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



# Green Pace Secure Development Policy

# Joseph Grzywinski

## 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 | Any data that is input into the system should be validated to ensure that it is not malicious. This will prevent attacks using SQL injections against our systems. |
| 1. Heed Compiler Warnings | Any compiler warnings that are prompted through the system should be documented and taken care of. A lot of these can be due to updates for different dependencies that, if they are not fixed, could cause security issues in the future. |
| 1. Architect and Design for Security Policies | When structuring and planning for new applications or updates, architects must fit security policies into the applications design to ensure developers build it into the systems. While developers should already be doing this, putting into design documents will make it officially a part of development. |
| 1. Keep It Simple | While making applications that are not complex is normally not possible, we want to find the simplest possible solutions to each problem that still ensures a working a secure application. The more complex solution that we create for the application, the more susceptible it can be to attacks and securing the application can be more difficult. |
| 1. Default Deny | By default, new accounts will be denied access to the systems until either the proper credentials are provided or configured by an administrator. By limiting access for new accounts, it is easier to track who has access to which parts of the system and what capabilities each account has. It also will help keep bad actors from creating new accounts and gaining full access. |
| 1. Adhere to the Principle of Least Privilege | When granting system access to either a user or other programs within the company, give the least amount of access possible for them to complete their job. Giving more access than necessary can come with significant risk if accounts are compromised as it will give the hacker more access. |
| 1. Sanitize Data Sent to Other Systems | Before transferring or sending data to different systems, ensure that only necessary data is being sent and all sensitive information is either removed or encoded so it is not accessible by anyone that may unexpectedly gain access to these databases. |
| 1. Practice Defense in Depth | When designing all applications and internal systems we must use layered security at the network, application, and database levels. The more layers of security there are within our system the less likely a breach will affect the system in any way as any hackers will have to get past each layer to get any valuable information. At the same time we must use a reasonable amount of layers as too many can slow our systems down or make it more difficult to maintain in the future. |
| 1. Use Effective Quality Assurance Techniques | To maintain coding standards, we should have QA teams working alongside development teams to do constant testing. Developers should also unit test through development to ensure code is up to production standards. |
| 1. Adopt a Secure Coding Standard | Ensure that all developers and QA are following the same secure coding standards. This consistency with a set standard will ensure that every application has the same security making it easier as a company to make updates to security and track leaks or vulnerabilities quickly. |

### 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] | Never qualify a reference type with const or volatile |

| **Noncompliant Code** |
| --- |
| This code correctly declares p to be a reference to a const-qualified char. The subsequent modification of p makes the program ill-formed. |
| #include <iostream>    **void** f(**char** c) {  **const** **char** &p = c;    p = 'p';    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| The solution to this error would be to remove the const qualifier. |
| #include <iostream>    **void** f(**char** c) {  **char** &p = c;    p = 'p';    std::cout << c << std::endl;  } |

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

| **Principles(s):** This goes in line with the principle ‘Validate Input Data’ since const as a reference type can cause us not to be able to validate input data properly to keep security standards. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL52** | [Insert text.] |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |
| Polyspace Bug Finder | R2024a | CERT C++: DCL52-CPP | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| SonarQube C/C++ Plugin | 4.10 | S3708 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not rely on the value of a moved-from object |

| **Noncompliant Code** |
| --- |
| The integer values 0 through 9 are expected to be printed to the standard output stream from a std::string rvalue reference. However, because the object is moved and then reused under the assumption its internal state has been cleared, unexpected output may occur despite not triggering undefined behavior. |
| #include <iostream>  #include <string>    **void** g(std::string v) {    std::cout << v << std::endl;  }    **void** f() {    std::string s;  **for** (unsigned i = 0; i < 10; ++i) {      s.append(1, **static\_cast**<**char**>('0' + i));      g(std::move(s));    }  } |

| **Compliant Code** |
| --- |
| The std::string object is initialized to the expected value on each iteration of the loop. This practice ensures that the object is in a valid, specified state prior to attempting to access it in g(), resulting in the expected output. |
| #include <iostream>  #include <string>    **void** g(std::string v) {    std::cout << v << std::endl;  }    **void** f() {  **for** (unsigned i = 0; i < 10; ++i) {      std::string s(1, **static\_cast**<**char**>('0' + i));      g(std::move(s));    }  } |

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

| **Principles(s):** This coding standard follows the ‘Practice defense in Depth’ security principle since values from objects need to be validated as they move throughout an application with various levels of security. These layers of security for different portions of the application are the core values of defense in depth. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.MEM.NPD LANG.MEM.UVAR | Null Pointer Dereference Uninitialized Variable |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-EXP63-a | Do not rely on the value of a moved-from object |
| Polyspace Bug Finder | R2024a | CERT C++: EXP63-CPP | Checks for read operations that reads the value of a moved-from object |
| PVS-Studio | 7.34 | V1030 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Use valid references, pointers, and iterators to reference elements of a basic string |

| **Noncompliant Code** |
| --- |
| This example copies input into a std::string, replacing semicolon (;) characters with spaces. This example is noncompliant because the iterator loc is invalidated after the first call to insert(). The behavior of subsequent calls to insert() is undefined. |
| #include <string>    **void** f(**const** std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();  **for** (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

| **Compliant Code** |
| --- |
| The value of the iterator loc is updated because of each call to insert() so that the invalidated iterator is never accessed. The updated iterator is then incremented at the end of the loop. |
| #include <string>    **void** f(**const** std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();  **for** (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      loc = email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

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

| **Principles(s):** This goes in line with the principle ‘Validate Input Data’ since string inputs referenced by a pointer should always be validated as correct before using or changing it. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.UAF | Use After Free |
| Helix QAC | 2024.4 | DF4746, DF4747, DF4748, DF4749 | [Insert text.] |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR52-a | Use valid references, pointers, and iterators to reference elements of a basic\_string |
| Polyspace Bug Finder | R2024a | CERT C++: STR52-CPP | Checks for use of invalid string iterator |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-003-CPP] | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| This code can result in a SQL injection attack as it has no checks for common SQL injection patterns. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  records.clear();  char\* error\_message;  if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  } |

| **Compliant Code** |
| --- |
| By checking the SQL string as it comes in, we can weed out injection attacks by looking for frequently used words. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  records.clear();  std::array<std::string, 4> sql\_injections = { "or ", "OR ", "and ", "AND " };  for (std::string sql\_injection : sql\_injections) {  if (sql.find(sql\_injection) != std::string::npos) {  std::cout << "SQL Injection found" << std::endl;  return false;  }  }  char\* error\_message;  if (sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  } |

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

| **Principles(s):** This goes in line with the principle ‘Validate Input Data’ since SQL injection happens when a bad actor manipulates a SQL call to gain access to data. We must use proper checks on SQL calls to prevent this. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**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 |
| Coverity | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Fortify | 1.0 | HTTP\_Response\_Splitting SQL\_Injection\_\_Persistence SQL\_Injection | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| This example shows both the incorrect and correct techniques for freeing the memory associated with a linked list. In their incorrect example, p is freed before p->next is executed, so that p->next reads memory that has already been freed. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| You would solve this error by storing a reference to p->next  in q before freeing p. |
| #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):** Freed memory, if accessed, should throw warnings to the developer during development and testing. ‘Heed compiler warnings’ would be a security principle used to uphold this coding standard as developers should listen to these warnings and refactor their program, as necessary. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | dangling\_pointer\_use | Supported  Astrée reports all accesses to freed allocated memory. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM30 | Detects memory accesses after its deallocation and double memory deallocations |
| CodeSonar | 8.1p0 | ALLOC.UAF | Use after free |
| Coverity | 2017.07 | 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 |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly |
| #include <assert.h>    **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    **int** func(**void**) {  **assert**(**sizeof**(**struct** timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**));  } |

| **Compliant Code** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution. |
| struct timer {    unsigned char MODE;    unsigned int DATA;    unsigned int COUNT;  };    static\_assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int),                "Structure must not have any padding"); |

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

| **Principles(s):** static code tests should be apart of any applications we develop, and we need to plan for the creation and execution of these tests right from the beginning. This coding standard would fall under ‘Adopt a secure coding standard’ since all developers should do static test creation. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | misc-static-assert | [Insert text.] |
| CodeSonar | 8.1p0 | Custom | [Insert text.] |
| ECLAIR | 1.2 | CC2.DCL03 | [Insert text.] |
| LDRA tool suite | 9.7.1 | 44 S | [Insert text.] |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| The division operation has undefined behavior if denom equals zero. |
| #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;    }  }    **int** main(**int** argc, **char** \*argv[]) {  **if** (argc < 2) {  **return** 0;    }    **char** \*end = NULL;  **long** temp = **strtol**(argv[1], &end, 10);    **if** (end == argv[1] || 0 != \*end ||        ((LONG\_MIN == temp || LONG\_MAX == temp) && **errno** == ERANGE)) {      /\* Handle error \*/    }      denom = (**sig\_atomic\_t**)temp;  **signal**(SIGFPE, sighandle);    **long** result = 100 / (**long**)denom;  **return** 0;  } |

| **Compliant Code** |
| --- |
| The only portably safe way to leave a SIGFPE, SIGILL, or SIGSEGV handler is to invoke abort(), quick\_exit(), or \_Exit(). In the case of SIGFPE, the default action is abnormal termination, so no user-defined handler is required. |
| #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):** Normally if you were to return a signal handler, a warning will be displayed to the developer, so this coding standard would fall under ‘heed compiler warnings’ as it is something that developer should fix when that warning triggers in the console. |
| --- |

**Threat Level**

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

**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 |
| PC-lint Plus | 1.4 | 2671, 2764 | Fully supported |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Error Handling** | [STD-008-CPP] | Detect and handle standard library errors |

| **Noncompliant Code** |
| --- |
| In this example temp\_num, tmp2, and num\_of\_records are derived from a tainted source. As a result, an attacker can easily cause calloc() to fail by providing a large value for num\_of\_records. |
| #include <stdlib.h>  #include <string.h>    **enum** { SIG\_DESC\_SIZE = 32 };    **typedef** **struct** {  **char** sig\_desc[SIG\_DESC\_SIZE];  } signal\_info;    **void** func(**size\_t** num\_of\_records, **size\_t** temp\_num,  **const** **char** \*tmp2, **size\_t** tmp2\_size\_bytes) {    signal\_info \*start = (signal\_info \*)**calloc**(num\_of\_records,  **sizeof**(signal\_info));    **if** (tmp2 == NULL) {      /\* Handle error \*/    } **else** **if** (temp\_num > num\_of\_records || temp\_num == 0) {      /\* Handle error \*/    } **else** **if** (tmp2\_size\_bytes < SIG\_DESC\_SIZE) {      /\* Handle error \*/    }      signal\_info \*point = start + temp\_num - 1;  **memcpy**(point->sig\_desc, tmp2, SIG\_DESC\_SIZE);    point->sig\_desc[SIG\_DESC\_SIZE - 1] = '\0';    /\* ... \*/  **free**(start);  } |

| **Compliant Code** |
| --- |
| This example ensures the pointer returned by calloc() is not null. |
| #include <stdlib.h>  #include <string.h>    **enum** { SIG\_DESC\_SIZE = 32 };    **typedef** **struct** {  **char** sig\_desc[SIG\_DESC\_SIZE];  } signal\_info;    **void** func(**size\_t** num\_of\_records, **size\_t** temp\_num,  **const** **char** \*tmp2, **size\_t** tmp2\_size\_bytes) {    signal\_info \*start = (signal\_info \*)**calloc**(num\_of\_records,  **sizeof**(signal\_info));  **if** (start == NULL) {      /\* Handle allocation error \*/    } **else** **if** (tmp2 == NULL) {      /\* Handle error \*/    } **else** **if** (temp\_num > num\_of\_records || temp\_num == 0) {      /\* Handle error \*/    } **else** **if** (tmp2\_size\_bytes < SIG\_DESC\_SIZE) {      /\* Handle error \*/    }      signal\_info \*point = start + temp\_num - 1;  **memcpy**(point->sig\_desc, tmp2, SIG\_DESC\_SIZE);    point->sig\_desc[SIG\_DESC\_SIZE - 1] = '\0';    /\* ... \*/  **free**(start);  } |

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

| **Principles(s):** Errors within a coding library can be caught during development or even later down the line as updates can be made to libraries or codebases that may cause current libraries to have vulnerabilities or to not work at all. With the security principle ‘Use effective quality assurance techniques’ we can stay on top of library errors even after code is deployed and make the proper updates to the libraries used. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | error-information-unused error-information-unused-computed | Partially checked |
| CodeSonar | 8.1p0 | LANG.FUNCS.IRV LANG.ERRCODE.NOTEST LANG.ERRCODE.NZ | Ignored return value Missing Test of Error Code Non-zero Error Code |
| Coverity | 2017.07 | MISRA C 2012 Rule 22.8  MISRA C 2012 Rule 22.9  MISRA C 2012 Rule 22.10 | Implemented |
| LDRA tool suite | 9.7.1 | 80 D | Partially implemented |

#### 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** |
| --- |
| Since there is no check for overflow this code 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 ‘if’ statement in this code 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):** This standard would fall under ‘adopt a secure coding standard’ since proper checks should always be set in place to avoid overflow within code. This should be a consistent practice during development for all programmers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | **integer-overflow** | Fully checked |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| Cppcheck Premium | 24.11.0 | premium-cert-int32-c | [Insert text.] |
| LDRA tool suite | 9.7.1 | 493 S, 494 S | Partially implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Pointers** | [STD-010-CPP] | Do not store pointers returned by certain functions |

| **Noncompliant Code** |
| --- |
| Attempts to compare the value of the TMP and TEMP environment variables to determine if they are the same. |
| #include <stdlib.h>  #include <string.h>  #include <stdio.h>    **void** func(**void**) {  **char** \*tmpvar;  **char** \*tempvar;      tmpvar = **getenv**("TMP");  **if** (!tmpvar) {      /\* Handle error \*/    }    tempvar = **getenv**("TEMP");  **if** (!tempvar) {      /\* Handle error \*/    }  **if** (**strcmp**(tmpvar, tempvar) == 0) {  **printf**("TMP and TEMP are the same.\n");    } **else** {  **printf**("TMP and TEMP are NOT the same.\n");    }  } |

| **Compliant Code** |
| --- |
| Uses the malloc() and strcpy() functions to copy the string returned by getenv() into a dynamically allocated buffer. |
| #include <stdlib.h>  #include <string.h>  #include <stdio.h>    **void** func(**void**) {  **char** \*tmpvar;  **char** \*tempvar;    **const** **char** \*temp = **getenv**("TMP");  **if** (temp != NULL) {      tmpvar = (**char** \*)**malloc**(**strlen**(temp)+1);  **if** (tmpvar != NULL) {  **strcpy**(tmpvar, temp);      } **else** {        /\* Handle error \*/      }    } **else** {      /\* Handle error \*/    }      temp = **getenv**("TEMP");  **if** (temp != NULL) {      tempvar = (**char** \*)**malloc**(**strlen**(temp)+1);  **if** (tempvar != NULL) {  **strcpy**(tempvar, temp);      } **else** {        /\* Handle error \*/      }    } **else** {      /\* Handle error \*/    }    **if** (**strcmp**(tmpvar, tempvar) == 0) {  **printf**("TMP and TEMP are the same.\n");    } **else** {  **printf**("TMP and TEMP are NOT the same.\n");    }  **free**(tmpvar);  **free**(tempvar);  } |

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

| **Principles(s):** Having checks to ensure pointers do not use values from functions that return null can prevent this from happening and is a basic layer of defense against vulnerabilities caused by this bad coding practice. This would fall under defense in depth as it would be a basic level of security for the application that would be a part of a layered security pattern. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck Premium | 24.9.0 | premium-cert-env34-c |  |
| LDRA tool suite | 9.7.1 | 133 D | Fully implemented |
| Parasoft C/C++test | 2023.1 | CERT\_C-ENV34-a | Pointers returned by certain Standard Library functions should not be used following a subsequent call to the same or related function |
| Polyspace Bug Finder | R2024a | CERT C: Rule ENV34-C | Checks for misuse of return value from nonreentrant standard function |

**References:** Sei cert C++ coding standard. SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682

### 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.

A lot of the automation tools discussed in this document will be used within the Pre-production phase of the applications lifecycle. Tools such as sonar scans or Cppcheck will be implemented before the builds are sent to QA teams for testing. Developers should be running checks like this to speed up the development as it would take much more time to send to QA for them to run tests like this and send them back to the developer with a vulnerability list. Instead, a developer should have access to the list before submitting for testing so they can deal with issues on site and in real time.

### 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 | Unlikely | Low | Low | 3 |
| STD-002-CPP | Medium | Probable | Medium | High | 2 |
| STD-003-CPP | High | Probable | High | Medium | 2 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Low | Unlikely | High | Low | 3 |
| STD-008-CPP | High | Likely | Medium | High | 1 |
| STD-009-CPP | High | Likely | High | Medium | 2 |
| STD-010-CPP | Low | Probable | Medium | Low | 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 is used to always encrypt data even when not in use. In our application, all local data should be encrypted and secure in the case that data that may not be in use is compromised. A lot of the security in place is reactive and will be based on the actions of bad actors trying to get into the systems but with this proactive approach, even if the data is stolen the bad actor will not be able to access it. |
| Encryption in flight | This policy is meant to secure data as it is being transferred through a network either between applications or between the application and the user. With data being transferred for the customer all these communications should be encrypted to prevent bad actors from intercepting transmissions as they will most likely have customer information within them. This can be done with SSL. |
| Encryption in use | This policy is meant to ensure that data is encrypted while it is being used. For our company’s application we want to ensure that any of the entrepreneurs are accessing their data for either information or changing data within their profile or dataset. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This policy is meant to check a user to make sure that they are authorized to access the information being requested. This would be used for customers logging in to make sure they have the correct credentials as well as a secondary check such as a commonly used device or an email validation to make sure their log-in data was not compromised. |
| Authorization | This policy determines the permissions that are given to a type of user. This would range from customer to developer to admin in the company as a user should not be able to modify certain parts of the application or access certain data while an admin should have access to a majority or the applications services. |
| Accounting | The accounting policy is meant to be a more reactive approach after authentication and authorization have been set. This policy will monitor user activity to see if any suspicious activity is taking place while their account is being accessed. In our case if any bad actors were to gain access to an account this monitoring system should be able to see if they are doing something out of the ordinary like trying to access parts of the application that they may be blocked from or even making purchases that are out of the ordinary for that account. |

**\***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 | 12/08/2024 | Initial Template | Joseph Grzywinski |  |
| [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 |