**Green Pace Developer: Security Policy Guide**



# 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 | This core security principle validates data being passed into the program. Therefore, verifying external data to ensure it isn’t corrupt or would cause any issues to the system(s). |
| 1. Heed Compiler Warnings | This requires following along with warning messages brought up in your code. This would help in preventing issues that may allow malicious code into your system. |
| 1. Architect and Design for Security Policies | This is what covers security policies that an organization should follow. This is important in standardizing the security measures taken to ensure a more secure system. |
| 1. Keep It Simple | Your code should be kept short and simple to lessen the chance of errors. This also saves the company money as the code won’t require as much time to update. |
| 1. Default Deny | Deny access permissions to all except those who are allowed to access the data. This is until a user shows that they have the required security keys to access the data. |
| 1. Adhere to the Principle of Least Privilege | This is in regard to any user, program, or process having the bare minimum privileges to the system. This makes it so that there is less of a chance of the system being used maliciously, with less people having high-level access. |
| 1. Sanitize Data Sent to Other Systems | This involves removing any confidential information before being sent to another system. |
| 1. Practice Defense in Depth | Creating a large layer of protective measures to protect important data. |
| 1. Use Effective Quality Assurance Techniques | Ensure that you are testing your code and that hacking attempts can be thwarted. |
| 1. Adopt a Secure Coding Standard | By obtaining secure coding standard’s, we are bettering our code and by cleaning it up and protecting our systems against hacking attempts. |

### 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** | INT50-CPP | Do not cast to an out-of-range enumeration value. |

| **Noncompliant Code** |
| --- |
| This piece of code tries to check if a value is in the range of acceptable numerable values. This is noncompliant, mainly due to it casting the enumeration type beforehand, making it unable to represent the specified integer. The value’s valid range is represented by “EnumType” being [0..3]. In this case, if a value were passed to f(), then EnumType would result in an undefined value in the conditional statement’s outcome. |
| **enum** EnumType {    First,    Second,    Third  };    **void** f(**int** intVar) {    EnumType enumVar = **static\_cast**<EnumType>(intVar);    **if** (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| This code verifies that the value can be represented by the enumeration type before performing the conversation to guarantee it does not result in an undefined value. It restricts the converted integer to another with a specific enumerator value. |
| **enum** EnumType {    First,    Second,    Third  };    **void** f(**int** intVar) {  **if** (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = **static\_cast**<EnumType>(intVar);  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 | N/A |
| CodeSonar | 6.2p0 | LANG.CAST.COERCE LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2022.1 | C++3013 | N/A |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | CTR50-CPP | Guarantee that container indices and iterators are within the valid range. |

| **Noncompliant Code** |
| --- |
| The function “insert\_in\_table()” has two int parameters, being “pos” and “value”. Both of these being easily influenced by data from untrusted sources. This function checks the upper bounds of the array, specified by “tableSize” yet not checking the lower bounds. “Pos” is a signed int, which means that this parameter may assume a negative integer, which would result in an error, which is referenced by “table”. |
| #include <cstddef>    **void** insert\_in\_table(**int** \*table, std::**size\_t** tableSize, **int** pos, **int** value) {  **if** (pos >= tableSize) {      // Handle error  **return**;    }    table[pos] = value;  } |

| **Compliant Code** |
| --- |
| This is one of the easiest ways to fix this noncompliant issue. Here, the parameter “pos” is declared as “size\_t”, disallowing negative arguments to pass through the function. |
| #include <cstddef>    **void** insert\_in\_table(**int** \*table, std::**size\_t** tableSize, std::**size\_t** pos, **int** value) {  **if** (pos >= tableSize) {      // Handle error  **return**;    }    table[pos] = value;  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TO  LANG.MEM.TU  LANG.MEM.TBA  LANG.STRUCT.PBB  LANG.STRUCT.PPE | Buffer overrun  Buffer underrun  Type overrun  Type underrun  Tainted buffer access  Pointer before beginning of object  Pointer past end of object |
| LDRA tool suite | 9.7.1 | 45 D, 47 S, 476 S, 489 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X | Partially Implemented |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-CTR50-a | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2021b | CERT C++: CTR50-CPP | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR50-CPP | Guarantee that storage for strings has sufficient space for character data and the null terminator. |

| **Noncompliant Code** |
| --- |
| This input is unbounded, which may result in a buffer overflow. In the second piece of code, the first read will not result in an overflow, could allow a trunicated string to fill “bufOne”. Then, the second read could result in an overflow regarding “bufTwo”. |
| #include <iostream>    **void** f() {  **char** buf[12];    std::cin >> buf;  }  #include <iostream>    **void** f() {  **char** bufOne[12];  **char** bufTwo[12];    std::cin.width(12);    std::cin >> bufOne;    std::cin >> bufTwo;  } |

| **Compliant Code** |
| --- |
| A compliant solution is making sure that the data isn’t trunicated. Additionally, It is important to avoid overflows by using “std::string” in place of a bounded array. |
| #include <iostream>  #include <string>    **void** f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Klocwork | 2022.1 | C++2835, C++2836, C++2839, C++5216 | N/A |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially Implemented |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the 'char' buffer to store input from 'std::cin' |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | MEM56-CPP | Do not store an already-owned pointer value in an unrelated smart pointer. |

| **Noncompliant Code** |
| --- |
| Two unrelated smart pointers are created starting from the same pointer. When “p2” is broken, it deles the pointer’s value in which it manipulates. Then, when “p1” is eliminated, it removes the same pointer. This adds more to its vulnerability. |
| #include <memory>    **void** f() {  **int** \*i = **new** **int**;    std::shared\_ptr<**int**> p1(i);    std::shared\_ptr<**int**> p2(i);  } |

| **Compliant Code** |
| --- |
| The “std::shared\_ptr” objects relate to each other with copy construction. When “p2” is eliminated, the use court for the shared pointer is decremented and isn’t zero. Then, when “p1” is eliminated, the use court for the pointer is decrementing onto zero, therefore getting rid of the managed pointer. Finally, “std::make\_shared()” is called, in place of allocating a raw pointer and placing it in its own local variable. |
| #include <memory>    **void** f() {    std::shared\_ptr<**int**> p1 = std::make\_shared<**int**>();    std::shared\_ptr<**int**> p2(p1);  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Dangling\_pointer\_use | N/A |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM56 | N/A |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-MEM56-a | Do not store an already-owned pointer value in an unrelated smart pointer |
| Polyspace Bug Finder | R2021b | CERT C++: MEM56-CPP | Checks for use of already-owned pointers (rule fully covered) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM52-CPP | Detect and handle memory allocation errors. |

| **Noncompliant Code** |
| --- |
| This is non-compliant as the results of the allocation are not verified when an array of int is developed using “::operator new[] (std:size\_t)”. Because the function is marked as “noexcept”, the caller assumes it does not throw any exceptions. |
| #include <cstring>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** **int**[size];    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

| **Compliant Code** |
| --- |
| Using “std::nothrow”, the new operator is able to return a null pointer or a pointer to the allocated space. Then, test the pointed to ensure it is not a “nullptr” before referencing the timer. |
| #include <cstring>  #include <new>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** (std::**nothrow**) **int**[size];  **if** (!copy) {      // Handle error  **return**;    }    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool suite | 9.7.1 | 45 D | Partially Implemented |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| Polyspace Bug Finder | R2021b | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |
| PRQA QA-C++ | 4.4 | 3225, 3226, 3227, 3228, 3229, 4632 | N/A |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | DCL03-C | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| This uses an “assert()” macro to declare a property concerning a memory-mapped structure that is highly important for the code to work effectively. This should be placed within a function, then executed. |
| #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 condition statement is an effective method. By using “#error”, clear diagnostic messages are able to be presented. Additionally, no runtime penalty occurs due to this method evaluating assertions at compile time. |
| **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):** Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Mist-static-assert | Checked by clang-tidy |
| CodeSonar | 6.2p0 | 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 | Honor exception specifications  Skip to end of metadata |

| **Noncompliant Code** |
| --- |
| The function here is declared as nonthrowing. However, “std::vector::resize()” allows for exceptions to be thrown when the requested memory is unable to be allocated. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) noexcept(**true**) {    v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| Here, the “noexcept-specification” is removed thus showing that the function allows all exceptions. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) {    v.resize(s); // May throw, but that is okay  } |

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

| **Principles(s):** Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | bad-function (AUTOSAR.26.5.1A) | Fully Checked |
| Clang | 4.0 | Cert-msc50-cpp | Checked by clang-tidy |
| CodeSonar | 6.2p0 | BADFUNC.RANDOM.RAND | Use of rand |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-MSC50-a | Do not use the rand() function for generating pseudorandom numbers |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] Object Oriented Programming | OOP50-CPP | Do not invoke virtual functions from constructors or destructors |

| **Noncompliant Code** |
| --- |
| The base class tries to stop and release an object’s resources using calls to functions from the constructor plus the destructor. The “B::B()” constructor calls “B::seize()” rather than “D::seize()”. Similarly, “B::~B()” calls “B::release()” instead of “D::release()”. |
| **struct** B {    B() { seize(); }  **virtual** ~B() { release(); }    **protected**:  **virtual** **void** seize();  **virtual** **void** release();  };    **struct** D : B {  **virtual** ~D() = **default**;    **protected**:  **void** seize() override {      B::seize();      // Get derived resources...    }    **void** release() override {      // Release derived resources...      B::release();    }  }; |

| **Compliant Code** |
| --- |
| The constructors and destructors call a private member function, “mine”, instead of a virtual function. Every class is responsible for getting and releasing its own resources. |
| **class** B {  **void** seize\_mine();  **void** release\_mine();    **public**:    B() { seize\_mine(); }  **virtual** ~B() { release\_mine(); }    **protected**:  **virtual** **void** seize() { seize\_mine(); }  **virtual** **void** release() { release\_mine(); }  };    **class** D : **public** B {  **void** seize\_mine();  **void** release\_mine();    **public**:    D() { seize\_mine(); }  **virtual** ~D() { release\_mine(); }    **protected**:  **void** seize() override {      B::seize();      seize\_mine();    }    **void** release() override {      release\_mine();      B::release();    }  }; |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2022.1 | C++4260, C++4261, C++4273, C++4274, C++4275, C++4276, C++4277, C++4278, C++4279, C++4280, C++4281, C++4282 | N/A |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-OOP50-a  CERT\_CPP-OOP50-b  CERT\_CPP-OOP50-c  CERT\_CPP-OOP50-d | Avoid calling virtual functions from constructors  Avoid calling virtual functions from destructors  Do not invoke class's virtual functions from any of its constructors  Do not invoke class's virtual functions from its destructor |
| Polyspace Bug Finder | R2021b | CERT C++: OOP50-CPP | Checks for virtual function call from constructors and destructors (rule fully covered) |
| RuleChecker | 20.10 | Virtual-call-in-constructor | Fully Checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] Input Output (FIO) | FIO51-CPP | Close files then they are no longer needed. |

| **Noncompliant Code** |
| --- |
| The “std::fstream” object file is constructed, with the contructor calling “std::basic\_filebuf<T>::open()”. The default “std::terminate\_handler” being called by “std::terminate()” is “std::abort()” which fails to call on destructors. Therefore, the underlying object is not properly closed. |
| #include <exception>  #include <fstream>  #include <string>    **void** f(**const** std::string &fileName) {    std::fstream file(fileName);  **if** (!file.is\_open()) {      // Handle error  **return**;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| “std::fstream::close()” is called before “std::terminate()” which makes sure that the file’s resources are properly closed. |
| #include <exception>  #include <fstream>  #include <string>    **void** f(**const** std::string &fileName) {    std::fstream file(fileName);  **if** (!file.is\_open()) {      // Handle error  **return**;    }    // ...    file.close();  **if** (file.fail()) {      // Handle error    }    std::terminate();  } |

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

| **Principles(s):** Adopt a secure coding standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | ALLOC.LEAK | Leak |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Polyspace Bug Finder | R2021b | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered). |
| Klocwork | 2022.1 | RH.LEAK | N/A |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] Expressions | EXP53-CPP | Do not read uninitialized memory. |

| **Noncompliant Code** |
| --- |
| Here, we see that an uninitialized local variable is evaluated as part of an expression to print its value, ending in undesirable results. |
| #include <iostream>    **void** f() {  **int** i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| Instead, we can initialize the object before printing its value. |
| #include <iostream>    **void** f() {  **int** i = 0;    std::cout << i;  } |

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

| **Principles(s):** Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2021b | CERT C++:EXP53-CPP | Checks for: > Non-initialized variable > Non-initialized pointer Rule partially covered. |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-EXP53-a | Avoid use before initialization. |
| LDRA tool suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Avoid use before initialization. |
| CodeSonar | 6.2p0 | LANG.STRUCT.RPL LANG.MEM.UVAR | Return pointer to local uninitialized variable. |

### 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 |
| --- | --- | --- | --- | --- | --- |
| INT50-CPP | Medium | Unlikely | Medium | P4 | L3 |
| CTR50-CPP | High | Likely | High | P9 | L2 |
| STR50-CPP | High | Likely | Medium | P18 | L1 |
| MEM56-CPP | High | Likely | Medium | P18 | L1 |
| MEM52-CPP | High | Likely | Medium | P18 | L1 |
| DCL03-C | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Medium | Unlikely | Low | P6 | L2 |
| OOP50-CPP | Low | Unlikely | Medium | P2 | L3 |
| FIO51-CPP | Medium | Unlikely | Medium | P4 | L3 |
| EXP53-CPP | High | Probable | 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 in rest | This type of encryption refers to when data is encrypted while it is stored. This encryption also prevents this data from being read if the proper key is not present. This is a great way to have additional layers of protection. |
| Encryption at flight | This ensures data is encrypted as it is being sent. However, the data is not as secured while it is stored or used. The data becomes encrypted while being sent to another storage location. This is a great way to protect data if it is intercepted with malicious intent. Particularly, this is a useful encryption method for those who work from home. This is because data can be secured as it is sent from the company network to the workers. |
| Encryption in use | This method allows data to be encrypted while it is being used, where it can be accessed depending on the security level. This is great for company databases with creating some layers of protection, separating user and employee activity. Only higher level employees can access the data if needed. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is important in that it allows users to access the system after they have proven they have access through login systems. |
| Authorization | This identifies he level of access given to a user. In this case, the administrator would have access to most items such as databases, secure files, customer records, etc. The administrator is also allowed to make changes to the system by adding and deleting items, to include user accounts. Lower-end users would have much less authorization in the system. |
| Accounting | Accounting allows for the keeping of records for when data has been changed in a system, and which user has made those changes. This allows for a record of certain changes, so if someone makes a mistake, or does something malicious, it can be addressed and who did it can be known. |

**\***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**   
  **> Adhere to the Principle of Least Privilege** – OS logs are the lowest level, at the end user.   
  **> Heed Compiler Warnings** – Operating system logs could provide insight to issues that lay within the code. This allows developers to find issues and ensure the system is working properly.  
  **> Use Effective Quality Assurance Techniques –** Ensure code in the Operating Systema re secure to protect against hacking attempts.
* **Firewall logs**   
  **> Validate Input Data** – Validate any information sent to the system and ensure that there are no security risks.  
  > Default Deny -   
  **> Adopt a Secure Coding Standard** – Allowing the developers to have certain standards when protecting the system.
* **Anti-malware logs**  
  **> Validate Input Data** – Validate any information sent to the system and ensure that there are no security risks.  
  **> Default Deny** – Authorization and authentication verification are required first before the user account is granted access to the system.   
  **> Practice Defense in Depth** – Multiple layers of protection, such as here, assist in protecting the system from malicious attacks.

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 | 04/01/2022 | First Draft | Brandon Lombard |  |
| 3.0 | 04/08/2022 | Final Draft | Brandon Lombard |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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