**Green Pace Developer: Security Policy Guide Template**



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 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 12](#_Toc52464063)

[Coding Standard 6 15](#_Toc52464064)

[Coding Standard 7 17](#_Toc52464065)

[Coding Standard 8 19](#_Toc52464066)

[Coding Standard 9 22](#_Toc52464067)

[Coding Standard 10 25](#_Toc52464068)

[Defense-in-Depth Illustration 27](#_Toc52464069)

[Project One 27](#_Toc52464070)

[1. Revise the C/C++ Standards 27](#_Toc52464071)

[2. Risk Assessment 27](#_Toc52464072)

[3. Automated Detection 27](#_Toc52464073)

[4. Automation 27](#_Toc52464074)

[5. Summary of Risk Assessments 28](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 29](#_Toc52464076)

[7. Map the Principles 30](#_Toc52464077)

[Audit Controls and Management 31](#_Toc52464078)

[Enforcement 31](#_Toc52464079)

[Exceptions Process 31](#_Toc52464080)

[Distribution 32](#_Toc52464081)

[Policy Change Control 32](#_Toc52464082)

[Policy Version History 32](#_Toc52464083)

[Appendix A Lookups 32](#_Toc52464084)

[Approved C/C++ Language Acronyms 32](#_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 | Insert Input validation ensure that bogus data does not pass validation. If the wrong input is entered and does not match the specific requirements, an error will occur . Simplest example would be entering letters in a numerical field or inputting lesser than the required number of digits in a field, ex a field of date of birth when 2 inputs are typed where MM/DD/YYYY 8 are required, or a date of birth prior to 1920, etc. |
| 1. Heed Compiler Warnings | It is recommended to use the highest warning level for the complier. It is also recommended to eliminate warnings rather than ignore them as they possess security flaws and vulnerability concerns. |
| 1. Architect and Design for Security Policies | Ensure up to date security policies are implemented when designing the software architecture. |
| 1. Keep It Simple | Avoid complex system and algorithms that can cause error or prone to vulnerabilities. Keep the code design small and simple |
| 1. Default Deny | Base access decisions based on permissions rather than exclusions. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least privileges to compile. While the most privileged tasks should be compiled executing the least amount of time. |
| 1. Sanitize Data Sent to Other Systems | Sanitize the data passed through subsystems. |
| 1. Practice Defense in Depth | Manage risks by implementing defense in depth strategies. The DiD if one layer of defense is compromised, another layer would intervene and attempt to negate the threat. |
| 1. Use Effective Quality Assurance Techniques | Penetration testing is a critical technique to eliminate vulnerabilities. Independent reviews are also recommended. All these techniques combined can lead to a more secure system. |
| 1. Adopt a Secure Coding Standard | Develop a secure coding standard for development regardless 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-C++] | **Do not cast to an out-of-range enumeration value** |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the 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 integer value. On a two's complement system, the valid range of values that can be represented by EnumType are [0..3], so if a value outside of that range were passed to f(), the cast to EnumType would result in an unspecified value, and using that value within the if statement results in [unspecified behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-unspecifiedbehavior). |
| **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. It does this 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):** It is possible for unspecified values to result in a buffer overflow, leading to the  execution of arbitrary code by an attacker. However, because enumerators are rarely used for  indexing into arrays or other forms of pointer arithmetic, it is more likely that this scenario will  result in data integrity violations rather than arbitrary code execution. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion  Bauhaus  Suite | 7.2.0 | CertC++-  Int50 |  |
| Helix QAC | 2021.2 | C++3013 |  |
| Parasoft  C/C++test | 2021.1 | CERT\_CPP-  INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| PRQA  QA-C++ | 4.4 | 3013 | [ |
| PVS-  Studio | 7.13 | V1016 |  |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-C++] | **Do not rely on the value of a moved from object.** |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the integer values 0 through 9 are expected to be printed to the standard output stream from a std::string rvalue reference. However, because the object is moved and then reused under the assumption its internal state has been cleared, unexpected output may occur despite not triggering [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <iostream>  #include <string>    void g(std::string v) {    std::cout << v << std::endl;  }    void f() {    std::string s;    for (unsigned i = 0; i < 10; ++i) {      s.append(1, static\_cast<**char**>('0' + i));      g(std::move(s));    } |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::string object is initialized to the expected value on each iteration of the loop. This practice ensures that the object is in a valid, specified state prior to attempting to access it in g(), resulting in the expected output. |
| [#include <iostream>  #include <string>    void g(std::string v) {    std::cout << v << std::endl;  }    void f() {    for (unsigned i = 0; i < 10; ++i) {      std::string s(1, static\_cast<**char**>('0' + i));      g(std::move(s));    }  } |

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

|  |
| --- |
| **Principles(s):** The state of a moved-from object is generally valid, but unspecified. Relying on unspecified values can lead to abnormal program termination as well as data integrity violations. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix-QAC | 2021.2 | C++4701,  C++4702,  C++4703 |  |
| Parasoft  C/C++test | 2021.1 | CERT\_CPP-  EXP63-a | Do not rely on the value of a moved from object. [ |
| Polyspace  Bug  Finder | R2021a | CERT C++:  EXP63-CPP | Checks for read operations that reads the value of a moved from object (rule fully covered) |
| PVS-  Studio | 7.13 | V1030 |  |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-C++] | **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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions%22%20%5Cl%20%22BB.Definitions-undefinedbehavior) 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):** Dereferencing a null pointer is [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior), typically [abnormal program termination](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-abnormaltermination). In some situations, however, dereferencing a null pointer can lead to the execution of arbitrary code [[Jack 2007](https://wiki.sei.cmu.edu/confluence/display/cplusplus/AA.+Bibliography#AA.Bibliography-Jack07), [van Sprundel 2006](https://wiki.sei.cmu.edu/confluence/display/cplusplus/AA.+Bibliography#AA.Bibliography-vanSprundel06)]. The indicated severity is for this more severe case; on platforms where it is not possible to [exploit](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions) a null pointer dereference to execute arbitrary code, the actual severity is low. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | assert\_failure |  |
| Helix QAC | 2021.2 | C++4770,C++4771,  C++4772,C++4773,  C++4774 |  |
| Klocwork | 2021.1 | [**NPD.CHECK.CALL.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**NPD.CHECK.CALL.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**NPD.CHECK.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)  [**NPD.CHECK.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)  [**NPD.CONST.CALL**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**NPD.CONST.DEREF**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**NPD.FUNC.CALL.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**NPD.FUNC.CALL.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**NPD.FUNC.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**NPD.FUNC.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)  [**NPD.GEN.CALL.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)[**NPD.GEN.CALL.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)[**NPD.GEN.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)[**NPD.GEN.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)  [**RNPD.CALL**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)[**RNPD.DEREF**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| Parasoft  C?C++test | 2021.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

### Coding Standard 4

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

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, two unrelated smart pointers 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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions%22%20%5Cl%20%22BB.Definitions-vulnerability). |
| #include <memory>    void f() {  **int** \*i = new **int**;    std::shared\_ptr<**int**> p1(i);    std::shared\_ptr<**int**> p2(i);  } |

| **Compliant Code** |
| --- |
| #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):** Passing a pointer value to a deallocation function that was not previously obtained by the matching allocation function results in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior), which can lead to exploitable [vulnerabilities](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-vulnerability). |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrėe | 20.10 | Dangling\_pointer\_use |  |
| Axivion  Bauhaus  Suite | 7.2.0 | CertC++-MEM56 |  |
| Helix QAC | 2021.2 | C++4721,C++4722,C++4723 |  |
| Parasoft  C/C++test | 2021.1 | CERT\_CPP-MEM56-a | Do not store an already owned value in an unrelated smart pointer |
| Polyspace  Bug  Finder | R2021a | CERT C++: MEM56-CPP | Checks for use of already owned pointers (rule fully covered) |
| PVS-Studio | 7.13 | V1006 |  |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-C++] | **Do not access freed memory** |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the [vulnerability](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-vulnerability) can be [exploited](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-exploit) to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problem |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...    s->f();  **delete** s; |

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

|  |
| --- |
| **Principles(s):** Reading previously dynamically allocated memory after it has been deallocated can lead to [abnormal program termination](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-abnormaltermination) and [denial-of-service attacks](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-denial-of-service). Writing memory that has been deallocated can lead to the execution of arbitrary code with the permissions of the vulnerable process. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrėe | 20.10 | Dangling\_pointer\_use |  |
| Axivion  Bauhaus  Suite | 7.2.0 | CertC++-MEM50 |  |
| Clang | 3.9 | Clang-analyzer-  Cplusplus.NewDelete  Clang-analyzer-  Alpha.security.ArrayBoundv2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| CodeSonar | 6.1p0 | ALLOC.UAF | Use after free |
| Compass/ROSE |  |  |  |
| Coverity | V7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Helix QAC | 2021.2 | C++4303, C++4304 |  |
| Klocwork | 2021.1 | [**UFM.DEREF.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.DEREF.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.FFM.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.FFM.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.RETURN.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.RETURN.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.USE.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**UFM.USE.MUST**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| LDRA tool suite | 9.7.1 | 483 S, 484 S | Partially implemented |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| Parasoft  Insure ++ |  |  | Runtime detection |
| Polyspace  Bug Finder | R2021a | CERT C++:MEM50-CPP | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |
| PRQA QA-C++ | 4.4 | 4303, 4304 |  |
| PVS-Studio | 7.13 | V586, v774 |  |
| Splint | 5.0 |  |  |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-C++] | Use a 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** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution: |
| **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):** Static assertion is a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities at compile time. The absence of static assertions, however, does not mean that code is incorrect. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion  Bauhaus Suite | 7.2.0 | CertC-DCLO3 |  |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 6.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| Compass/ROSE |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-C++] | **Do not let exceptions escape from destructors and deallocation functions** |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the class destructor does not meet the implicit noexcept guarantee because it may throw an exception even if it was called as the result of an exception being thrown. Consequently, it is declared as noexcept(false) but still can trigger [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <stdexcept>    **class** S {  **bool** has\_error() **const**;    **public**:    ~S() noexcept(**false**) {      // Normal processing  **if** (has\_error()) {  **throw** std::logic\_error("Something bad");      }    }  }; |

| **Compliant Code** |
| --- |
| Use of std::uncaught\_exception() in the destructor solves the termination problem by avoiding the propagation of the exception if an existing exception is being processed, as demonstrated in this noncompliant code example. However, by circumventing normal destructor processing, this approach may keep the destructor from releasing important resources. |
| #include <exception>  #include <stdexcept>    **class** S {  **bool** has\_error() **const**;    **public**:    ~S() noexcept(**false**) {      // Normal processing  **if** (has\_error() && !std::uncaught\_exception()) {  **throw** std::logic\_error("Something bad");      }    } |

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

|  |
| --- |
| **Principles(s):** Attempting to throw exceptions from destructors or deallocation functions can result in undefined behavior, leading to resource leaks or [denial-of-service attacks](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-denial-of-service). |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrėe | 20.10 | **destructor-without-noexcept delete-without-noexcept** | Fully checked |
| Axivion  Bauhaus  Suite | 7.2.0 | CertC++-  DCL57 | [Insert text.] |
| Helix QAC | 2021.2 | C++2045,  C++2047,  C++4032,  C++4631 |  |
| LDRA tool suite | 9.7.1 | 453 S | Partially implemented |
| Polyspace  Bug Finder | R2021a | CERT C++  DCL57-CPP | Checks for class destructors exiting with an exception (rule partially covered) |
| PVS-Studio | 7.13 | V509,V1045 |  |
| RuleChecker | 20.10 | Destructor-  without-no except  delete-without-  no except | Fully Checked |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-C++] | **Store a new value in pointers immediately after free()** |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the type of a message is used to determine how to process the message itself. It is assumed that message\_type is an integer and message is a pointer to an array of characters that were allocated dynamically. If message\_type equals value\_1, the message is processed accordingly. A similar operation occurs when message\_type equals value\_2. However, if message\_type == value\_1 evaluates to true and message\_type == value\_2 also evaluates to true, then message is freed twice, resulting in a double-free vulnerability. |
| **char** \*message;  **int** message\_type;    /\* Initialize message and message\_type \*/    **if** (message\_type == value\_1) {    /\* Process message type 1 \*/  **free**(message);  }  /\* ...\*/  **if** (message\_type == value\_2) {     /\* Process message type 2 \*/  **free**(message);  } |

| **Compliant Code** |
| --- |
| Calling free() on a null pointer results in no action being taken by free(). Setting message to NULL after it is freed eliminates the possibility that the message pointer can be used to free the same memory more than once. |
| **char** \*message;  **int** message\_type;    /\* Initialize message and message\_type \*/    **if** (message\_type == value\_1) {    /\* Process message type 1 \*/  **free**(message);    message = NULL;  }  /\* ... \*/  **if** (message\_type == value\_2) {    /\* Process message type 2 \*/  **free**(message);    message = NULL;  } |

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

|  |
| --- |
| **Principles(s):** Setting pointers to NULL or to another valid value after memory is freed is a simple and easily implemented solution for reducing dangling pointers. Dangling pointers can result in freeing memory multiple times or in writing to memory that has already been freed. Both of these problems can lead to an attacker executing arbitrary code with the permissions of the vulnerable process. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrėe | 20.10 |  | Supported: Astrée reports usage of invalid pointers. |
| Axivion  Bauhaus Suite | [Insert text.] | CertC-MEM01 | Fully implemented |
| CodeSonar | [Insert text.] | ALLOC.DF  ALLOC.UAF | Double free yser after free |
| Compass/ROSE | [Insert text.] |  |  |
| Coverity |  | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| LDRA tool suite |  | 484 S, 112 D | Partially implemented |
| Parasoft C/C++ test |  | **CERT\_C-MEM01-a CERT\_C-MEM01-b CERT\_C-MEM01-c CERT\_C-MEM01-d** | Do not use resources that have been freed Always assign a new value to an expression that points to deallocated memory Always assign a new value to global or member variable that points to deallocated memory Always assign a new value to parameter or local variable that points to deallocated memory |
| Parasoft Insure++ |  |  | Detects dangling pointers at runtime |
| Polyspace Bug  Finder | R2021a | CERT C:Rec.  MEM01-C | Checks for missing reset of a freed pointer (rec. fully covered) |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-C++] | Be careful when using functions that use file names for identification. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the file identified by file\_name is opened, processed, closed, and removed. However, it is possible that the file object identified by file\_name in the call to remove() is not the same file object identified by file\_name in the call to fopen(). |
| **char** \*file\_name;  **FILE** \*f\_ptr;    /\* Initialize file\_name \*/    f\_ptr = **fopen**(file\_name, "w");  **if** (f\_ptr == NULL)  {    /\* Handle error \*/  }    /\* ... \*/    **if** (chmod(file\_name, S\_IRUSR) == -1) {    /\* Handle error \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution uses the POSIX fchmod() and open() functions [[IEEE Std 1003.1:2013](https://wiki.sei.cmu.edu/confluence/display/c/AA.+Bibliography#AA.Bibliography-IEEEStd1003.1-2013)]. Using these functions guarantees that the file opened is the same file that is operated on. |
| **char** \*file\_name;  **int** fd;    /\* Initialize file\_name \*/    fd = open(    file\_name,    O\_WRONLY | O\_CREAT | O\_EXCL,    S\_IRWXU  );  **if** (fd == -1) {    /\* Handle error \*/  }    /\* ... \*/    **if** (fchmod(fd, S\_IRUSR) == -1) {    /\* Handle error \*/  } |

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

|  |
| --- |
| **Principles(s):** Many file-related vulnerabilities, such as [time-of-check, time-of-use (TOCTOU)](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-TOCTOU) race conditions, can be exploited to cause a program to access an unintended file. Using FILE pointers or file descriptors to identify files instead of file names reduces the chance of accessing an unintended file. Remediation costs are medium because, although insecure functions can be easily identified, simple drop-in replacements are not always available. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | **IO.RACE**  **IO.TAINT.FNAME**  **BADFUNC.TEMP.\*** | File System Race Condition  Tainted Filename  A collection of warning classes that report uses of library functions associated with temporary file vulnerabilities (including name issues). |
| Compass/ROSE |  |  | Can detect some violations of this recommendation. In particular, it warns when chown(), stat(), or chmod() are called on an open file |
| Coverity | 6.5 | TOCTOU | Fully implemented |
| Helix QAC | 2021.2 | C5011 |  |
| Klocwork | 2021.1 | [**SV.TOCTOU.FILE\_ACCESS**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| LDRA tool suite | 9.7.1 | 592 S | Fully implemented |
| Parasoft C/C++ test | 2021.1 | **CERT\_C-FIO01-a** **CERT\_C-FIO01-b** | Don't use chmod(), chown(), chgrp() Usage of functions prone to race is not allowed |
| PRQA QA-C | 9.7 | 5011 | Partially implemented |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-C++] | **Do not make assumptions about the size of environment variable** |

| **Noncompliant Code** |
| --- |
| This noncompliant code example copies the string returned by getenv() into a fixed-size buffer: |
| **void** f() {  **char** path[PATH\_MAX]; /\* Requires PATH\_MAX to be defined \*/  **strcpy**(path, **getenv**("PATH"));    /\* Use path \*/  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the strlen() function is used to calculate the size of the string, and the required space is dynamically allocated: |
| **void** f() {  **char** \*path = NULL;    /\* Avoid assuming $PATH is defined or has limited length \*/  **const** **char** \*temp = **getenv**("PATH");  **if** (temp != NULL) {      path = (**char**\*) **malloc**(**strlen**(temp) + 1);  **if** (path == NULL) {        /\* Handle error condition \*/      } **else** {  **strcpy**(path, temp);      }      /\* Use path \*/  **free**(path);    }  } |

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

|  |
| --- |
| **Principles(s):** Making assumptions about the size of an environmental variable can result in a buffer overflow. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | **LANG.MEM.BO LANG.MEM.TO (general)** | Buffer overrun Type overrun CodeSonar's taint analysis includes handling for taint introduced through the environment |
| Compass/ROSE |  |  | Can detect violations of the rule by using the same method as [STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator](https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator) |
| Klockwork | 2021.1 | [**ABV.ANY\_SIZE\_ARRAY**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**ABV.GENERAL**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**ABV.ITERATOR**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**ABV.MEMBER**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**ABV.STACK**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**ABV.TAINTED**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**ABV.UNKNOWN\_SIZE**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**ABV.UNICODE.BOUND\_MAP**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**ABV.UNICODE.FAILED\_MAP**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**ABV.UNICODE.NNTS\_MAP**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**ABV.UNICODE.SELF\_MAP**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) | [Insert text.] |
| Parasoft C/C++ test | 2021.1 | **CERT\_C-ENV01-a** **CERT\_C-ENV01-b** **CERT\_C-ENV01-c** | Don't use unsafe C functions that do write to range-unchecked buffers Avoid using unsafe string functions which may cause buffer overflows Avoid overflow when writing to a buffer |
| PC\_lint plus | 1.4 | 669 | Fully supported |
| Polyspace Bug  Finder | R2021a | [CERT C: Rec. ENV01-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.env01c.html) | Checks for tainted NULL or non-null-terminated string (rec. partially 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.

DevSecOps represents that security implementations are everyone’s duty and should be standard practice during all stages of the software development workflow.

Assess and plan are the first step. Security should be discussed, and a plan should be created. Design is the second step, where security tools are selected, best practices. The build phase focuses on automated security analysis. Developers need to be aware of third-party code dependencies, untrusted sources. Some of the known tools to execute build phase analysis are OWASP, Snyk. The test phase uses dynamic testing to detect user authentication, authorization, and SQL injection to detect live flaws. The following phase is transition and health check. At this point, the code should be tested. The PoLP concern needs to be addressed to ensure access is limited to owners and designated personnel. Configuration management should be audited and reviewed. Respond phase blocks attacks and neutralize threats. In the event of any attack, use logs and assess the damage, and roll back to a state prior to the attack.

## 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 | Low | 1 |
| STD-002-CPP | Medium | Probable | Medium | Medium | 2 |
| STD-003-CPP | High | Likely | Medium | Medium | 3 |
| STD-004-CPP | High | Likely | Medium | Medium | 3 |
| STD-005-CPP | High | Likely | Medium | medium | 3 |
| STD-006-CPP | Low | Unlikely | High | Medium | 3 |
| STD-007-CPP | Low | Likely | Medium | High | 4 |
| STD-008-CPP | High | Unlikely | Low | Low | 1 |
| STD-009-CPP | Medium | Likely | Medium | High | 4 |
| STD-010-CPP | Medium | Likely | Medium | High | 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 in rest | Encryption at rest refers to encryption the data on the disk to prevent any unauthorized access by encrypting any unencrypted data. Examples ( Encrypting and decrypting files and databases) |
| Encryption at flight | Encryption at flight refers to the data being transmitted. The data needs to be encrypted while transmitted otherwise, we risk sensitive data being exposed. Data in movement is protected within channels. Examples (IPsec VPN, HTTPS(SSL, TLS) |
| Encryption in use | In use Encryption refers to ensuring sensitive data is always encrypted. This protects sensitive data at all times |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is verifying the identify of users and granting access to authenticated users according to clearance protocols. Authentication is needed in implementing security protocols. We need to ensure that only authenticated users can gain access to their designated accounts. |
| Authorization | Granting the authenticated users the content based on the proper privileges. Often centralizing and externalization authorization makes the entity safer. Users must gain authorization for doing tasks. |
| Accounting | Accounting measures the resources a user consumes during access. This can be the amount of time the data is sent/received during a session. |

**\***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 | 07/17/2021 | Initial Template | Michael Farag |  |
| 1.1 | 08/08/2021 | Final template | Michael Farag | [Insert text.] |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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

References

Mike BMike B 3, Dr. MohanDr. Mohan 2122 bronze badges, & AndyMacAndyMac 3. (1966, September 1). *What's a practical example of encryption "in use" or "in process"?* Information Security Stack Exchange. https://security.stackexchange.com/questions/182398/whats-a-practical-example-of-encryption-in-use-or-in-process.

Atlassian. (n.d.). DevSecOps tools. Atlassian. https://www.atlassian.com/devops/devops-tools/devsecops-tools.

*Confluence*. INT50-CPP. Do not cast to an out-of-range enumeration value - SEI CERT C++ Coding Standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/display/cplusplus/INT50-CPP.+Do+not+cast+to+an+out-of-range+enumeration+value.

*Confluence*. EXP63-CPP. Do not rely on the value of a moved-from object - SEI CERT C++ Coding Standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP63-CPP.+Do+not+rely+on+the+value+of+a+moved-from+object#:~:text=invoking%20its%20destructor.-,Do%20not%20rely%20on%20the%20value%20of%20a%20moved%2Dfrom,values%20leads%20to%20unspecified%20behavior.

*Confluence*. STR51-CPP. Do not attempt to create a std::string from a null pointer - SEI CERT C++ Coding Standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR51-CPP.+Do+not+attempt+to+create+a+std%3A%3Astring+from+a+null+pointer.

*Confluence*. MEM56-CPP. Do not store an already-owned pointer value in an unrelated smart pointer - SEI CERT C++ Coding Standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM56-CPP.+Do+not+store+an+already-owned+pointer+value+in+an+unrelated+smart+pointer.

*Confluence*. MEM50-CPP. Do not access freed memory - SEI CERT C++ Coding Standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory.

*Confluence*. DCL03-C. Use a static assertion to test the value of a constant expression - SEI CERT C Coding Standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression.

*Confluence*. DCL57-CPP. Do not let exceptions escape from destructors or deallocation functions - SEI CERT C++ Coding Standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL57-CPP.+Do+not+let+exceptions+escape+from+destructors+or+deallocation+functions#:~:text=Do%20not%20let%20exceptions%20escape%20from%20destructors%20or%20deallocation%20functions&text=If%20a%20deallocation%20function%20terminates,never%20have%20well%2Ddefined%20behavior.

*Confluence*. MEM01-C. Store a new value in pointers immediately after free() - SEI CERT C Coding Standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152148.

*Confluence*. FIO01-C. Be careful using functions that use file names for identification - SEI CERT C Coding Standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/display/c/FIO01-C.+Be+careful+using+functions+that+use+file+names+for+identification#:~:text=Be%20careful%20using%20functions%20that%20use%20file%20names%20for%20identification&text=Many%20file%2Drelated%20security%20vulnerabilities,of%20the%20file%20object%20itself.

*Confluence*. ENV01-C. Do not make assumptions about the size of an environment variable - SEI CERT C Coding Standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/display/c/ENV01-C.+Do+not+make+assumptions+about+the+size+of+an+environment+variable#:~:text=Do%20not%20make%20any%20assumptions,allocated%20(see%20STR31%2DC.