**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 the data input into a system helps mitigate and prevent vulnerabilities caused by malicious inputs. Input validation helps ensure that only legit and expected data is accepted into the system, reducing the risk of SQL injections and other types of attacks. |
| 1. Heed Compiler Warnings | The compiler gives potential warning messages that flag possible vulnerabilities in the code during the development process. Developers need to understand and fix these warnings before code goes into production. |
| 1. Architect and Design for Security Policies | Create an architecture that considers potential security risks and design the system with security in mind. Have clear and understandable policies to aid the process. |
| 1. Keep It Simple | Simple code reduces complexity and increases understanding during the development process. Simple code often produces less vulnerabilities. |
| 1. Default Deny | Deny access to all resources by default and only grant access to those that are explicitly authorized. This reduces the risk of unauthorized users accessing the system and helps prevent security breaches. |
| 1. Adhere to the Principle of Least Privilege | When a new user is assigned to a role, they should receive bare minimum level of access that is required to do their tasks. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed into complex subsystems like the command shell and config files. This ensures it can’t be exploited by malicious hackers. |
| 1. Practice Defense in Depth | Implement multiple layers of security to protect against different types of attacks. Having multiple layers gives assurance that the system is still protected if one layer fails. |
| 1. Use Effective Quality Assurance Techniques | Effective quality assurance techniques such as code review and testing identify potentially hazardous code. Routine testing ensures that a vulnerability will reveal itself with time. |
| 1. Adopt a Secure Coding Standard | Set a coding standard, which is a set of guidelines and practices, that all developers should follow to ensure the safety of their code. This can include implementation and guidelines 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** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STR50-CPP | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| The input is unbounded. This can potentially lead to a buffer overflow |
| #include <iostream>    void f() {    char buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| Data is stored in a std::string instead 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):** 1 – Validate input data: Validating input data helps mitigate certain types of attacks like SQL Injection. No private data can be retrieved if the input data doesn’t carry over into bits that are storing sensitive information.  7 – Sanitize data sent to other systems: This principle banks off of principle 1, while validating the data is it a good idea to sanitize it as it goes into the system as well as another layer of protection against malicious code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stream-input-char-array** | Partially checked + soundly supported |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++5216**  **DF2835, DF2836, DF2839,** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | **NNTS.MIGHT** **NNTS.TAINTED** **NNTS.MUST** **SV.UNBOUND\_STRING\_INPUT.CIN** |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT50-CPP | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| This example attemps to check whether a given value is within range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given inter value. |
| 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 compliant solution checks that the value can be represented by the enumeration type before performing the conversion. |
| 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):** 2 – Heed Compiler Warnings: The IDE’s compiler will hault the code from compiling and point out where the error is in this example; trying to cast an out-of-range enumeration value. This prevents any bugs or corrupted information from appearing. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | Medium | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **cast-integer-to-enum** | |  |  | | --- | --- | |  | Partially checked | |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-INT50** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | |  |  | | --- | --- | |  | 8.1p0 | | **LANG.CAST.COERCE**  **LANG.CAST.VALUE** | Coercion Alters Value  Cast Alters Value |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++3013** |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR51-CPP | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| A std::string object is created from the results of a call to std::getenv(). Because std::getenv() return a null pointer on failure, the code can lead to undefined behavior. |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

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

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

| **Principles(s):** 2 – Heed Compiler Warnings: When the compiler throws an error it is best to remedy it as soon as possible as returning a null pointer on std::string can result in undefined behavior.  4 – Keep It Simple: Keeping calls simple will help reduce any odd errors the code may produce. It also makes it easier to maintain over the lifecycle of the software. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | |  |  | | --- | --- | |  | **assert\_failure** | |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | |  |  | | --- | --- | |  | 8.1p0 | | |  |  | | --- | --- | |  | **LANG.MEM.NPD** | | Null Pointer Dereference |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | |  |  | | --- | --- | |  | 2024.2 | | **DF4770, DF4771, DF4772, DF4773, DF4774** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.2 | **NPD.CHECK.CALL.MIGHT** **NPD.CHECK.CALL.MUST** **NPD.CHECK.MIGHT** **NPD.CHECK.MUST** **NPD.CONST.CALL** **NPD.CONST.DEREF** **NPD.FUNC.CALL.MIGHT** **NPD.FUNC.CALL.MUST** **NPD.FUNC.MIGHT** **NPD.FUNC.MUST** **NPD.GEN.CALL.MIGHT** **NPD.GEN.CALL.MUST** **NPD.GEN.MIGHT** **NPD.GEN.MUST** **RNPD.CALL** **RNPD.DEREF** |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STR30-C | Do not attempt to modify string literals |

| **Noncompliant Code** |
| --- |
| In this example, the char pointer str is initialized to the address of a string literal. Attempting to modify the string literal is undefined behavior. |
| char \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| As an array initializer, a string literal specifies the initial values of characters in an array as well as the size of the array. This code creates a copy of the string literal in the space allocated to the character array str. |
| char str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):** 3 – Architect and Design for Security Policies: This is a principle due to the attempt of modifying a string literal resulting in undefined behavior. Not knowing what the code will output presents a potential breach of code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Low | Low | Low | 4 |

**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 |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **PW** | |  |  | | --- | --- | |  | Deprecates conversion from a string literal to "char \*" | |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.2 | |  |  | | --- | --- | |  | **C0556, C0752, C0753, C0754**  **C++3063, C++3064, C++3605, C++3606, C++3607** | |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM31-C | Free dynamically allocated memory when no longer needed |

| **Noncompliant Code** |
| --- |
| The object allocated by the call to malloc() is not freed before the end of the lifetime of the last pointer text\_buffer referring to the object |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {    char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);    if (text\_buffer == NULL) {      return -1;    }    return 0;  } |

| **Compliant Code** |
| --- |
| In this compliant example, the pointer is deallocated with a call to free() |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {    char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);    if (text\_buffer == NULL) {      return -1;    }      free(text\_buffer);    return 0;  } |

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

| **Principles(s):** 9 – Use Effective Quality Assurance Techniques: Using quality techniques will mitigate memory allocations when dealing with the code. Forgetting to free memory can cause memory leaks, causing a program to crash. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MEM31** | Can detect dynamically allocated resources that are not freed |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **ALLOC.LEAK** | Leak |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 2.15 | **memleak** **leakReturnValNotUsed** **leakUnsafeArgAlloc** **memleakOnRealloc** | Doesn't use return value of memory allocation function |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | [Insert text.] | 2024.2 | **CL.FFM.ASSIGN** **CL.FFM.COPY** **CL.SHALLOW.ASSIGN** **CL.SHALLOW.COPY** **FMM.MIGHT** **FMM.MUST** |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | MSC11-C | Incorporate diagnostic tests using assertions |

| **Noncompliant Code** |
| --- |
| This code uses assert() macro to verify that memory allocation succeeded. However, because memory availability depends on the overall state of the system, using assert() to verify that memory allocation succeeded is inappropriate because doing so might lead to an abrupt termination of the process, potentially opening to the possibility of denial-of-service attack. |
| char \*dupstring(const char \*c\_str) {    size\_t len;    char \*dup;      len = strlen(c\_str);    dup = (char \*)malloc(len + 1);    assert(NULL != dup);      memcpy(dup, c\_str, len + 1);    return dup;  } |

| **Compliant Code** |
| --- |
| The compliant code demonstrates returning NULL in the case of memory exhaustion |
| char \*dupstring(const char \*c\_str) {    size\_t len;    char \*dup;      len = strlen(c\_str);    dup = (char\*)malloc(len + 1);    /\* Detect and handle memory allocation error \*/    if (NULL == dup) {        return NULL;    }      memcpy(dup, c\_str, len + 1);    return dup;  } |

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

| **Principles(s):** 4 – Keep it simple: When testing, it is best practice to test individual components down to the smallest minimum possible. Another way to think of this is to only test a single function at a time.  9 – Use Effective Quality Assurance Techniques: Diagnostic tests using assertions is a great way to test a codebase and make sure variables are being stored properly, functions behave correctly, and much more. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | High | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | |  |  | | --- | --- | |  | **LANG.FUNCS.ASSERTS** | | Not enough assertions |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **ASSERT\_SIDE\_EFFECT** | |  |  | | --- | --- | |  | Can detect the specific instance where assertion contains an operation/function call that may have a side effect | |
| |  |  | | --- | --- | | [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) |  | | 2023.1 | **CERT\_C-MSC11-a** | Assert liberally to document internal assumptions and invariants |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR51-CPP | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| In this example, neither f() or main() catch exceptions thrown by throwing\_func(). |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    f();  } |

| **Compliant Code** |
| --- |
| In this example, the main entry point handles all exceptions, wnsuring the stack is unwound up to main(). |
| 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):** 2 – Heed Compiler Warnings: Handling exceptions is a vital part of testing and debugging a program. Ensuring the correct exceptions are thrown at the correct times is of utmost importance for the dev cycle. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | 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 |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | |  |  | | --- | --- | |  | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | OOP53-CPP | Write constructor member initializers in the canonical order |

| **Noncompliant Code** |
| --- |
| In this example, the member initializer list attempts to initialize someVal first and then initialize dependsOnSomeVal. Because the members aren’t initialized inorder, the code fails. |
| class C {    int dependsOnSomeVal;    int someVal;    public:    C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| The initialization order is swapped around, enabling the function to perform. |
| class C {    int someVal;    int dependsOnSomeVal;    public:    C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

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

| **Principles(s):** 4 - Keep It Simple: Simple functions and members are easier to follow and diagnose if something goes wrong when compiling the codebase.  10 – Adopt a Secure Coding Standard: Adopting a standard for a team to work with enables ease of development due to their being no questions in how member functions should be written. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **initializer-list-order** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-OOP53** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.STRUCT.INIT.OOMI** | Out of Order Member Initializers |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **206 S** | Fully implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | FIO51-CPP | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| In this example, the file is constructed but never closed, therefor not destructing the file. |
| #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** |
| --- |
| This example correctly closes the file by calling fstream::close() before std::terminate(), ensuring the file is 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):** 9 – Use Effective Quality Assurance Techniques: Ensuring a file is closed helps ensure that no data is inadvertently written to a file where it shouldn’t be. Without this, bugs or corrupted data may occur. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Low | Low | Low | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **ALLOC.LEAK** | Leak |
| Helix QAC | 2024.2 | **DF4786, DF4787, DF4788** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | |  |  | | --- | --- | |  | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | | Checks for resource leak (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Error Handling | ERR62-CPP | Detect errors when converting a string to a number |

| **Noncompliant Code** |
| --- |
| In this example, multiple numeric values are converted from the standard input stream. However, if the text received from the standard input cannot be converted into a numeric value represented by int, the value stored into the variables I and j may be unexpected. |
| #include <iostream>    void f() {    int i, j;    std::cin >> i >> j;    // ...  } |

| **Compliant Code** |
| --- |
| The compliant code has exceptions enabled so that any conversion failures results in an exception being thrown. |
| #include <iostream>    void f() {    int i, j;      std::cin.exceptions(std::istream::failbit | std::istream::badbit);    try {      std::cin >> i >> j;      // ...    } catch (std::istream::failure &E) {      // Handle error    }  } |

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

| **Principles(s):** 2 – Heed Compiler Warnings: Fixing compiler warnings as they come up is important to fix as the effected code may have dependencies that require the code that is throwing the warning. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR62** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-err34-c | Checked by clang-tidy; only identifies use of unsafe C Standard Library functions corresponding to ERR34-C |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | |  |  | | --- | --- | |  | **BADFUNC.ATOF BADFUNC.ATOI BADFUNC.ATOL BADFUNC.ATOLL** | | Use of atof Use of atoi Use of atol Use of atoll |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | |  |  | | --- | --- | |  | R2024a | | [CERT C++: ERR62-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr62cpp.html) | Checks for unvalidated string-to-number conversion (rule fully covered) |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STR50-CPP | High | Likely | High | High | 1 |
| INT50-CPP | Low | Likely | Low | Medium | 4 |
| STR51-CPP | Medium | Low | Low | Medium | 3 |
| STR30-C | Medium | Low | Low | Low | 4 |
| MEM31-C | High | Low | Medium | High | 2 |
| MSC11-C | Low | High | Low | High | 4 |
| ERR51-CPP | Low | Medium | Low | Medium | 3 |
| OOP53-CPP | Low | Low | Medium | High | 2 |
| FIO51-CPP | Low | Low | Low | Low | 4 |
| ERR62-CPP | Medium | Medium | Low | Medium | 3 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest refers to any data that is in an encrypted state while it is in storage. Even if access to the data is gained, it will be unreadable without having the key to decrypt it. This is an important policy that provides a layer of defense in case a data breach happens. The data will be out but unreadable without the key. |
| Encryption in flight | Encryption in flight refers to any data in an encrypted state while it is in transit from one place to another. This ensures that if the data is intercepted while in transit, it will be unreadable because the interceptor doesn’t have the key to decode it. |
| Encryption in use | Encryption in use refers to the encryption of data while it is being used by a system. It is used to keep the data secure even as it is changing. It is important to encrypt data while in use because otherwise it would be the weak link between data at rest and data in flight. This leaves no hole where the data is ever decrypted. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of confirming a user is who they say they are. The most common use of authentication is when a user attempts to login to a system. This policy applies whenever a user attempts to gain access to a system or something within a system. The typical way authentication is implemented is by using usernames and passwords. |
| Authorization | Authorization is the process of confirming a user is supposed to have access to the system they are requesting. Authorization confirms the level of access a user has within a system. Once a user is authenticated, they are free to access parts of a system they have access to. It helps keep users out of places that they shouldn’t be. |
| Accounting | Accounting is the process of tracking and logging all requests made by a user. One example would be logging the changes a user makes to a database or files. It is important to keep this log to determine what users do, and have a trace if something goes wrong. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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