**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 | Validating input data involves checking input from users for functionality and security purposes. It is used to make sure the input being entered meets what the program is looking for and that it is not malicious. Incorrect input length or type can lead to buffer overflows and injection attacks. |
| 1. Heed Compiler Warnings | Heeding compiler warnings involves paying attention to warnings given by the compiler. These warnings will flag vulnerabilities and other issues in the code. Noticing and fixing these warnings can help keep the code sure and functioning correctly. |
| 1. Architect and Design for Security Policies | Architect and design for security policies involve taking security into consideration during the early stages of the SDLC. This means addressing potential security vulnerabilities and developing to prevent them. |
| 1. Keep It Simple | Keeping it simple means developing code to be as simple as possible. This is helpful for reducing errors and vulnerabilities that can be created with more complex programs. |
| 1. Default Deny | Default deny means allowing user access to specific data and denying everything else by default. This can prevent users from accessing sensitive data that is meant to be secure. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege means the user should only have access to necessary information. Only giving the user access to what is needed reduces the risk of unauthorized actions. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data sent to other systems makes sure the data is safe and sent correctly. This involves cleaning the data by removing unwanted characters and formatting it correctly. |
| 1. Practice Defense in Depth | Defense in depth means implementing multiple layers of security into the program. Each layer strengthens security as there is more for an attacker to get through. |
| 1. Use Effective Quality Assurance Techniques | Using effective quality assurance techniques involves regularly testing the program for vulnerabilities. It is important to discover potential vulnerabilities early in the SDLC for better efficiency. Regular security testing can help discover security risks so they can be fixed right away. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard means following security guidelines while developing code. It can help a development team ensure they are working toward the same security goals as they have a guideline to follow. |

### 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** | **Converting a pointer to integer or integer to pointer** |
| --- | --- | --- |
| **Data Type** | INT-036-C | Converting pointer to integers or vice versa incorrectly can lead to incorrect results and undefined behavior. |

| **Noncompliant Code** |
| --- |
| Converting a 64-bit pointer cannot be represented in the 32-bit integer type. |
| **void** f(**void**) {  **char** \*ptr;  /\* ... \*/  unsigned **int** number = (unsigned **int**)ptr;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Any pointer to void can be converted to uintptr\_t and back without changing value. |
| #include <stdint.h>    **void** f(**void**) {  **char** \*ptr;  /\* ... \*/  **uintptr\_t** number = (**uintptr\_t**)ptr;  /\* ... \*/  } |

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

| **Principles(s):** Heed compiler warnings. The compiler will generate a warning in this scenario because of the incorrect integer type. It will notify the problem and may also generate a solution. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **pointer-integral-cast**  **pointer-integral-cast-implicit**  **function-pointer-integer-cast**  **function-pointer-integer-cast-implicit** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-INT36** | Fully implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.CAST.PC.CONST2PTR** **LANG.CAST.PC.INT** | Conversion: integer constant to pointer Conversion: pointer/integer |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-INT36-b** | A conversion should not be performed between a pointer to object type and an integer type other than 'uintptr\_t' or 'intptr\_t' |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that division and remainder operations do not result in divide-by-zero errors** |
| --- | --- | --- |
| **Data Value** | INT-033-C | Divide-by-zero errors can lead to unexpected behavior and integer overflow. |

| **Noncompliant Code** |
| --- |
| The division of s\_a and s\_b results in a divide-by-zero error. |
| #include <limits.h>    **void** func(**signed** **long** s\_a, **signed** **long** s\_b) {  **signed** **long** result;  **if** ((s\_a == LONG\_MIN) && (s\_b == -1)) {  /\* Handle error \*/  } **else** {  result = s\_a / s\_b;  }  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| By testing the division operation there will be no divide-by-zero errors. |
| #include <limits.h>    **void** func(**signed** **long** s\_a, **signed** **long** s\_b) {  **signed** **long** result;  **if** ((s\_b == 0) || ((s\_a == LONG\_MIN) && (s\_b == -1))) {  /\* Handle error \*/  } **else** {  result = s\_a / s\_b;  }  /\* ... \*/  } |

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

| **Principles(s):** Use effective quality assurance techniques. Using a unit test on this function will help determine that there is a divide by zero error. If left unnoticed the program could experience unexpected behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | **P6** | **L2** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **int-division-by-zero**  **int-modulo-by-zero** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.ARITH.DIVZERO** **LANG.ARITH.FDIVZERO** | Division by zero Float Division By Zero |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **43 D, 127 D, 248 S, 629 S, 80 X** | Partially implemented |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **DIVIDE\_BY\_ZERO** | Fully implemented |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do not attempt to modify string literals.** |
| --- | --- | --- |
| **String Correctness** | STR-030-C | Modifying string literals often results in an access violation because they are stored in read-only memory. |

| **Noncompliant Code** |
| --- |
| Initializing the char pointer str to the address of the string literal can cause unexpected behavior. |
| **char** \*str = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| Creating a copy of the string literal in the array str allows str to be modified. |
| **char** str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):** Adopt a secure coding standard. If a string literal needs to be modified, it is important to have a copy. Following this coding standard will prevent unwanted behavior when modifying strings. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **string-literal-modfication** **write-to-string-literal** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-STR30** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **157 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-STR30-a** **CERT\_C-STR30-b** | A string literal shall not be modified Do not modify string literals |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Sanitize data passed to complex subsystems** |
| --- | --- | --- |
| **SQL Injection** | STR-002-C | String data may contain special characters that can lead to malicious behavior. |

| **Noncompliant Code** |
| --- |
| Users can enter a string for email that will allow access. |
| **sprintf**(buffer, "/bin/mail %s < /tmp/email", addr);  **system**(buffer); |

| **Compliant Code** |
| --- |
| Acceptable characters are set so input can be rejected and cleaned. |
| **static** **char** ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  "1234567890\_-.@";  **char** user\_data[] = "Bad char 1:} Bad char 2:{";  **char** \*cp = user\_data; /\* Cursor into string \*/  **const** **char** \*end = user\_data + **strlen**( user\_data);  **for** (cp += **strspn**(cp, ok\_chars); cp != end; cp += **strspn**(cp, ok\_chars)) {  \*cp = '\_';  } |

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

| **Principles(s):** Sanitize data sent to other systems. Users should not be able to enter malicious characters to gain access. Data should be cleaned and checked for the correct format to be used. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **IO.INJ.COMMAND** **IO.INJ.FMT** **IO.INJ.LDAP** **IO.INJ.LIB** **IO.INJ.SQL** **IO.UT.LIB** **IO.UT.PROC** | Command injection Format string injection LDAP injection Library injection SQL injection Untrusted Library Load Untrusted Process Creation |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | **TAINTED\_STRING** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **108 D, 109 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-STR02-a** **CERT\_C-STR02-b** **CERT\_C-STR02-c** | Protect against command injection Protect against file name injection Protect against SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Properly deallocate dynamically allocated resources** |
| --- | --- | --- |
| **Memory Protection** | MEM-051-CPP | Deallocating resources that are not allocated dynamically can result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| Undefined behavior occurs because it is attempting to free memory that has not been returned. |
| #include <iostream>    **struct** S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    **void** f() {  alignas(**struct** S) **char** space[**sizeof**(**struct** S)];  S \*s1 = **new** (&space) S;    // ...    **delete** s1;  } |

| **Compliant Code** |
| --- |
| The delete s1; line is removed and replaced with s1->S();. It is now calling memory returned by new(). |
| #include <iostream>    **struct** S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    **void** f() {  alignas(**struct** S) **char** space[**sizeof**(**struct** S)];  S \*s1 = **new** (&space) S;    // ...    s1->~S();  } |

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

| **Principles(s):** Adopt a secure coding standard. Following best coding practices for deallocation of resources will keep the program secure and performing well. Improper memory use can lead to unexpected behavior and memory leaks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks -Wmismatched-new-delete clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **ALLOC.FNH** **ALLOC.DF** **ALLOC.TM** **ALLOC.LEAK** | Free non-heap variable Double free Type mismatch Leak |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MEM51-a** **CERT\_CPP-MEM51-b** **CERT\_CPP-MEM51-c** **CERT\_CPP-MEM51-d** | Use the same form in corresponding calls to new/malloc and delete/free Always provide empty brackets ([]) for delete when deallocating arrays Both copy constructor and copy assignment operator should be declared for classes with a nontrivial destructor Properly deallocate dynamically allocated resources |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Assertions** | DCL-003-C | Static assertions can be used to find incorrect assumptions during compiling. |

| **Noncompliant Code** |
| --- |
| An assertion is used on a memory-map structure that is needed 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** |
| --- |
| A preprocessor conditional statement can be used for an assertion of constant expressions. |
| **struct** timer {  unsigned **char** MODE;  unsigned **int** DATA;  unsigned **int** COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** Use effective quality assurance techniques. Correct code structure can ensure the program functions as intended. Using static assertions can be helpful to find incorrect assumptions that could be harmful to the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | ERR-051-CPP | All exceptions thrown must be caught by a matching exception handler. Not doing so can cause the program to terminate or keep the exception uncaught. |

| **Noncompliant Code** |
| --- |
| There is no matching exception handler to catch exceptions from throwing\_func(). |
| **void** throwing\_func() noexcept(**false**);    **void** f() {  throwing\_func();  }    **int** main() {  f();  } |

| **Compliant Code** |
| --- |
| The main() has exception handlers to catch all exceptions thrown. |
| **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): Heed compiler warnings. The compiler will give a warning when there is an exception without an exception handler. It is important to notice this error so all exceptions can be caught.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all** **early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Declare objects with appropriate storage durations** |
| --- | --- | --- |
| Declaration | DCL-030-C | Objects have storage durations and attempting to access them outside of their duration can lead to unexpected behavior. |

| **Noncompliant Code** |
| --- |
| C\_str is out of its scope when assigning it to the variable P. |
| #include <stdio.h>    **const** **char** \*p;  **void** dont\_do\_this(**void**) {  **const** **char** c\_str[] = "This will change";  p = c\_str; /\* Dangerous \*/  }    **void** innocuous(**void**) {  **printf**("%s\n", p);  }    **int** main(**void**) {  dont\_do\_this();  innocuous();  **return** 0;  } |

| **Compliant Code** |
| --- |
| The variables c\_str and p are declared with the same storage duration and p can now be assigned to c\_str. |
| **void** this\_is\_OK(**void**) {  **const** **char** c\_str[] = "Everything OK";  **const** **char** \*p = c\_str;  /\* ... \*/  }  /\* p is inaccessible outside the scope of string c\_str \*/ |

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

| **Principles(s):** Keep it simple. When declaring objects, it is important to declare them correctly. It is also important that these objects are accessed appropriately. Improper declaration and accessing can lead to unexpected behavior in the program. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | **P6** | **L2** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **pointered-deallocation**  **return-reference-local** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL30** | Fully implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.STRUCT.RPL** | Returns pointer to local |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **RETURN\_LOCAL** | Finds many instances where a function will return a pointer to a local stack variable. Coverity Prevent cannot discover all violations of this rule, so further verification is necessary |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Guarantee that library functions do not form invalid pointers** |
| --- | --- | --- |
| Arrays | ARR-038-C | Invalid pointers can be caused by using an incorrect element count and lead to pointers that do not point or unexpected behavior. |

| **Noncompliant Code** |
| --- |
| An incorrect element count is used in call to wmemcpy. |
| #include <string.h>  #include <wchar.h>    **static** **const** **char** str[] = "Hello world";  **static** **const** wchar\_t w\_str[] = L"Hello world";  **void** func(**void**) {  **char** buffer[32];  wchar\_t w\_buffer[32];  memcpy(buffer, str, sizeof(str)); /\* Compliant \*/  wmemcpy(w\_buffer, w\_str, sizeof(w\_str)); /\* Noncompliant \*/  } |

| **Compliant Code** |
| --- |
| The correct element count is used when calling functions. |
| #include <string.h>  #include <wchar.h>    **static** **const** **char** str[] = "Hello world";  **static** **const** wchar\_t w\_str[] = L"Hello world";  **void** func(**void**) {  **char** buffer[32];  wchar\_t w\_buffer[32];  memcpy(buffer, str, strlen(str) + 1);  wmemcpy(w\_buffer, w\_str, wcslen(w\_str) + 1);  } |

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

| **Principles(s):** Keep it simple. It is important to be mindful of element count when calling functions. Element count should be linked to the function called to prevent unexpected behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **array\_out\_of\_bounds** | Supported  Astrée reports all out-of-bound accesses within library analysis stubs. The user may provide additional stubs for arbitrary (library) functions. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.MEM.BO** **LANG.MEM.BU** **BADFUNC.BO.\*** | Buffer overrun Buffer underrun A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152309) | 9.7.1 | **64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X** | Partially Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-ARR38-a** **CERT\_C-ARR38-b** **CERT\_C-ARR38-c** **CERT\_C-ARR38-d** | Avoid overflow when reading from a buffer Avoid overflow when writing to a buffer Avoid buffer overflow due to defining incorrect format limits Avoid overflow due to reading a not zero terminated string |

#### Coding Standard 10

| **Coding Standard** | **Label** | **All exit handlers must return normally** |
| --- | --- | --- |
| Exit | ENV-032-C | Exit handlers must terminate by returning. If an exit handler has nowhere to return to or returns to the wrong place, unexpected behavior can occur. |

| **Noncompliant Code** |
| --- |
| Exit() can be called more than once and the program can be unpredictable. |
| #include <stdlib.h>    **void** exit1(**void**) {  /\* ... Cleanup code ... \*/  **return**;  }    **void** exit2(**void**) {  **extern** **int** some\_condition;  **if** (some\_condition) {  /\* ... More cleanup code ... \*/  **exit**(0);  }  **return**;  }    **int** main(**void**) {  **if** (**atexit**(exit1) != 0) {  /\* Handle error \*/  }  **if** (**atexit**(exit2) != 0) {  /\* Handle error \*/  }  /\* ... Program code ... \*/  **return** 0;  } |

| **Compliant Code** |
| --- |
| Using the exit handler atexit() will make the function exit by returning. |
| #include <stdlib.h>    **void** exit1(**void**) {  /\* ... Cleanup code ... \*/  **return**;  }    **void** exit2(**void**) {  **extern** **int** some\_condition;  **if** (some\_condition) {  /\* ... More cleanup code ... \*/  }  **return**;  }    **int** main(**void**) {  **if** (**atexit**(exit1) != 0) {  /\* Handle error \*/  }  **if** (**atexit**(exit2) != 0) {  /\* Handle error \*/  }  /\* ... Program code ... \*/  **return** 0;  } |

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

| **Principles(s):** Adopt a secure coding standard. Best practices should be used for the program to return to the right place. Incorrect use of exit and return can lead the program behaving unexpectedly. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Medium | **P12** | **L1** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **user\_defined**  **bad-function**  **bad-function-use** | Soundly supported |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **BADFUNC.ABORT** **BADFUNC.EXIT** **BADFUNC.LONGJMP** | Use of abort Use of exit Use of longjmp |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **122 S** **7 S** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-ENV32-a** | Properly define exit handlers |

### 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.

The existing DevOps structure is a great starting point for automation. During the pre-production stage automation tools can be used to conduct static code analysis, unit testing, and vulnerability testing. Static code analysis and unit testing should be done often during development. Vulnerability testing can be done during development but also should be done in the production stage. Additional automation can be added in the production stage like threat alerts, continuous integration, and accounting. Tools that monitor attempted attacks can notify the company as soon as they happen. Continuous integration tools can be used to instantly update the system when development changes have been made. Accounting can be done with tools that record information regarding changes that have been made to the system. All of these tools can be added to the existing DevOps structure to support the development and longevity of the program.

### 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 |
| INT-036-C | Low | Probable | High | P2 | L3 |
| INT-033-C | Low | Likely | Medium | **P6** | **L2** |
| STR-030-C | Low | Likely | Low | **P9** | **L2** |
| STR-002-C | High | Likely | Medium | **P18** | **L1** |
| MEM-051-CPP | High | Likely | Medium | **P18** | **L1** |
| DCL-003-C | Low | Unlikely | High | **P1** | **L3** |
| ERR-051-CPP | Low | Probable | Medium | **P4** | **L3** |
| DCL-030-C | High | Probable | High | **P6** | **L2** |
| ARR-038-C | High | Likely | Medium | **P18** | **L1** |
| ENV-032-C | Medium | Likely | Medium | **P12** | **L1** |

### 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 protect data that is being stored and not in use. The data is encrypted and is not readable until it is decrypted with a key. It is used to prevent attackers from accessing sensitive information. |
| Encryption in flight | Encryption in flight is used to protect data when it is being transferred from one place to another. It uses encryption methods to keep the data safe if it is interception during the transfer. The data stays encrypted until it needs to be accessed as intended. |
| Encryption in use | Encryption in use is used to protect data while it is being used. It ensures that the data is encrypted at all times until completely necessary. This prevents unwanted access to the data while it is being used. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is used to verify users and make sure they are who they say they are. To do this, usernames and passwords are used along with more secure methods like multi-factor verification. This prevents attackers from logging into users' profiles and accessing personal information. |
| Authorization | Authorization grants access to users based on who they are and what they are looking to access in the system. Full access can be given to administrators when changes need to be made. The default deny principle should be applied to give users access to only what they need to keep the program secure. |
| Accounting | Accounting monitors changes in the system and records who made the changes and information regarding the changes. It is used to track activity in the system. It can also be used as proof when an attack may occur and find who, when, and what was done. |

**\***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 |  |
| 2.0 | 11/18/2024 | Module Three Milestone | Luke Kundinger |  |
| 3.0 | 12/12/2024 | Project One | Luke Kundinger |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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