**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 | All untrusted input must be sanitized and encoded to prevent injection attacks. Input used in SQL statements must be parameterized. |
| 1. Heed Compiler Warnings | Compiler warnings are there for a reason. In all continuous integration/ continuous deployment pipelines warnings should be treated as errors and cause builds to fail. Once all compiler warnings have been fixed, the build can proceed. Requests to ignore compiler warnings will be considered on a case-by-case basis and the ignore rules will be set on the line of code causing the error rather than turned off in the file or solution. |
| 1. Architect and Design for Security Policies | The system must be designed with security policies in mind from the start, and must have continual reviews to ensure the security standards are maintained. |
| 1. Keep It Simple | Complex code is more difficult to understand, and when code is not well understood by all developers, bugs may be unintentionally written and expose the system to exploits. |
| 1. Default Deny | Unless permission to perform an action is explicitly granted by the authorization, the action must be forbidden. |
| 1. Adhere to the Principle of Least Privilege | Users and systems should only be given the minimal amount of access required to complete the scope of their work. Any higher access than that is a security risk and should be eliminated. |
| 1. Sanitize Data Sent to Other Systems | When sending data outside of the Green Pace controlled environment all data must be sufficiently sanitized to only contain information the systems have clearance to process. Customer personally identifiable information must be redacted or omitted. |
| 1. Practice Defense in Depth | There is no perfect system, and new exploits are discovered all the time. To have an adequate defense, measures to prevent unauthorized access must my layered for other layers of the defense to pick up in cases where an unknown flaw exposes one layer. |
| 1. Use Effective Quality Assurance Techniques | The QA team must follow industry best practices to ensure security and functionality of all systems. |
| 1. Adopt a Secure Coding Standard | Keeping all developers on the same page with regards to the coding standard is necessary when working as a team. The team is all pulling together to achieve the same goal, and while we believe that every member of the team makes every decision to be what they believe is the best thing for the team with the information they had at the time, we must all check each other’s work to make sure that nothing slips through.  To such end, all code will receive two code reviews before being merged. |

### 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** | **Ensure that integer conversions do not result in lost or misinterpreted data** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Integer conversions, both implicit and explicit (using a cast), must be guaranteed not to result in lost or misinterpreted data. This rule is particularly true for integer values that originate from untrusted sources and are used in any of the following ways:  - Integer operands of any pointer arithmetic, including array indexing  - The assignment expression for the declaration of a variable length array  - The postfix expression preceding square brackets [] or the expression in square brackets [] of a subscripted designation of an element of an array object  - Function arguments of type size\_t or rsize\_t (for example, an argument to a memory allocation function)  <https://wiki.sei.cmu.edu/confluence/display/c/INT31-C.+Ensure+that+integer+conversions+do+not+result+in+lost+or+misinterpreted+data> |

| **Noncompliant Code** |
| --- |
| Unsigned to Signed  Type range errors, including loss of data (truncation) and loss of sign (sign errors), can occur when converting from a value of an unsigned integer type to a value of a signed integer type. This noncompliant code example results in a truncation error on most implementations: |
| #include <limits.h>    void func(void)  {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Unsigned to Signed  Validate ranges when converting from an unsigned type to a signed type. This compliant solution can be used to convert a value of unsigned long int type to a value of signed char type: |
| #include <limits.h>    void func(void)  {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  if (u\_a <= SCHAR\_MAX)  {  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  }  else  {  /\* Handle error \*/  }  } |

| **Principles(s):** Validate Input Data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Sonar Lint | 6.10.0 | Sonar Lint | [SonarLint](https://www.sonarsource.com/products/sonarlint/) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that operations on signed integers do not result in overflow** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Signed integer overflow is undefined behavior 36. Consequently, implementations have considerable latitude in how they deal with signed integer overflow. (See MSC15-C. Do not depend on undefined behavior.) An implementation that defines signed integer types as being modulo, for example, need not detect integer overflow. Implementations may also trap on signed arithmetic overflows, or simply assume that overflows will never happen and generate object code accordingly. It is also possible for the same conforming implementation to emit code that exhibits different behavior in different contexts. For example, an implementation may determine that a signed integer loop control variable declared in a local scope cannot overflow and may emit efficient code on the basis of that determination, while the same implementation may determine that a global variable used in a similar context will wrap.  <https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow> |

| **Noncompliant Code** |
| --- |
| Addition  This noncompliant code example can result in a signed integer overflow during the addition of the signed operands si\_a and si\_b |
| void func(signed int si\_a, signed int si\_b)  {  signed int sum = si\_a + si\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution ensures that the addition operation cannot overflow, regardless of representation. |
| #include <limits.h>    void f(signed int si\_a, signed int si\_b)  {  signed int sum;  if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b)))  || ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b))))  {  /\* Handle error \*/  }  else  {  sum = si\_a + si\_b;  }  } |

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

| **Principles(s):** Validate Input Data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Sonar Lint | 6.10.0 | Sonar Lint | [SonarLint](https://www.sonarsource.com/products/sonarlint/) |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Copying data to a buffer that is not large enough to hold that data results in a buffer overflow. Buffer overflows occur frequently when manipulating strings [Seacord 2013b]. To prevent such errors, either limit copies through truncation or, preferably, ensure that the destination is of sufficient size to hold the character data to be copied and the null-termination character. (See STR03-C. Do not inadvertently truncate a string.)  When strings live on the heap, this rule is a specific instance of MEM35-C. Allocate sufficient memory for an object. Because strings are represented as arrays of characters, this rule is related to both ARR30-C. Do not form or use out-of-bounds pointers or array subscripts and ARR38-C. Guarantee that library functions do not form invalid pointers.  <https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator> |

| **Noncompliant Code** |
| --- |
| Off-by-One Error  This noncompliant code example demonstrates an off-by-one error [Dowd 2006]. The loop copies data from src to dest. However, because the loop does not account for the null-termination character, it may be incorrectly written 1 byte past the end of dest. |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {  size\_t i;    for (i = 0; src[i] && (i < n); ++i) {  dest[i] = src[i];  }  dest[i] = '\0';  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the loop termination condition is modified to account for the null-termination character that is appended to dest: |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

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

| **Principles(s):** Validate Input Data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Sonar Lint | 6.10.0 | Sonar Lint | [SonarLint](https://www.sonarsource.com/products/sonarlint/) |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL injection** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | SQL injection vulnerabilities arise in applications where elements of a SQL query originate from an untrusted source. Without precautions, the untrusted data may maliciously alter the query, resulting in information leaks or data modification. The primary means of preventing SQL injection are sanitization and validation, which are typically implemented as parameterized queries and stored procedures.  <https://learn.microsoft.com/en-us/cpp/data/oledb/issuing-a-parameterized-query?view=msvc-170>  <https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection> |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows code to authenticate a user to a system. The password is passed as a string, the database connection is created. Passwords must be hashed, but that is omitted from this example.  Unfortunately, this code example permits a SQL injection attack by incorporating the un-sanitized input argument username into the SQL command, allowing an attacker to inject validuser' OR '1'='1. |
| #include <atldbcli.h>  #include <iostream>  using namespace std;  void login(str username, str password)  {  CDataSource connection;  CSession session;  CCommand<CAccessor<CUser>> users;  LPCSTR clsid; // Initialize CLSID\_MSDASQL here  LPCTSTR pName = L"NWind";  connection.Open(clsid, pName, NULL, NULL, DBPROP\_AUTH\_INTEGRATED);  session.Open(connection);  users.Open(session, "SELECT \* FROM db\_user WHERE username = '"  + username + "' AND password = '" + password + "'");  // Get data from the rowset  while (users.MoveNext() == S\_OK)  {  cout << users.FirstName;  cout << users.LastName;  }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses a parametric query with a ? character as a placeholder for the argument. This code also validates the length of the username argument, preventing an attacker from submitting an arbitrarily long user name. |
| #include <atldbcli.h>  #include <iostream>  using namespace std;  void login(str username, str password)  {  CDataSource connection;  CSession session;  CCommand<CAccessor<CUser>> users;  LPCSTR clsid; // Initialize CLSID\_MSDASQL here  LPCTSTR pName = L"NWind";  connection.Open(clsid, pName, NULL, NULL, DBPROP\_AUTH\_INTEGRATED);  session.Open(connection);  // Set the parameter for the query  users.Username = username;  users.Password = password; // obviously this must be hashed. Different standard.  users.Open(session, "SELECT \* FROM db\_user WHERE username = ? AND password = ?");  // Get data from the rowset  while (users.MoveNext() == S\_OK)  {  cout << users.FirstName;  cout << users.LastName;  }  } |

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

| **Principles(s):** Validate Input Data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Sonar Lint | 6.10.0 | Sonar Lint | [SonarLint](https://www.sonarsource.com/products/sonarlint/) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Evaluating a pointer—including dereferencing the pointer, using it as an operand of an arithmetic operation, type casting it, and using it as the right-hand side of an assignment—into memory that has been deallocated by a memory management function is undefined behavior. Pointers to memory that has been deallocated are called dangling pointers. Accessing a dangling pointer can result in exploitable vulnerabilities.  It is at the memory manager's discretion when to reallocate or recycle the freed memory. When memory is freed, all pointers into it become invalid, and its contents might either be returned to the operating system, making the freed space inaccessible, or remain intact and accessible. As a result, the data at the freed location can appear to be valid but change unexpectedly. Consequently, memory must not be written to or read from once it is freed.  <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory> |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):** Architect and Design for Security Policies |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Moderate | Moderate | Moderate | Moderate | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Sonar Lint | 6.10.0 | Sonar Lint | [SonarLint](https://www.sonarsource.com/products/sonarlint/) |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Do not use assertions for user input validation** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | If the argument expression of this macro with functional form compares equal to zero (i.e., the expression is false), a message is written to the standard error device and abort is called, terminating the program execution.  Assertions should be disabled for production code, therefore user input validations should not be in assertions.  <https://cplusplus.com/reference/cassert/assert/> |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| #include <assert.h>  #include <iostream>  void print\_input(std::str userInput) {  assert (userInput != ""); // supposed to make sure user input isn’t empty.  std::cout << userInput << endl;  } |

| **Compliant Code** |
| --- |
| [Compliant description] |
| #include <assert.h>  #include <iostream>  void print\_input(std::str userInput) {  if (userInput == "")  {  std::cout << "User input cannot be an empty string\n";  }  else  {  std::cout << userInput << endl;  }  } |

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

| **Principles(s):** Architect and Design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Sonar Lint | 6.10.0 | Sonar Lint | [SonarLint](https://www.sonarsource.com/products/sonarlint/) |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | When an exception is thrown, control is transferred to the nearest handler with a type that matches the type of the exception thrown. If no matching handler is directly found within the handlers for a try block in which the exception is thrown, the search for a matching handler continues to dynamically search for handlers in the surrounding try blocks of the same thread. The C++ Standard, [except.handle], paragraph 9 [ISO/IEC 14882-2014], states the following:  If no matching handler is found, the function std::terminate() is called; whether or not the stack is unwound before this call to std::terminate() is implementation-defined.  The default terminate handler called by std::terminate() calls std::abort(), which abnormally terminates the process. When std::abort() is called, or if the implementation does not unwind the stack prior to calling std::terminate(), destructors for objects may not be called and external resources can be left in an indeterminate state. Abnormal process termination is the typical vector for denial-of-service attacks. For more information on implicitly calling std::terminate(), see ERR50-CPP. Do not abruptly terminate the program.  <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions> |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

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

| **Principles(s):** Architect and Design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Sonar Lint | 6.10.0 | Sonar Lint | [SonarLint](https://www.sonarsource.com/products/sonarlint/) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not use a bitwise operator with a Boolean-like operand** |
| --- | --- | --- |
| Bitwise Operators | STD-008-CPP | Mixing bitwise and relational operators in the same full expression can be a sign of a logic error in the expression where a logical operator is usually the intended operator. Do not use the bitwise AND (&), bitwise OR (|), or bitwise XOR (^) operators with an operand of type \_Bool, or the result of a relational-expression or equality-expression. If the bitwise operator is intended, it should be indicated with use of a parenthesized expression.  <https://wiki.sei.cmu.edu/confluence/display/c/EXP46-C.+Do+not+use+a+bitwise+operator+with+a+Boolean-like+operand> |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a bitwise & operator is used with the results of an equality-expression: |
| if (!(getuid() & geteuid() == 0))  {  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution uses the && operator for the logical operation within the conditional expression: |
| if (!(getuid() && geteuid() == 0))  {  /\* ... \*/  } |

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

| **Principles(s):** Keep It Simple |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Low | Low | Low | 0 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Sonar Lint | 6.10.0 | Sonar Lint | [SonarLint](https://www.sonarsource.com/products/sonarlint/) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Free dynamically allocated memory when no longer needed** |
| --- | --- | --- |
| Memory Management | STD-009-CPP | Before the lifetime of the last pointer that stores the return value of a call to a standard memory allocation function has ended, it must be matched by a call to free() with that pointer value.  <https://wiki.sei.cmu.edu/confluence/display/c/MEM31-C.+Free+dynamically+allocated+memory+when+no+longer+needed> |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the object allocated by the call to malloc() is not freed before the end of the lifetime of the last pointer text\_buffer referring to the object: |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }  return 0;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the pointer is deallocated with a call to free() |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }    free(text\_buffer);  return 0;  } |

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

| **Principles(s):** Architect and Design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Sonar Lint | 6.10.0 | Sonar Lint | [SonarLint](https://www.sonarsource.com/products/sonarlint/) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do not use floating-point variables as loop counters** |
| --- | --- | --- |
| Data Types | STD-010-CPP | Because floating-point numbers represent real numbers, it is often mistakenly assumed that they can represent any simple fraction exactly. Floating-point numbers are subject to representational limitations just as integers are, and binary floating-point numbers cannot represent all real numbers exactly, even if they can be represented in a small number of decimal digits.  In addition, because floating-point numbers can represent large values, it is often mistakenly assumed that they can represent all significant digits of those values. To gain a large dynamic range, floating-point numbers maintain a fixed number of precision bits (also called the significand) and an exponent, which limit the number of significant digits they can represent.  Different implementations have different precision limitations, and to keep code portable, floating-point variables must not be used as the loop induction variable. See Goldberg's work for an introduction to this topic [Goldberg 1991].  For the purpose of this rule, a loop counter is an induction variable that is used as an operand of a comparison expression that is used as the controlling expression of a do, while, or for loop. An induction variable is a variable that gets increased or decreased by a fixed amount on every iteration of a loop [Aho 1986]. Furthermore, the change to the variable must occur directly in the loop body (rather than inside a function executed within the loop).  <https://wiki.sei.cmu.edu/confluence/display/c/FLP30-C.+Do+not+use+floating-point+variables+as+loop+counters> |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a floating-point variable is used as a loop counter. The decimal number 0.1 is a repeating fraction in binary and cannot be exactly represented as a binary floating-point number. Depending on the implementation, the loop may iterate 9 or 10 times. |
| void func(void)  {  for (float x = 0.1f; x <= 1.0f; x += 0.1f)  {  /\* Loop may iterate 9 or 10 times \*/  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the loop counter is an integer from which the floating-point value is derived: |
| #include <stddef.h>    void func(void)  {  for (size\_t count = 1; count <= 10; ++count)  {  float x = count / 10.0f;  /\* Loop iterates exactly 10 times \*/  }  } |

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

| **Principles(s):** Architect and Design for Security Policies |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Low | Low | Low | 0 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Sonar Lint | 6.10.0 | Sonar Lint | [SonarLint](https://www.sonarsource.com/products/sonarlint/) |

### 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.

#### Assess and plan

During the planning phase of the DevSecOps process flow, any emerging threats are identified as news comes out regarding vulnerabilities. Any changes to regulations governing the software that the company produces must also be monitored closely to ensure adherence to all required policies. In many cases periodic audits are required to verify this, so planning must also account for development bandwidth to support in audits. In addition to all the new threats and regulations, the backlog of any outstanding vulnerabilities that were lower priority should be scheduled to be developed.

#### Design

Given all the new requirements for defending vulnerabilities and staying compliant with regulations, there is also a newly sorted backlog of features and fixes that need to be designed. Taking the current system into account, the architecture of the system should be designed, tweaked, or redesigned if necessary to meet previous requirements in addition to all the newly added ones. The design will most likely be a team effort because systems are often so complex, no human can have an in-depth knowledge of every part at the same time.

#### Build

During the build phase, the software is being written. Building also means that the company’s continuous integration/continuous delivery (CI/CD) pipeline is building the software and deploying to the appropriate environments as developers commit code. This phase also includes heavy unit testing, the goal of which is to achieve 100% code coverage. The goal of that level of code coverage may seem unrealistic, but the repositories in which I currently work have 80+%, and my code is always in the 90s when I check it in.

#### Verify and test

Once the code has been written, integration tests should be written to ensure that all parts of the system are working together as desired, and that any vulnerabilities identified during the planning phase of the operation are protected adequately. Testing may also involve certifications with third party integrations. In order to make those certification processes run very smoothly, a suite of test cases with the certification data can be set up prior, and then executed on command during the certification.

#### Transition and health check

Transitioning from the development, or pre-production phase into the production phase means that the system is now running in a live environment. It must be double and triple checked that all required configurations have been completed in the production environment. A common error, at least for me, is to get everything configured perfectly in lower environments where tests can be run repeatedly and fixed on the fly without affecting production data. In production, in order to have a clean environment without a bunch of test data clogging the system, the configuration should all be planned beforehand and executed at once.

#### Monitor and detect

The Monitoring and detection phase involved tuning alerts for error logs, maintaining on call status for developers to be able to respond even outside of normal business hours in the event of an anomaly. Setting up good logging during the build phase is going to pay dividends during this phase because clear logs of errors and what the system is doing make a big difference in tracking down issues in a production environment where there is very little insight into issues outside of the logs.

#### Respond

In the event that an attack or attacks are detected, the team must be prepared to respond and know what to do during an attack. Just as with any other skill, it must be practiced to maintain the highest level of proficiency. Some companies practice by intentionally causing mayhem in the services to ensure that there is sufficient redundancy in both services and connections to keep the service running during an attack.

#### Maintain and stabilize

Now that the system is in equilibrium, it must still be maintained by scaling appropriately for the number of users, distributed geographically based on user locations and to ensure redundancy in the event of an issue, and further stabilization can take place based on any lessons learned during the previous phases.

### 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 | Low | Moderate | Low | Low | 1 |
| STD-002-CPP | Moderate | Moderate | Low | Moderate | 2 |
| STD-003-CPP | High | Moderate | Low | High | 3 |
| STD-004-CPP | High | Moderate | Moderate | High | 4 |
| STD-005-CPP | Moderate | Moderate | Moderate | Moderate | 2 |
| STD-006-CPP | Low | Low | Low | Low | 1 |
| STD-007-CPP | Low | Low | High | Low | 1 |
| STD-008-CPP | Low | Low | Low | Low | 0 |
| STD-009-CPP | High | Moderate | High | High | 3 |
| STD-010-CPP | Low | Low | Low | Low | 0 |

### 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 at rest is when it is being stored in a system such as a database. The policy of encryption at rest means that when data is written to the storage media, it is encrypted as part of the database transaction. Data at rest must be encrypted always in production, and data in lower environments should be adequately redacted to not compromise production data. |
| Encryption in flight | Data in flight is when it is being transmitted from system to system or within a single system. An example of data in flight is a request made on a web page, the request data is in flight as it is transmitted through the internet to the application. Data in flight should be encrypted using TLS 1.3 or higher in all testing environments and production, but it is acceptable for local testing to be done in the clear if the request stays within company networks. |
| Encryption in use | Data in use is when data is actively being processed by a system. Where possible, data should be encrypted during processing. This also applies to parts of a record that are needed later on in a process, but are not being processed at that time. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication proves that the user is who they say they are. Login via username and password with multi-factor auth is an acceptable form of authorization for company systems. |
| Authorization | Authorization determines what resources, and what permissions on those resources an authenticated user has. The authorization policy is that each user and service should have the minimum user level of access required to accomplish their task and that permissions should fail closed. The policy always applies to all users. Only specific authorized users are granted write access, or the ability to apply changes to the database. An admin credentials is required for the addition of new users. |
| Accounting | Accounting is the practice of counting API usage for a particular auth token to appropriately bill the user of use of the system. The accounting policy applies mainly to users who are on an API usage billing schedule. It is also a useful diagnostic tool, but primarily applies to billing. For users who have a billing plan based on Files accessed by the users, a count of those access requests are logged and provided to the accounting department. |

**\***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 | 10/09/2022 | Project 1 | Mark Holden |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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