**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 | This is used to validate the data that is being inputted into the program. You always want to validate any data that is being brought into the program from an outside source. This will ensure that you aren’t bringing any data in that is corrupt or could possibly harm the program or operating system. |
| 1. Heed Compiler Warnings | You want to ensure that you follow any warning messages that you get when trying to compile your code. By doing this we will be able to help prevent any hackers from entering the program through an open door. Once you get these warnings you want to change your code to resolve them. |
| 1. Architect and Design for Security Policies | These will help you create and design software that is able to enforce any security policies and implement them. |
| 1. Keep It Simple | By following this you will be able to fix any errors or warnings that come about when debugging your program. If you created code that was too complex it would make it difficult to find any errors within it. |
| 1. Default Deny | This is used to allow authorized people access to certain data and denying access to the ones that aren’t authorized. Once someone that is unauthorized shows that they have the correct security keys to access the data is when they will be able to do so. |
| 1. Adhere to the Principle of Least Privilege | This is here to ensure that everyone has the correct amount of privilege needed and is not given too much or too little. If there is a time when someone needs their privilege elevated, they can request this, and it should only be authorized for a set amount of time. |
| 1. Sanitize Data Sent to Other Systems | This is in place to ensure that any data that is being passed onto a complex subsystem such as a command shell is sanitized before doing so. |
| 1. Practice Defense in Depth | Using multi-layer defenses will help the system stay protected and less vulnerable to attacks. By having multiple layers this means that even if one layer gets hacked or goes down there are still other layers to keep the system from being compromised. |
| 1. Use Effective Quality Assurance Techniques | By using good quality techniques, you will be able to identify and eliminate any vulnerabilities efficiently. Some examples of this would be penetration testing, source code audits, or fuzz testing. |
| 1. Adopt a Secure Coding Standard | You want to adopt a secure code standard when choosing your choice of language and platform. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| This checks whether a given value is in the range of the accepted enumeration values. This code may not be able to represent the given integer value when casting it. |
| **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 checks to see if the integer value is being represented by the enumeration type before converting it. This ensures that they won’t end up as an unspecified value. Once complete it will have converted the value into an enumerator type. |
| **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):** An overflow of the buffer that can be created by any value. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Cast-integer-to-enum | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 | N/A |
| CodeSonar | 8.1p0 | LANG.CAST.COERCE | Coercion Alters Value |
| Helix QAC | 2024.2 | C++3013 | N/A |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Use of valid references, pointers, and iterators to reference elements of a container. |

| **Noncompliant Code** |
| --- |
| This is noncompliant because the POS is determined as being invalid after the first call to insert() and the loop iterations don’t have any defined behavior. |
| #include <deque>    **void** f(**const** **double** \*items, std::**size\_t** count) {    std::deque<**double**> d;    auto pos = d.begin();  **for** (std::**size\_t** i = 0; i < count; ++i, ++pos) {      d.insert(pos, items[i] + 41.0);    }  } |

| **Compliant Code** |
| --- |
| This is compliant because the POS has a valid iterator assigned to it which prevents the original behavior from being undefined. |
| #include <deque>    **void** f(**const** **double** \*items, std::**size\_t** count) {    std::deque<**double**> d;    auto pos = d.begin();  **for** (std::**size\_t** i = 0; i < count; ++i, ++pos) {      pos = d.insert(pos, items[i] + 41.0);    }  } |

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

| **Principles(s):** By using behaviors that are not identified it can cause the wrong pointers and references to be used. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Code Sonar | 8.1p0 | ALLOC.UAF | Use After Free |
| Helix QAC | 2024.2 | DF4746, DF4747, DF4748, DF4749 | N/A |
| Klocwork | 2024.2 | ITER.CONTAINER.MODIFIED | N/A |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-CTR51-a | Do not modify the container while iterating over it |

#### Coding Standard 3

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

| **Noncompliant Code** |
| --- |
| This is noncompliant because std::getenv() returns a null pointer failure. This is happening due to their being undefined behavior because the environment variable doesn’t exist. |
| #include <cstdlib>  #include <string>    **void** f() {    std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| [Compliant description] |
| #include <cstdlib>  #include <string>    **void** f() {  **const** **char** \*tmpPtrVal = std::**getenv**("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");  **if** (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):** By not defining a behavior you can cause null pointers to be unreferenced which can cause a program to terminate at any point. This can be used by hackers to execute their code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.MEM.NPD | Null Pointer Deference |
| Helix QAC | 2024.2 | DF4770, DF4771, DF4772, DF4773, DF4774 | N/A |
| Astree | 22.10 | assert\_failure | N/A |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Do not store already owned pointer value in an unrelated smart pointer. |

| **Noncompliant Code** |
| --- |
| This code is not compliant because two unrelated smart pointers are constructed from the same underlying pointer value. Because of this when the local P2 variable is destroyed the pointer value that it managed was also deleted. Then when the P1 variable is destroyed it deletes the same pointer value that was already deleted causing a double-free vulnerability. |
| #include <memory>    **void** f() {  **int** \*i = **new** **int**;    std::shared\_ptr<**int**> p1(i);    std::shared\_ptr<**int**> p2(i);  } |

| **Compliant Code** |
| --- |
| This code is compliant because the std:shared\_ptr objects are related by copy construction. This means that when the P2 variable is destroyed the pointer value that is shared is decremented and still nonzero. So, when the P1 variable is destroyed the use count for the pointer is decreased to zero which results in the pointer being destroyed. It also calls for std:make\_shared() instead of trying to allocate a raw pointer and then storing its value in a 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):** By passing a pointer value to a deallocation function that wasn’t obtained by a matching allocation function previously you can cause there to be undefined behavior which can cause exploitable vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Dangling\_pointer\_use | N/A |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MEM56-a | Do not store an already owned pointer value in an unrelated smart pointer. |
| PVS-Studio | 7.33 | V1006 | N/A |
| Polyspace Bug Finder | R2024a | 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** | [STD-005-CCP] | Properly deallocate dynamically allocated resources. |

| **Noncompliant Code** |
| --- |
| This code is noncompliant because when the variable is passed as the expression to the placement operator it passes the pointer to ::operator delete() which causes an undefined behavior due to ::operator delete() trying to delete memory that was returned originally by ::operator 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;      // ...    **delete** s1;  } |

| **Compliant Code** |
| --- |
| This code is compliant because when calling ::operator delete() is removed and it’s specifically calls s1’s destructor instead. |
| #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):** By passing a pointer value to a deallocation function that wasn’t obtained by a matching allocation function previously you can cause there to be undefined behavior which can cause exploitable vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks -Wmismatched-new-delete clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| Helix QAC | 2024.2 | **C++2110, C++2111, C++2112, C++2113, C++2118, C++3337, C++3339, C++4262, C++4263, C++4264** | N/A |
| LDRA tool suite | 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 Insure++ | N/A | N/A | Runtime Decision |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use 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** |
| --- |
| This code is compliant because it added a preprocessor conditional statement for assertions that only involves 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):** This is a good tool to diagnose and eliminating defects in the software. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 | N/A |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handles all exceptions thrown before main () begins executing. |

| **Noncompliant Code** |
| --- |
| This code is noncompliant because the constructor used for S may throw an exception that isn’t caught when the globalS is constructed during program setup. |
| **struct** S {    S() noexcept(**false**);  };    **static** S globalS; |

| **Compliant Code** |
| --- |
| This is compliant because it creates globalS into a local variable with static storage duration which allows any exception to be thrown during object construction to be caught. This also lets the constructor S be executed when the function globalS is called instead of at the start of the program. This doesn’t require any source code to be modified either. |
| **struct** S {    S() noexcept(**false**);  };    S &globalS() {  **try** {  **static** S s;  **return** s;    } **catch** (...) {      // Handle error, perhaps by logging it and gracefully terminating the application.    }    // Unreachable.  } |

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

| **Principles(s):** This can be a result in throwing an exception that can’t be caught. This can result in attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | potentially-throwing-static-initialization | Partially checked |
| Clang | 3.9 | Cert-err58-cpp | Checked by clang-tidy |
| Helix QAC | 2024.2 | |  |  | | --- | --- | | C++4634, C++4636, C++4637, C++4639 |  | | N/A |
| RuleChecker | 22.10 | Potentially-throwing-static-initialization | Partially Checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Object Oriented Programming** | [STD-008-CPP] | Do not invoke virtual functions from constructors or destructors. |

| **Noncompliant Code** |
| --- |
| This code is noncompliant because the base class is attempting to seize and release an objects resource through calls to virtual functions from the constructor and destructor. The B::B() constructor calls B::seize() when it should be calling D::sieze(). |
| **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** |
| --- |
| This is compliant because the constructors and destructors are calling a nonvirtual, private member function instead of the virtual function. By doing this it makes each class responsible for seizing and releasing their 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):** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | virtual-call-in-constructor invalid\_function\_pointer | Fully Checked |
| CodeSonar | 8.1p0 | **LANG.STRUCT.VCALL\_IN\_CTOR**  **LANG.STRUCT.VCALL\_IN\_DTOR** | Virtual Call in Constructor  Virtual Call in Destructor |
| LDRA tool suite | 9.7.1 | 467 S, 92 D | Fully Implemented |
| RuleChecker | 22.10 | Virtual-call-in constructor | Fully Checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Miscellaneous** | [STD-009-CPP] | Value-returning functions must return a value from all exit paths. |

| **Noncompliant Code** |
| --- |
| This code is noncompliant because there is no return for the input value for positive input. |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;    }  } |

| **Compliant Code** |
| --- |
| This code is compliant because all code paths have a return. |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;    }  **return** a;  } |

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

| **Principles(s):** Not being able to return a value from a code path in a value-returning function will result in undefined behavior. This can cause data integrity violations. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Return-implicit | Fully Checked |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this rule, such as function-try-blocks |
| CodeSonar | 8.1p0 | LANG.STRUCT.MRS LANG.STRUCT.NVNR | Missing return statement Non-void noreturn, |
| Klocwork | 2024.2 | **FUNCRET.GEN**  **FUNCRET.IMPLICIT** | N/A |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Input Output** | [STD-010-CPP] | Do not alternately input and output file from a file stream without an intervening flush or positioning call. |

| **Noncompliant Code** |
| --- |
| This is noncompliant because the code is appending data to a file and then is reading from the same file. |
| #include <stdio.h>    **enum** { BUFFERSIZE = 32 };    **extern** **void** initialize\_data(**char** \*data, **size\_t** size);    **void** func(**const** **char** \*file\_name) {  **char** data[BUFFERSIZE];  **char** append\_data[BUFFERSIZE];  **FILE** \*file;      file = **fopen**(file\_name, "a+");  **if** (file == NULL) {      /\* Handle error \*/    }      initialize\_data(append\_data, BUFFERSIZE);    **if** (**fwrite**(append\_data, 1, BUFFERSIZE, file) != BUFFERSIZE) {      /\* Handle error \*/    }  **if** (**fread**(data, 1, BUFFERSIZE, file) < BUFFERSIZE) {      /\* Handle there not being data \*/    }    **if** (**fclose**(file) == EOF) {      /\* Handle error \*/    }  } |

| **Compliant Code** |
| --- |
| This code is compliant because fseek() is called between the input and output which eliminates the undefined behavior. |
| #include <stdio.h>    **enum** { BUFFERSIZE = 32 };  **extern** **void** initialize\_data(**char** \*data, **size\_t** size);    **void** func(**const** **char** \*file\_name) {  **char** data[BUFFERSIZE];  **char** append\_data[BUFFERSIZE];  **FILE** \*file;      file = **fopen**(file\_name, "a+");  **if** (file == NULL) {      /\* Handle error \*/    }      initialize\_data(append\_data, BUFFERSIZE);  **if** (**fwrite**(append\_data, BUFFERSIZE, 1, file) != BUFFERSIZE) {      /\* Handle error \*/    }    **if** (**fseek**(file, 0L, SEEK\_SET) != 0) {      /\* Handle error \*/    }    **if** (**fread**(data, BUFFERSIZE, 1, file) != 0) {      /\* Handle there not being data \*/    }    **if** (**fclose**(file) == EOF) {      /\* Handle error \*/    }  } |

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

| **Principles(s):** If you try to input or output from a stream without having an intervening flush or a positioning call you will create undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | IO.IOWOP  IO.OIWOP | Input After Output Without Positioning  Output After Input Without Positioning |
| Helix QAC | 2024.2 | |  |  |  | | --- | --- | --- | |  | DF4711, DF4712, DF4713 |  | | N/A |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-FI050-a | |  | | --- | |  |   Do not alternately input and output from a stream without an intervening flush or positioning call |
| Polyspace Bug Finder | R2024a | CERT C++: FI050-CPP | Checks for alternating input and output from a stream without flush or positioning call. |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Medium | Unlikely | Medium | P4 | 3 |
| STD-002-CPP | High | Probable | High | P6 | 2 |
| STD-003-CPP | High | Likely | Medium | P18 | 1 |
| STD-004-CPP | High | Likely | Medium | P18 | 1 |
| STD-005-CPP | High | Likely | Medium | P18 | 1 |
| STD-006-CPP | Low | Unlikely | High | P1 | 3 |
| STD-007-CPP | Low | Likely | Low | P9 | 2 |
| STD-008-CPP | Low | Unlikely | Medium | P2 | 3 |
| STD-009-CPP | Medium | Likely | Medium | P8 | 2 |
| STD-010-CPP | Low | Likely | Medium | P6 | 2 |

### 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 | This is used to prevent hackers from being able to gain access to any data that isn’t encrypted. This is accomplished by making sure that the data on the disk is encrypted. By doing this the hacker must get passed the encryption to access the data that is on the hard drive after obtaining the hard and not having access to the encryption keys. |
| Encryption in flight | This happens when the data is trying to be encrypted and transmitted simultaneously. Some data is encrypted while being transmitted instead of being stored on a hard drive that is encrypted. This creates an extra layer of security to the data. |
| Encryption in use | This allows access to encrypted data that is either in rest or in motion. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is when you validate who someone is before granting them access to the system. Some examples of this would be having them login using certain credentials like a username or a password. |
| Authorization | Authorization refers to how much access certain users have within the system. Depending on the user depends on the permissions they have. For example, if you are an admin in the system you are going to have more permissions and access than a general system user. |
| Accounting | Accounting consists of keeping track of users and what kind of activity they are doing within their specified level of authorization within the system. This helps with keeping them accountable for their actions. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 09/25/2024 | Module 3 Assignment | Douglas Wright |  |
| 1.2 | 10/12/2024 | Project 1 | Douglas Wright |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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