection attacks by separating executable code from untrusted data. This approach follows secure coding guidance from the Open Web Application Security Project (OWASP, 2025) and standard secure-coding practices for C and C++ database interfaces such as SQLite and MySQL APIs. |

| **Noncompliant Code** |
| --- |
| Concatenation with raw input. |
| std::string sql = "SELECT \* FROM users WHERE name='" + name + "'";  execute(db, sql); |

| **Compliant Code** |
| --- |
| Bind parameter via driver API. |
| sqlite3\_stmt\* stmt{};  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):** 1 (Validate Input Data), 7 (Sanitize Data Sent to Other Systems), 9 (Use Effective Quality Assurance Techniques) |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (CFamily) | 10.x (server) + CFamily 7.x | Rule S3649 - SQL Injection | Identifies dynamic SQL string concatenation and enforces binding via query parameters or ORM functions; integrated into pull request gates. |
| clang-tidy | 18.x | cert-err58-cpp (Exception safety / input handling) | Warns when user input is used directly in expressions or passed unsanitized to APIs; can be combined with custom checkers for unsafe SQL string building. |
| Cppcheck | 2.14+ | Category “security” | Detects concatenation of string literals with variables likely to form SQL statements; static pattern search in code bases without runtime. |
| Semgrep | 1.67+ | Rule ID: cpp.sql-injection.concat | Searches for C++ functions building SQL queries via string concatenation and reports vulnerabilities inline during CI analysis. |

References:

OWASP. (2025). SQL Injection Prevention Cheat Sheet.

Open Worldwide Application Security Project.

Retrieved October 2025 from https://cheatsheetseries.owasp.org/cheatsheets/SQL\_Injection\_Prevention\_Cheat\_Sheet.html

SQLite. (2025). C/C++ Interface for SQLite: Bind interfaces.

Retrieved October 2025 from https://www.sqlite.org/c3ref/bind\_blob.html

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Rule: Prefer RAII and C++ smart pointers (std::unique\_ptr, std::shared\_ptr) and resource-wrapper types over raw new/delete and manual close/free.  Rationale: RAII guarantees timely release even on early return or exception, preventing leaks, double-free, and use-after-free; it also simplifies ownership and error paths (C++ Core Guidelines; SEI CERT, archived). |

| **Noncompliant Code** |
| --- |
| Leak on early return. |
| Widget\* w = new Widget();  if (!w->init()) return; // leak  use(w);  delete w; |

| **Compliant Code** |
| --- |
| RAII with std::unique\_ptr. |
| auto w = std::make\_unique<Widget>();  if (!w->init()) return;  use(\*w); |

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

| **Principles(s):** 5 (Default Deny), 6 (Least Privilege), 8 (Defense in Depth) |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (CFamily) | 10.x + CFamily 7.x | Memory/resource leak; “not closed” descriptors; suspicious new/delete pairs | Finds leaks on exceptional/early-return paths; enforces RAII usage. |
| clang-tidy | 18.x | modernize-make-unique, modernize-make-shared, cppcoreguidelines-owning-memory, bugprone-use-after-move | Nudges to factory helpers, discourages manual new, and catches ownership bugs. |
| Cppcheck | 2.14+ | memleak, resourceLeak, uselessexpr (on deleted paths) | Detects leaks and missing close/free; easy CLI for CI artifacts. |
| Valgrind (Memcheck) | 3.22+ | Leak summary / definitely lost | Runtime verification that all allocations are freed; attach logs to build artifacts. |

Reference:

SEI CERT. (2025). MEM51-CPP: Properly deallocate dynamically allocated resources.

Software Engineering Institute, Carnegie Mellon University.

Archived version retrieved October 2025 from

https://web.archive.org/web/20230401032535/https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM51-CPP.+Properly+deallocate+dynamically+allocated+resources

Stroustrup, B., Sutter, H., & ISO C++ WG. (2025). C++ Core Guidelines: R — Resource management.

Retrieved October 2025 from https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rr-resource

cppreference. (2025). std::unique\_ptr.

Retrieved October 2025 from https://en.cppreference.com/w/cpp/memory/unique\_ptr

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-LLL] | Rule: Do not rely on assert for runtime validation or side effects. Assertions may be compiled out, so never use them to guard untrusted input; perform real checks and fail safely.  Rationale: Using assert for runtime validation can remove safeguards in release builds, leading to unsafe behavior or program termination. Instead, use explicit error handling or contract-based checks as recommended by the C++ Core Guidelines and Google C++ Style Guide (Stroustrup et al., 2025; Google, 2025). |

| **Noncompliant Code** |
| --- |
| Input length only guarded by assert. |
| assert(len < MAX\_LEN);  process(buf, len); |

| **Compliant Code** |
| --- |
| Explicit runtime check with error handling. |
| if (len >= MAX\_LEN) throw std::invalid\_argument("too long");  process(buf, len); |

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

| **Principles(s):** 1 (Validate Input Data), 8 (Defense in Depth), 9 (Use Effective QA Techniques) |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (CFamily) | Server 10.x + CFamily 7.x | Assertions used for argument validation / production code (CFamily assert misuse rules) | Marks assert used to validate external inputs or enforce logic in release code; enforce via Quality Gate. |
| clang-tidy | 18.x | bugprone-assert-side-effect, clang-analyzer-core (reachable abort) | Finds side effects inside assert and paths that rely on assert to stop execution. |
| Cppcheck | 2.14+ | assertWithSideEffect | Detects assert that changes program state or is used for runtime checks. |
| GSL Contracts | GSL 4.x | Expects/Ensures usage checks (lint/static checks in CI) | Encourage contract-style pre/postconditions for dev builds while keeping explicit runtime checks in production. |

References:

Stroustrup, B., Sutter, H., & ISO C++ WG. (2025). C++ Core Guidelines: Assertions and contracts (Expects/Ensures).

Retrieved October 2025 from https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#i-assertions-and-contracts

Google. (2025). Google C++ Style Guide: Exceptions and assertions.

Retrieved October 2025 from https://google.github.io/styleguide/cppguide.html#Exceptions\_and\_Error\_Handling

LLVM Project. (2025). LLVM Programmer’s Manual: Assertions.

Retrieved October 2025 from https://llvm.org/docs/ProgrammersManual.html#assertions

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Rule: Throw by value; catch by const&. Use RAII so resources are released when exceptions unwind.  Rationale: Catching by value causes slicing and extra copies; catching by const& preserves dynamic type and avoids cost. RAII ensures deterministic cleanup on all exception paths, preventing leaks and inconsistent state (C++ Core Guidelines; cppreference). |

| **Noncompliant Code** |
| --- |
| Throws pointer; catch by value. |
| throw new std::runtime\_error("oops");  try { /\* ... \*/ } catch(std::exception e) { /\* sliced \*/ } |

| **Compliant Code** |
| --- |
| Throw by value; catch by reference. |
| throw std::runtime\_error("oops");  try { /\* ... \*/ } catch (const std::exception& e) { /\* handle \*/ } |

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

| **Principles(s):** 3 (Architect and Design for Security Policies), 5 (Default Deny), 8 (Defense in Depth) |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| clang-tidy | 18.x | misc-throw-by-value-catch-by-reference, cert-err61-cpp, clang-analyzer-cplusplus.NewDeleteLeaks | Ensures correct throw/catch forms, CERT-style exception usage, and finds leaks that occur on exceptional exits. |
| SonarQube (CFamily) | Server 10.x + CFamily 7.x | Exception-safety and RAII misuse rules (CFamily) | Detects catch-by-value, rethrow misuse, and missing cleanup on exception paths; blocks PR via Quality Gate. |
| Cppcheck | 2.14+ | Category: warning/performance (catch by value, throw in destructor/noexcept), resource-leak on exceptional path | Static checks for exception safety anti-patterns and resource leaks; CLI-friendly reports. |
| Polyspace Bug Finder | R2025b | Exception safety / CERT C++ set | Proves/flags paths where exceptions bypass cleanup; exports audit-ready findings. |

References:

Stroustrup, B., Sutter, H., & ISO C++ WG. (2025). C++ Core Guidelines: E.15 (throw by value, catch by const reference) and R - Resource management (RAII).

Retrieved October 2025 from https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines

cppreference. (2025). Exception handling (C++).

Retrieved October 2025 from https://en.cppreference.com/w/cpp/language/exceptions

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Initialization Safety and Predictability | [STD-008-CPP] | Rule: Initialize all variables and class members before use. Avoid indeterminate values by using brace or member initializers, constructors, or default member initializers.  Rationale:Uninitialized variables can cause undefined behavior, data leakage, or inconsistent program state. Ensuring that all variables are initialized establishes predictable behavior, improves maintainability, and prevents attackers from exploiting leftover memory content. This practice aligns with SEI CERT EXP53-CPP, CWE-457, and the C++ Core Guidelines’ recommendation to always initialize objects (Carnegie Mellon University Software Engineering Institute, 2025; MITRE, 2025; Stroustrup, Sutter, & ISO C++ Working Group, 2025). |

| **Noncompliant Code** |
| --- |
| Accumulator used uninitialized. |
| int sum;  for (int x : vec) sum += x; |

| **Compliant Code** |
| --- |
| Initialize to known value / use algorithms. |
| int sum = 0;  for (int x : vec) sum += x;  // or: int sum = std::accumulate(vec.begin(), vec.end(), 0); |

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

| **Principles(s):** 1 (Validate Input Data), 8 (Defense in Depth), 9 (Use Effective Quality Assurance Techniques) |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (CFamily) | 10.x + CFamily 7.x | Rule S3981 - Uninitialized variables should not be used | Detects variables and fields used before initialization; integrates into CI/CD Quality Gates for enforcement. |
| clang-tidy | 18.x | clang-analyzer-core.uninitialized.UndefReturn, clang-analyzer-core.CallAndMessage | Identifies reads of uninitialized variables or fields across code paths; runs automatically during static analysis. |
| Cppcheck | 2.14+ | uninitvar, uninitStructMember | Reports local and class member variables that may be used uninitialized; supports XML output for audit pipelines. |
| Polyspace Bug Finder | R2025b | CERT C++ MSC30-C / MSC32-C | Performs formal verification to prove initialization before use and exports audit-ready compliance reports. |

References:

Carnegie Mellon University, Software Engineering Institute. (2025). EXP53-CPP. Do not read uninitialized memory. CERT C++ Secure Coding Standard. Retrieved October 2025, from https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory

MITRE. (2025). CWE-457: Use of uninitialized variable. Common Weakness Enumeration. Retrieved October 2025, from https://cwe.mitre.org/data/definitions/457.html

Stroustrup, B., Sutter, H., & ISO C++ Working Group. (2025). C++ Core Guidelines: ES.20 – Always initialize an object. Retrieved October 2025, from https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#es20-always-initialize-an-object

cppreference. (2025). Variable initialization (C++). Retrieved October 2025, from https://en.cppreference.com/w/cpp/language/initialization

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| RAII for Non-memory Resources | [STD-009-CPP] | Rule: Use RAII (Resource Acquisition Is Initialization) wrappers for non-memory resources such as files, sockets, and mutexes instead of manual open/close or lock/unlock calls.  Rationale: Manual resource handling often leads to leaks, deadlocks, or incomplete cleanup when exceptions or early returns occur. RAII automatically releases resources at scope exit, ensuring safe, predictable behavior across all execution paths. This practice aligns with SEI CERT FIO51-CPP and CWE-404 recommendations for proper resource shutdown, as well as the C++ Core Guidelines for deterministic resource management (Carnegie Mellon University Software Engineering Institute, 2025; MITRE, 2025; Stroustrup, Sutter, & ISO C++ Working Group, 2025). |

| **Noncompliant Code** |
| --- |
| Possible missed unlock on throw. |
| mtx.lock();  do\_work(); // may throw  mtx.unlock(); |

| **Compliant Code** |
| --- |
| Scope guard unlocks automatically. |
| std::lock\_guard<std::mutex> lk(mtx);  do\_work(); |

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

| **Principles(s):** 5 (Default Deny), 6 (Least Privilege), 8 (Defense in Depth) |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (CFamily) | 10.x + CFamily 7.x | Rule S3584 - Resources should be properly released | Detects unclosed file handles, sockets, and synchronization primitives; integrated CI Quality Gate enforcement. |
| clang-tidy | 18.x | clang-analyzer-cplusplus.NewDeleteLeaks, cert-err33-c | Analyzes resource lifetimes and warns if RAII wrappers aren’t used for critical objects. |
| Cppcheck | 2.14+ | resourceLeak, unclosedFile | Finds file, socket, and mutex leaks caused by missing release calls; exports reports for DevSecOps dashboards. |
| Polyspace Bug Finder | R2025b | CERT C++ FIO42-CPP - Close files when they are no longer needed | Formally verifies file and lock closure on all paths and produces compliance artifacts. |

References:

Carnegie Mellon University, Software Engineering Institute. (2025). FIO51-CPP: Close files when they are no longer needed. CERT C++ Secure Coding Standard. Retrieved October 2025 from https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed

MITRE. (2025). CWE-404: Improper resource shutdown or release. Common Weakness Enumeration. Retrieved October 2025 from https://cwe.mitre.org/data/definitions/404.html

Stroustrup, B., Sutter, H., & ISO C++ Working Group. (2025). C++ Core Guidelines: R — Resource management. Retrieved October 2025 from https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#Rr-resource

cppreference. (2025). std::lock\_guard. Retrieved October 2025 from https://en.cppreference.com/w/cpp/thread/lock\_guard

#### 

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Bound-Safe Interfaces Using std::span | [STD-010-CPP] | Rule: Prefer std::span (or iterator + size patterns) over raw pointer + length arguments. Binding the data and its bounds prevents mismatched size errors, improves safety, and makes intent explicit.  Rationale: Raw pointers do not inherently track array sizes, which can result in buffer overruns, undefined behavior, or memory corruption. Using std::span enforces compile-time and runtime bounds checking while maintaining lightweight performance. This approach aligns with SEI CERT ARR30-C for bounds checking, CWE-119 (Improper Restriction of Operations within the Bounds of a Memory Buffer), and the C++ Core Guidelines on safe array handling (Carnegie Mellon University Software Engineering Institute, 2025; MITRE, 2025; Stroustrup, Sutter, & ISO C++ Working Group, 2025). |

| **Noncompliant Code** |
| --- |
| Pointer+len mismatch overruns array. |
| void fill(int\* out, size\_t n);  int a[10];  fill(a, 20); // oops |

| **Compliant Code** |
| --- |
| Accept std::span<int>. |
| #include <span>  void fill(std::span<int> out){ for(int& v: out) v = 42; }  int a[10];  fill(a); // size known = 10 |

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

| **Principles(s):** 1 (Validate Input Data), 4 (Keep It Simple), 8 (Defense in Depth) |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube (CFamily) | 10.x + CFamily 7.x | Rule S3514 - Untrusted data used in array access | Flags array/pointer indexing that uses untrusted or tainted values and potential out-of-bounds access. Helps enforce bound-safe interfaces (e.g., std::span) and blocks merges via Quality Gate when violations are detected. |
| clang-tidy | 18.x | cppcoreguidelines-pro-bounds-array-to-pointer-decay, cert-arr30-c | Warns when pointer + length are used unsafely; promotes std::span or range-based loops. |
| Cppcheck | 2.14+ | arrayIndexOutOfBounds, bufferAccessOutOfBounds | Reports possible out-of-bounds access when passing raw arrays with separate length parameters. |
| Polyspace Bug Finder | R2025b | CERT ARR30-C – Do not form or use out-of-bounds pointers or array subscripts | Performs formal verification of array bounds and span-safe access; generates audit-ready results. |

References:

Carnegie Mellon University, Software Engineering Institute. (2025). ARR30-C: Do not form or use out-of-bounds pointers or array subscripts. CERT C Secure Coding Standard. Retrieved October 2025 from https://wiki.sei.cmu.edu/confluence/display/c/ARR30-C.+Do+not+form+or+use+out-of-bounds+pointers+or+array+subscripts

MITRE. (2025). CWE-119: Improper restriction of operations within the bounds of a memory buffer. Common Weakness Enumeration. Retrieved October 2025 from https://cwe.mitre.org/data/definitions/119.html

Stroustrup, B., Sutter, H., & ISO C++ Working Group. (2025). C++ Core Guidelines: ES.42 – Keep use of arrays simple and safe. Retrieved October 2025 from https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines#es42-keep-use-of-arrays-simple-and-safe

cppreference. (2025). std::span (C++ reference). Retrieved October 2025 from https://en.cppreference.com/w/cpp/container/span

### 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

Automation will enforce and verify compliance with all Green Pace secure-coding standards by integrating static analysis, testing, and deployment validation directly into the DevSecOps pipeline. In pre-production, every commit triggers CI jobs that run SonarQube, clang-tidy, and Cppcheck; merges are blocked until the quality gates pass (SonarSource, 2025; LLVM Project, 2025; Cppcheck Team, 2024).

In production, continuous monitoring and configuration checks confirm that deployed artifacts remain compliant. Dependency and vulnerability scans plus audit-ready reports provide feedback to engineers and security analysts and are archived in the Risk Management Dashboard (OWASP Foundation, 2025; NIST, 2022)

Following the DevSecOps loop in the diagram:

Design -> Build -> Verify/Test: Static/dynamic analysis, unit tests, and dependency scanning run automatically; non-compliant builds fail the pipeline (SonarSource; LLVM Project; Cppcheck Team).

Assess/Plan -> Monitor/Detect: Live systems are continuously checked for drift, config non-compliance, and emerging vulns (OWASP DSOMM; NIST SP 800-204C).

Respond -> Maintain/Stabilize: Findings feed the backlog for remediation, closing the loop and improving posture over time (NIST SP 800-204C).

References:

Cppcheck Team. (2024). Cppcheck manual (v2.18). Retrieved October 2025 from https://cppcheck.sourceforge.io/manual.pdf

LLVM Project. (2025). clang-tidy 18 documentation. Retrieved October 2025 from https://releases.llvm.org/18.1.1/tools/clang/tools/extra/docs/clang-tidy/index.html

OWASP Foundation. (2025). DevSecOps Maturity Model (DSOMM). Retrieved October 2025 from https://owasp.org/www-project-devsecops-maturity-model/

SonarSource. (2025). C static code analysis rules. Retrieved October 2025 from https://rules.sonarsource.com/c/

National Institute of Standards and Technology (NIST). (2022). SP 800-204C: Implementation of DevSecOps for cloud-native applications. Retrieved October 2025 from https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-204C.pdf

### 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 Data Type Correctness | High | Low | Medium | High | 2 |
| STD-002-CPP Data Value Validation | High | Medium | Low | High | 3 |
| STD-003-CPP String Correctness | High | Low | Medium | High | 3 |
| STD-004-CPP SQL Injection | High | Medium | Low | High | 3 |
| STD-005-CPP Memory Protection | High | Medium | Medium | High | 3 |
| STD-006-CPP Assertions | Medium | Medium | Low | Medium | 2 |
| STD-007-CPP Exceptions | Medium | Low | Medium | High | 2 |
| STD-008-CPP Initialization Safety and Predictability | High | Medium | Low | High | 3 |
| STD-009-CPP RAII for Non-Memory Resources | High | Medium | Medium | High | 3 |
| STD-010-CPP Bound-Safe Interfaces Using std::span | High | Medium | Low | High | 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 | Encryption at rest protects stored data from unauthorized access or theft. All Green Pace systems must use AES-256 or stronger encryption for disks, databases, and file storage. Keys are rotated and stored using a centralized Key Management Service (KMS). Even if a storage device or backup is compromised, encrypted data remains unreadable without its key (NIST, 2020; Microsoft, 2025; OWASP Foundation, 2025a). |
| Encryption in flight | Encryption in flight safeguards data moving between users, applications, and services. All network communications—including APIs, web traffic, and email—must use TLS 1.3 or higher to prevent interception or modification. Certificates must be validated and automatically renewed through the CI/CD pipeline. This policy aligns with the OWASP TLS guidelines and mitigates man-in-the-middle attacks (IETF, 2018; OWASP Foundation, 2025b). |
| Encryption in use | Encryption in use secures data actively processed in memory or in cloud workloads. Green Pace employs secure enclaves such as Intel SGX or AMD SEV and OS-level memory protection to isolate sensitive data during computation. This reduces exposure to insider or hypervisor attacks (NIST, 2023; Cloud Security Alliance, 2025). |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication verifies user or system identity before granting access. Green Pace enforces multifactor authentication (MFA) and stores all credentials as salted, hashed values using adaptive algorithms such as Argon2 or bcrypt, following NIST SP 800-63B guidelines (NIST, 2023). |
| Authorization | Authorization defines and limits what authenticated users can do. Green Pace applies least-privilege and role-based access control (RBAC) to all systems, reviewing roles quarterly to prevent privilege creep. These controls align with Zero Trust principles (CISA, 2025; OWASP Foundation, 2025c). |
| Accounting | Accounting ensures that every access, configuration change, and data-handling action is logged and auditable. Logs must be immutable, time-synchronized, and centrally stored for review by the Security Operations Center (SOC). This supports rapid incident response and regulatory compliance (NIST, 2023; Cloud Security Alliance, 2025). |

References:

Cloud Security Alliance. (2025). Confidential Computing and Data Protection Best Practices. Retrieved October 2025 from https://cloudsecurityalliance.org/

CISA – Cybersecurity and Infrastructure Security Agency. (2025). Zero Trust Maturity Model Version 2.0. Retrieved October 2025 from https://www.cisa.gov/resources-tools/resources/zero-trust-maturity-model

IETF – Internet Engineering Task Force. (2018). RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3. Retrieved October 2025 from https://datatracker.ietf.org/doc/html/rfc8446

Microsoft. (2025). Azure Storage encryption for data at rest. Retrieved October 2025 from https://learn.microsoft.com/en-us/azure/security/fundamentals/encryption-atrest

NIST – National Institute of Standards and Technology. (2020). FIPS 197: Advanced Encryption Standard (AES). Retrieved October 2025 from https://csrc.nist.gov/publications/detail/fips/197/final

NIST – National Institute of Standards and Technology. (2023). Special Publication 800-63B: Digital Identity Guidelines – Authentication and Lifecycle Management. Retrieved October 2025 from https://pages.nist.gov/800-63-3/sp800-63b.html

OWASP Foundation. (2025a). Cryptographic Storage Cheat Sheet. Retrieved October 2025 from https://cheatsheetseries.owasp.org/cheatsheets/Cryptographic\_Storage\_Cheat\_Sheet.html

OWASP Foundation. (2025b). Transport Layer Security (TLS) Cheat Sheet. Retrieved October 2025 from https://cheatsheetseries.owasp.org/cheatsheets/Transport\_Layer\_Security\_Cheat\_Sheet.html

OWASP Foundation. (2025c). Key Management Cheat Sheet. Retrieved October 2025 from https://cheatsheetseries.owasp.org/cheatsheets/Key\_Management\_Cheat\_Sheet.html

**\***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 |  |
| 1.1 | 09/20/2025 | Completed Security Definitions, Coding Standards tables. | Dominoe LaMattina |  |
| 1.2 | 10/11/2025 | Completed coding standards, summary of risk, and Automation | Dominoe LaMattina |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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