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 | Input validation is the process of analyzing inputs and disallowing those which are considered unsuitable. It only allows inputs that meet specific criteria which makes it impossible for attackers to enter an input designed to cause harm to a system. |
| 1. Heed Compiler Warnings | This is a warning to let developers know of potential error/issue in code. They indicate things that might cause problems that a developer is not aware of. |
| 1. Architect and Design for Security Policies | When implementing security policies software architecture and design is considered.  Example: A system that needs different privileges at different times  - Break the system apart into subsystems.  - Recall: Secure State Machine Design Pattern |
| 1. Keep It Simple | Designs should be kept simple and small to reduce errors from developing when coding. Complex designs lead to more errors and less security assurance |
| 1. Default Deny | This is when access is denied by default and access is only granted if conditions are met. |
| 1. Adhere to the Principle of Least Privilege | Processes should execute with the minimal set of privileges. Only hold elevated privileges for a short time to reduce the chances of privilege escalation or execution of protected code. |
| 1. Sanitize Data Sent to Other Systems | This is to avoid injection attacks. Unused functions may pass and cause damages like SQL injection. Sanitizing data before passing out info to other systems will check for these issues before they go out. |
| 1. Practice Defense in Depth | This means creating multiple layers of defense so it will not be vulnerable to attackers. |
| 1. Use Effective Quality Assurance Techniques | This means doing proper testing, penetration testing, and more testing can be effective in quality assurance techniques. |
| 1. Adopt a Secure Coding Standard | Applying coding standards to be secure when writing code. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not declare or define a reserved identifier |

| **Noncompliant Code** |
| --- |
| A common practice is to use a macro in preprocessor conditional that guards against multiple inclusions of a header file. Many programs use reserved name as the header guard. |
| *#ifndef \_MY\_HEADER\_H\_*  *#define \_MY\_HEADER\_H\_*    *// Contents of <my\_header.h>*    *#endif // \_MY\_HEADER\_H\_* |

| **Compliant Code** |
| --- |
| This compliant solution avoids using leading or trailing underscores in the name of the header guard. |
| *#ifndef MY\_HEADER\_H*  *#define MY\_HEADER\_H*    *// Contents of <my\_header.h>*    *#endif // MY\_HEADER\_H* |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Reserved identifier | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL51 |  |
| Clang | 3.9 | -Wreserved-id-macro  -Wuser-defined-literals | The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverything. This flag does not  catch all instances of this rule, such as redefining reserved names. |
| CodeSonar | 8.1p0 | LANG.ID.NU.MK  LANG.STRUCT.DECL.RESERVED | Macro name is C keyword  Declaration of reserved name |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not cast or delete pointers to incomplete classes |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a class attempts to implement the pimpl idiom but deletes a pointer to an incomplete class type, resulting in undefined behavior if Body has a nontrivial destructor. |
| *class Handle {*  *class Body \*impl; // Declaration of a pointer to an incomplete class*  *public:*  *~Handle() { delete impl; } // Deletion of pointer to an incomplete class*  *// ...*  *};* |

| **Compliant Code** |
| --- |
| In this compliant solution, the deletion of impl is moved to a part of the code where the body is defined. |
| *class Handle {*  *class Body \*impl; // Declaration of a pointer to an incomplete class*  *public:*  *~Handle();*  *// ...*  *};*    *// Elsewhere*  *class Body { /\* ... \*/ };*    *Handle::~Handle() {*  *delete impl;*  *}* |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2023b | CERT C++: EXP57-CPP | Checks for conversion or deletion of incomplete class pointer |
| Coverity | 6.5 | DELETE\_VOID | Fully implemented |
| CodeSonar | 8.1p0 | LANG.CAST.PC.INC | Conversion: pointer to incomplete |
| LDRA tool suite | 9.7.1 | 169 S, 554 S | Enhanced Enforcement |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::string object is created from the results of a call to std::getenv(). |
| *#include <cstdlib>*  *#include <string>*    *void f() {*  *std::string tmp(std::getenv("TMP"));*  *if (!tmp.empty()) {*  *// ...*  *}*  *}* |

| **Compliant Code** |
| --- |
| In this compliant solution, the results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| *#include <cstdlib>*  *#include <string>*    *void f() {*  *const char \*tmpPtrVal = std::getenv("TMP");*  *std::string tmp(tmpPtrVal ? tmpPtrVal : "");*  *if (!tmp.empty()) {*  *// ...*  *}*  *}* |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Code Sonar | 8.1p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2023b | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule partially covered). |
| Astrée | 22.10 | Assert\_failure |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL Injection |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows JDBC code to authenticate a user to a system. The password is passed as a char array, the database connection is created, and then the passwords are hashed. |
| *import java.sql.Connection;*  *import java.sql.DriverManager;*  *import java.sql.ResultSet;*  *import java.sql.SQLException;*  *import java.sql.Statement;*    *class Login {*  *public Connection getConnection() throws SQLException {*  *DriverManager.registerDriver(new*  *com.microsoft.sqlserver.jdbc.SQLServerDriver());*  *String dbConnection =*  *PropertyManager.getProperty("db.connection");*  *// Can hold some value like*  *// "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"*  *return DriverManager.getConnection(dbConnection);*  *}* |

| **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. |
| *public void doPrivilegedAction(*  *String username, char[] password*  *) throws SQLException {*  *Connection connection = getConnection();*  *if (connection == null) {*  *// Handle error*  *}*  *try {*  *String pwd = hashPassword(password);*    *// Validate username length*  *if (username.length() > 8) {*  *// Handle error*  *}* |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting checker | Trust and security errors |
| CodeSonar | 8.1p0 | JAVA.IO.INJ.SQL | SQL Injection(Java) |
| Parasoft Jtest | 2023.1 | CERT.IDS00.TDSQL | Project against SQL injection |
| SonarQube | 9.9 | S2077  S3649 | Executing SQL queries is security-sensitive  SQL queries should not be vulnerable to injection attacks |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-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. |
| *#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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| CodeSonar | 8.1p0 | ALLOC.UAF | Use after free |
| Coverity | V7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Understand the termination behavior of assert() and abort() |

| **Noncompliant Code** |
| --- |
| This noncompliant code example defines a function that is called before the program exits to clean up. |
| void cleanup(void) {  /\* Delete temporary files, restore consistent state, etc. \*/  }    int main(void) {  if (atexit(cleanup) != 0) {  /\* Handle error \*/  }    /\* ... \*/    assert(/\* Something bad didn't happen \*/);    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the call to assert() is replaced with an if statement that calls exit() to ensure that the proper termination routines are run |
| *void cleanup(void) {*  */\* Delete temporary files, restore consistent state, etc. \*/*  *}*    *int main(void) {*  *if (atexit(cleanup) != 0) {*  */\* Handle error \*/*  *}*    */\* ... \*/*    *if (/\* Something bad happened \*/) {*  *exit(EXIT\_FAILURE);*  *}*    */\* ... \*/*  *}* |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 8.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not return from a computational exception signal handler |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the division operation has undefined behavior if denom equals 0. |
| *#include <errno.h>*  *#include <limits.h>*  *#include <signal.h>*  *#include <stdlib.h>*    *volatile sig\_atomic\_t denom;*    *void sighandle(int s) {*  */\* Fix the offending volatile \*/*  *if (denom == 0) {*  *denom = 1;*  *}*  *}* |

| **Compliant Code** |
| --- |
| The only portably safe way to leave a SIGFPE, SIGILL, or SIGSEGV handler is to invoke abort(), quick\_exit(), or\_Exit(). |
| *#include <errno.h>*  *#include <limits.h>*  *#include <signal.h>*  *#include <stdlib.h>*    *int main(int argc, char \*argv[]) {*  *if (argc < 2) {*  *return 0;*  *}*    *char \*end = NULL;*  *long denom = strtol(argv[1], &end, 10);*    *if (end == argv[1] || 0 != \*end ||*  *((LONG\_MIN == denom || LONG\_MAX == denom) && errno == ERANGE)) {*  */\* Handle error \*/*  *}*    *long result = 100 / denom;*  *return 0;*  *}* |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.STRUCT.RFCESH | Return from computational Exception Signal Handler |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced enforcement |
| Parasoft C/C++test | 2023.1 | CERT\_C-SIG35-a | Do not return from a computational exception signal handler |
| Polyspace Bug Finder | R2023b | CERT C: Rule SIG35-C | Checks for return from computational exception signal handler (rule fully covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | [STD-008-CPP] | Do not access a variable through a pointer of an incompatible type |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, an object of type float is incremented through an int \*. The programmer can use the unit in the last place to get the next representable value for a floating-point type. |
| *#include <stdio.h>*    *void f(void) {*  *if (sizeof(int) == sizeof(float)) {*  *float f = 0.0f;*  *int \*ip = (int \*)&f;*  *(\*ip)++;*  *printf("float is %f\n", f);*  *}*  *}* |

| **Compliant Code** |
| --- |
| In this compliant solution, the standard C function nextafterf() is used to round toward the highest representable floating -point value. |
| *#include <float.h>*  *#include <math.h>*  *#include <stdio.h>*    *void f(void) {*  *float f = 0.0f;*  *f = nextafterf(f, FLT\_MAX);*  *printf("float is %f\n", f);*  *}* |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2024.1 | C0310, C0751, C3305  C++3017, C++3030, C++3033 |  |
| LDRA tool suite | 9.7.1 | 94 S, 554 S | Partially implemented |
| Parasoft C/C++test | 2023.1 | CERT\_C-EXP39-a | There shall be no implicit conversions from integral to floating type |
| Polyspace Bug Finder | R2023b | CERT C: Rule EXP39-C | Checks for cast to pointer pointing to object of different type (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers | [STD-009-CPP] | Ensure that operations on signed integers do not result in overflow |

| **Noncompliant Code** |
| --- |
| 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | integer-overflow | Fully checked |
| LDRA tool suite | 9.7.1 | 493 S, 494 S | Partially implemented |
| Parasoft C/C++test | 2023.1 | CERT\_C-INT32-a  CERT\_C-INT32-b  CERT\_C-INT32-c | Avoid signed integer overflows  Integer overflow or underflow in constant expression in '+', '-', '\*' operator  Integer overflow or underflow in constant expression in '<<' operator |
| TrustInSoft Analyzer | 1.38 | signed\_overflow | Exhaustively verified |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [STD-010-CPP] | Exclude user input from format strings |

| **Noncompliant Code** |
| --- |
| The incorrect\_password() function in this noncompliant code example is called during identification and authentication to display an error message if the specified user is not found or the password is incorrect. The function accepts the name of the user as a string referenced by user. |
| *#include <stdio.h>*  *#include <stdlib.h>*  *#include <string.h>*    *void incorrect\_password(const char \*user) {*  *int ret;*  */\* User names are restricted to 256 or fewer characters \*/*  *static const char msg\_format[] = "%s cannot be authenticated.\n";*  *size\_t len = strlen(user) + sizeof(msg\_format);*  *char \*msg = (char \*)malloc(len);*  *if (msg == NULL) {*  */\* Handle error \*/*  *}* |

| **Compliant Code** |
| --- |
| This compliant solution fixes the problem by replacing the fprintf() call with a call to fputs(), which outputs msg directly to stderr without evaluating its contents. |
| *#include <stdio.h>*  *#include <stdlib.h>*  *#include <string.h>*    *void incorrect\_password(const char \*user) {*  *int ret;*  */\* User names are restricted to 256 or fewer characters \*/*  *static const char msg\_format[] = "%s cannot be authenticated.\n";*  *size\_t len = strlen(user) + sizeof(msg\_format);*  *char \*msg = (char \*)malloc(len);*  *if (msg == NULL) {*  */\* Handle error \*/*  *}* |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO30 | Partially implemented |
| CodeSonar | 8.1p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| PC-lint Plus | 1.4 | 592 | Partially supported: reports non-literal format strings |
| Polyspace Bug Finder | R2023b | CERT C: Rule FIO30-C | Checks for tainted string format (rule partially covered) |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | Medium | Unlikely | Medium | Low(4) | 3 |
| STD-003-CPP | High | Likely | Medium | High(18) | 1 |
| STD-004-CPP | High | Likely | Medium | High(18) | 1 |
| STD-005-CPP | High | Likely | Medium | High(18) | 1 |
| STD-006-CPP | Low | Unlikely | High | Low(1) | 3 |
| STD-007-CPP | Low | Unlikely | High | Low(1) | 3 |
| STD-008-CPP | Medium | Unlikely | High | Low(2) | 3 |
| STD-009-CPP | High | Likely | High | High(9) | 2 |
| STD-010-CPP | High | Likely | Medium | High(18) | 1 |

### 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 is encryption that is used to help protect data that is stored on a disk (including solid-state drives) or backup media |
| Encryption in flight | Encryption in flight protects contents in transit. Encryption at rest maintains protection at the remote destination until the recipient applies the proper passphrase. |
| Encryption in use | Encryption in use enables computations on ciphertext in a way that, once you decrypt data, the output is the same as if the computer operated with plaintext files. |
| Citations | * Google. (n.d.). Default encryption at rest  |  documentation  |  google cloud. Google. <https://cloud.google.com/docs/security/encryption/default-encryption> * Content encryption: In Flight and at rest. (n.d.). <https://www.ibm.com/docs/en/aspera-on-cloud?topic=encryption-content-in-flight-rest> * Velimirovic, A. (2023, November 16). Data Encryption in use explained. phoenixNAP Blog. https://phoenixnap.com/blog/encryption-in-use#:~:text=Encryption%20in%20use%20enables%20computations,data%20into%20memory%20for%20processing. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the act of confirming the truth of an attribute of a single piece of data claimed true by an entity. In contrast with identification, which refers to the act of stating or otherwise indicating a claim purportedly attesting to a person or thing’s identity |
| Authorization | Authorization is the function of specifying access rights/privileges to resources related to information security and computer security in general and to access control in particular |
| Accounting | This is the process that keeps track of a user’s activity while attached to a system; the trail included the amount of time attached, the resources accessed, and how much data transferred. Accounting data is used for trending, detecting breaches, and forensic investigating. |
| Citation | * Authentication, authorization, accounting and Identity Management. CCSI, A Sourcepass Company. (2019, April 29). https://www.ccsinet.com/blog/aaa-identity-management/ |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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 |