

# Green Pace Secure Development Policy

### Southern New Hampshire University

# 6 - 2 Project One: Security Policy

### Bairon Gomez

### Professor Alex Pettit

### CS 405 – Secure Coding

**October 4, 2023**

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validating input data is one of the primary methods used to avoid malicious attacks in software applications. This method consists of validating data before it is used or processed within the system. It is important to catch malicious data from data input as soon as it is entered in the system. Establishing a system that flags suspicious data and test the information to determine whether the data will be harmful or not is the key to prevent vulnerabilities from being exploited. There are different types of data inputs that can exploit the system, but they are mainly from untrusted external sources. However, validating the input data from trusted and untrusted sources will effectively contribute to securing the system. |
| 1. Heed Compiler Warnings | Warnings are important and relevant for developers to read, inspect, and act upon if necessary. Implementing this practice will prevent bugs and unforeseen issues from arising due to negligence from the warnings. The warnings generated may not be a liability for unsafe code, but if it is not looked at it can unravel into a problem that will require resources to fix. These warnings are visible when compiling the code and should be set to ensure that the program is running accordingly. Some methods used to generate compiler warnings that can stop the program as soon as a flaw is detected are dynamic and static analysis methods. |
| 1. Architect and Design for Security Policies | While designing the architecture of the application, it is important to incorporate security policies. Neglecting to consider and implement any security policies until the end can will result in unforeseen issues and restructuring the majority of the project. Incorporating security policies in each phase of the architecture design will lead to the development of a strong secure project. |
| 1. Keep It Simple | The complexity of the system has a direct impact over the performance, maintainability, and legibility. Keeping the system simple and straight forward with comments within the code will improve the overall end result of the system and reduce the creation of mistakes such as bugs and errors. |
| 1. Default Deny | Having default deny as a security measure in the system is highly important and valuable. What default deny adds into the system security wise is denying access by default to the systems database, user accounts, etc. unless the proper authentication is presented. Preventing users from accessing valuable information in the system will secure the overall program from users that have malicious intents. |
| 1. Adhere to the Principle of Least Privilege | Minimizing the accessibility of users in the system can prevent the program from being exploited. Not only does this method mitigate risks, but it also contributes to accountability. Reducing the accessibility to a handful of employees that require credentials to enter the system enables a track record of what changes were made and who accessed the system. Using this method will beneficial to reduce risks and to maintain a track record of changes made. |
| 1. Sanitize Data Sent to Other Systems | Checking and validating data that is sent to external sources will prevent malicious or compromised data from being transmitted. Ensuring that data is checked and validated to be clean from malware will prevent infecting external systems. This process is important to maintain a reputable reputation. |
| 1. Practice Defense in Depth | Establishing multiple layers of security to mitigate and prevent any malware from accessing the system is known as defense in depth. Incorporating this method will create layers of security as a safety net after one layer has been bypassed. It is important to have different layers of security to contain a well secured system, but there are some key aspects to consider when using this method. This method can get complex to incorporate and maintain over time since it deals with different type of security layers that are different from each other. |
| 1. Use Effective Quality Assurance Techniques | Using effective quality assurance techniques results in accountability, testing, improvements, and delivering the best results. Implementing this method can be done in different way, but the key practices to deliver the best results are directly associated with testing and improving current or previous work. Testing code from the developer standpoint and user perspective to gain performance and user experience information to improve the system will deliver a solid well grounded project. |
| 1. Adopt a Secure Coding Standard | Incorporating secure coding standards is applicable and desirable in every project. Without it, vulnerabilities would be left unattended which will lead to compromised data and system extortion. Applying coding consistency, vulnerability prevention, code review, and constant training are the some practices that contribute to secure coding standards. The coding standards may vary slightly depending on what companies value, but overall they are composed of similar traits that all correlate with securing the system. Using secure coding practice is applicable and should always be applicable to any development language and project. |

### 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 example attempts to check whether a given value is within the range of acceptable enumeration values, but with one key mistake. The mistake in the code is having the enumeration type assigned first before checking if it is within range. |
| **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 to guarantee the conversion does not result in an unspecified value. This is done by restricting the converted value to one for which there is 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):** Keep it Simple – Preventing complications or errors in calculations it is essential to implement and assign integers/buffers in the following manner: straightforward and concise |
| --- |

**Threat Level**

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

**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 |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | Cast-integer-to-enum | Partially Checked |

#### Coding Standard 2

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

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, pos is invalidated after the first call to insert(), and subsequent loop iterations have undefined 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** |
| --- |
| In this compliant solution, pos is assigned a valid iterator on each insertion, preventing undefined 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) {      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):** Architect and Design for Security Policies – When we design the structure and plan how our program uses memory, we can prevent common mistakes and make it more secure. By making sure we only access things in the right way, we avoid making the program vulnerable and breaking security rules. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | ALLOC.UAF | Use After Free |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Overflow\_upon\_dereference |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP\_CTR51-a | Do not modify container while iterating over it |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | CERT C++: CTR51-CPP | Checks for use of invalid iterator (rule partially covered) |

#### 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** |
| --- |
| In this noncompliant code example, a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist (or some other error occurs). |
| #include <cstdlib>  #include <string>    **void** f() {    std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| In this compliant 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):** Validate Input Data – Validating data is essential to prevent buffer overflow, null pointers, and unpredictable outcome. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule partially covered) |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-STRS1-a | Avoid null pointer dereferencing |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | LANG.MEM.NPD | Null Pointer Dereference |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Assert\_failure |  |

#### Coding Standard 4

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

| **Noncompliant Code** |
| --- |
| The example below contains two unrelated smart pointers that are constructed from the same underlying pointer value. When the local, automatic variable p2 is destroyed, it deletes the pointer value it manages. Then, when the local, automatic variable p1 is destroyed, it deletes the same pointer value, resulting in 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** |
| --- |
| The solution below contains then std::shared\_ptr objects are related to one another through copy construction. When the local, automatic variable p2 is destroyed, the use count for the shared pointer value is decremented but still nonzero. Then, when the local, automatic variable p1 is destroyed, the use count for the shared pointer value is decremented to zero, and the managed pointer is destroyed. This compliant solution also calls std::make\_shared() instead of allocating a raw pointer and 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):** Validate Input Data – Validating input data prevents malware from entering the system and checks the if the data is safe to process. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Dangling\_pointer\_use |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-MEM56 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-MEM56-a | Do not store an already owned pointer value in a unrelated smart pointer |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | CERTC++: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-cpp] | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| Below is a an example that results in undefined behavior. The local variable space is passed as the expression to the placement new operator. The resulting pointer of that call is then passed to ::operator delete(), resulting in undefined behavior due to ::operator delete() attempting to free memory that was not returned 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** |
| --- |
| The correct compliant code is performed by removing the call to::operator delete() instead of explicitly calling s1’s destructor. This example showcases one of the few times when explicitly invoking a destructor is warranted. |
| #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):** Keep it Simple – Memory management aligns with keeping things simple by preventing unnecessary the usage of unnecessary bits. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM  ALLOC.LEAK | Fee non-heap variable. Double free Type mismatch Leak. |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D | Partially Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d | Use the same form in corresponding call to new/malloc and delete/free. Always provide empty brackets for delete when deallocating arrays. |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | R2023a |  | Runtime detection |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete  clang-analyzer-unix.MismatchedDeallocator | Checked y clang-tidy, but does not catch all violations of this rule |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | CERT C++: MEM51-CPP | Checks for:   * Invalid deletion of pointer * Invalid free of pointer * Deallocation of previously deallocated pointer |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| The noncompliant code uses the assert() macro to assert() 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** |
| --- |
| In this compliant solution the assertion only involving a constant expression, a preprocessor conditional statement may be used, as shown in the solution. |
| #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**));  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – Identifying issues early will prevent the problem from escalating. Using static assertions provides developers with quality assurance and can eliminate defects in doing so.  Keep it Simple – Do not make the program complex without having the need to do so. Create or use assertions in the simplest implications to find problems hidden in the code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-DCL03 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | Misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | CC2.DCL03 | Fully Implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 44 S | Fully Implemented |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| The example shows the constructor for S may throw an exception that is not caught when globalS is constructed during program startup. |
| **struct** S {    S() noexcept(**false**);  };    **static** S globalS; |

| **Compliant Code** |
| --- |
| The compliant solution makes globalS into a local variable with static storage duration, allowing any exceptions thrown during object construction to be caught because the constructor for S will be executed the first time the function globalS() is called rather than at program startup. This solution does require the programmer to modify source code so that previous uses of globalS are replaced by a function call to globalS(). |
| **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):** Architect and Design for Security Policies – With a well thought out design developers can catch exceptions  Keeping it Simple - Developers accustomed to writing straight-forward and clean code make it simple for others to understand their objective. This contributes to validating if assertions are working as intended when used. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Potentially-throwing-static-initialization | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | Cert-err58-cpp | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | LANG.STRUCT.EXCP.THROW | Use of Throw |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP\_ERR58-a | Exceptions shall be raised only after start-up and before termination of the program |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | CERT C++: ERR58-CPP | Checks for exceptions raised during program startup |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-cppL] | Do not use pointer-to-member operators to access nonexistent members |

| **Noncompliant Code** |
| --- |
| The noncompliant example below shows a pointer-to-member object is obtained from D::g but is then upcast to be a B::\*. When called on an object whose dynamic type is D, the pointer-to-member call is well defined. However, the dynamic type of the underlying object is B, which results in undefined behavior. |
| **struct** B {  **virtual** ~B() = **default**;  };    **struct** D : B {  **virtual** ~D() = **default**;  **virtual** **void** g() { /\* ... \*/ }  };    **void** f() {    B \*b = **new** B;      // ...    **void** (B::\*gptr)() = **static\_cast**<**void**(B::\*)()>(&D::g);    (b->\*gptr)();  **delete** b;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the upcast is removed, rendering the initial code ill-formed and emphasizing the underlying problem that B::g() does not exist. This compliant solution assumes that the programmer's intention was to use the correct dynamic type for the underlying object. |
| **struct** B {  **virtual** ~B() = **default**;  };    **struct** D : B {  **virtual** ~D() = **default**;  **virtual** **void** g() { /\* ... \*/ }  };    **void** f() {    B \*b = **new** D; // Corrected the dynamic object type.      // ...  **void** (D::\*gptr)() = &D::g; // Moved static\_cast to the next line.    (**static\_cast**<D \*>(b)->\*gptr)();  **delete** b;  } |

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

| **Principles(s):**  Keep it Simple – Creating simple and straight-forward code has many benefits. One of those benefits is preventing the access to members that do not exist.  Architect and Design for Security Policies – Establishing clear policies that outline violations of security implication can highlight the outcome of accessing members that do not exist. This action would lead to unpredictable behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finde](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | CERT C++: OOP55-CPP | Checks for pointers to member accessing non-existing class members |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime Detection |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-OOP55-a | A cast shall not convert a pointer to a function to any other pointer type, including a pointer to a function type. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | LANG.MEM.UVAR | Uninitialized Variable |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-cpp] | Range check element access |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the value returned by the call to get\_index() may be greater than the number of elements stored in the string, resulting in undefined behavior. |
| #include <string>    **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| This compliant solution uses the std::basic\_string::at() function, which behaves in a similar fashion to the index operator[] but throws a std::out\_of\_range exception if pos >= size(). |
| #include <stdexcept>  #include <string>  **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");  **try** {      s.at(get\_index()) = '1';    } **catch** (std::out\_of\_range &) {      // Handle error    }  } |

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

| **Principles(s):**  Use Effective Quality Assurance Techniques – Quality assurance is the gatekeeper to enforcing proper use of element check. If the quality assurance neglects validating elements it can lead to out of bound reads/writes.  Adopt a Secure Coding Standard - Adopting secure coding standards falls in the category of ensuring that elements are within the appropriate range. If elements are not checked to be within range it leaves the program vulnerable. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | CERT C++: STR53-CPP | Checks for:   * Array Access out of bounds * Array access with tainted index * Pointer dereference with tainted offset |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-STR53-a | Guarantee that container indices are within the valid range |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | * Buffer overrun * Buffer underrun * Tainted buffer access * Type overrun * Type underrun |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | assert\_failure |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-cpp] | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| This code example is noncompliant because the resource allocated by the call to fopen() is not closed before the program terminates. Although exit() closes the file, the program has no way of determining if an error occurs while flushing or closing the file. |
| #include <stdio.h>  #include <stdlib.h>    **int** main(**void**) {  **FILE** \*f = **fopen**(filename, "w");  **if** (NULL == f) {  **exit**(EXIT\_FAILURE);    }    /\* ... \*/  **exit**(EXIT\_SUCCESS);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the program closes f explicitly before calling exit(), allowing any error that occurs when flushing or closing the file to be handled appropriately: |
| #include <stdio.h>  #include <stdlib.h>    **int** main(**void**) {  **FILE** \*f = **fopen**(filename, "w");  **if** (NULL == f) {      /\* Handle error \*/    }    /\* ... \*/  **if** (**fclose**(f) == EOF) {      /\* Handle error \*/    }  **exit**(EXIT\_SUCCESS);  } |

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

| **Principles(s):**  Adopt a Secure Coding Standard – Closing files can prevent information from being leaked or consume unnecessary bits in the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 |  | Supported, but no explicit checker |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | ALLOC.LEAK | Leak |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | RESOURCE\_LEAK | Partially Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | CERT\_C-FIO42-a | Ensure resources are freed |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023A | CERT C: Rule FIO42-C | Checks for resource leak |

### 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.

* To automate enforcing our standards, we need to create a central policy repository. This repository will be part of our assessment and planning process. It's where we store all our organizational policies, like compliance with regulations, release requirements, and general organization rules. This data is crucial for starting policy automation and enforcement. During the verification and testing phase, we'll use this repository. Automated tools will assess risks, send alerts, and provide notifications. Automating compliance helps us work more efficiently and avoid repeating tasks. We can also automate the transition and health check phases. By using automated penetration testing, we can reduce false alarms and make fixing issues smoother.
* Going a step further, we can fully automate the rest of the production process. We can set up a system to make logs and keep them in a database. This database will help us find vulnerabilities and stop potential attacks. We'll do this with methods like signature checks, ensuring data integrity, and using multiple security layers. To keep our system safe and steady, we can create automatic save points. These save points act like clean backups, so if there's an attack or a system problem, we can get back to a secure version.

### 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-003-CPP | HIGH | LIKELY | MEDIUM | P18 | L1 |
| STD-004-CPP | HIGH | LIKELY | MEDIUM | P18 | L1 |
| STD-005-CPP | HIGH | LIKELY | MEDIUM | P18 | L1 |
| STD-002-CPP | HIGH | PROBABLE | HIGH | P6 | L2 |
| STD-008-CPP | HIGH | PROBABLE | HIGH | P6 | L2 |
| STD-009-CPP | HIGH | UNLIKELY | MEDIUM | P6 | L2 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-001-CPP | MEDIUM | Unlikely | Medium | P4 | L3 |
| STD-006-CPP | LOW | UNLIKELY | HIGH | P1 | L3 |
| STD-007-CPP | LOW | LIKELY | LOW | P9 | L2 |

### 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 | Protecting data that is stored in devices is known a encryption at rest. The data could be stored in hard drives or databases but as long as it is protected with encryption in those states it all fall under the same category. Incorporating this method of encryption in the policy safeguards data from a data breach scenario. Hackers may be able to obtain the data, but not view what it contains due to encryption at rest methods being applied. In other words the data is unreadable and useless for the hacker. If encryption at rest is not applied to data it is prone to being exploited if the system has a data breach. |
| Encryption at flight | Securing data while it travels from point A to point B is the simplest way to describe encryption in flight. Data/information is constantly on the move. It travels from the network to the user or vice verse. When data is on the move it can be intercepted which is why it is important to encrypt data while it is in flight. Doing so will safeguard data from being viewed by unathorized individuals. The data is encrypted before it is in transit and is decrypted once it has reached its destination. Applying this policy mitigates the damage that can be done if data is breached. |
| Encryption in use | Safety measures can be used to restrict anyone other than the intended user from accessing data that is being used (manipulated) in software applications. Data is vulnerable while it is in an active state, but encrypting data while it is in the computers memory prevents attackers from eavesdropping. This policy applies and is imporatant to secure data from being prone to exploitation. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying that the user intending to log into the system has the proper credentials to access the system. This limits access to the system based on the authorization given by admin. Systems can have different methods to implement authentication such as physical key card, biometric features, or two factor authentication. This policy is applicable due to its security features of limiting access to authorized personal. |
| Authorization | Authorization focuses on limiting what actions a user can preform while they are in the network. Limiting what people can do in the system reduces the possibility of malware or vulnerabilities from being inserted. When applying authorization, the best way to think about is by sticking to the saying “less is best”. Meaning that the less access users have, the better it will be to prevent attacks. |
| Accounting | Accounting is described as keeping a log of the users actions while they are in the system. Keeping a record of what has been done while individuals are in the network will allow admin to have a backlog to view when suspicious activities are detected. This method can also be used to alert admin if their has been attempts to access or alter unathorized changes. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 10/04/23 | Project 1 | Bairon Gomez |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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