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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 13](#_Toc52464063)

[Coding Standard 6 16](#_Toc52464064)

[Coding Standard 7 18](#_Toc52464065)

[Coding Standard 8 21](#_Toc52464066)

[Coding Standard 9 23](#_Toc52464067)

[Coding Standard 10 25](#_Toc52464068)

[Defense-in-Depth Illustration 28](#_Toc52464069)

[Project One 28](#_Toc52464070)

[1. Revise the C/C++ Standards 28](#_Toc52464071)

[2. Risk Assessment 28](#_Toc52464072)

[3. Automated Detection 28](#_Toc52464073)

[4. Automation 28](#_Toc52464074)

[5. Summary of Risk Assessments 29](#_Toc52464075)

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

[7. Map the Principles 31](#_Toc52464077)

[Audit Controls and Management 32](#_Toc52464078)

[Enforcement 32](#_Toc52464079)

[Exceptions Process 32](#_Toc52464080)

[Distribution 33](#_Toc52464081)

[Policy Change Control 33](#_Toc52464082)

[Policy Version History 33](#_Toc52464083)

[Appendix A Lookups 33](#_Toc52464084)

[Approved C/C++ Language Acronyms 33](#_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 | Validate input data can check source data's accuracy and quality before processing it into the application's function. This can safeguard systems from malicious attacks such as SQL injections while ensuring that the data entered matches the expected format. When implementing validating input data, it can also enforce limitations to have acceptable ranges or formats to confirm the data's acceptance. |
| 1. Heed Compiler Warnings | Heed compiler warnings generate warnings that there is a potential issue with the code that will lead the program to function differently than intended, indicating security issues within the code. Using heed compiler warnings can advise early bug detection and security flaws that can mitigate the program to susceptible attacks. Heed compiler warnings can also identify codes that may cause future problems, such as if the code is poorly written or insufficient. Addressing warnings at an early phase can improve the overall quality and maintainability of the code. |
| 1. Architect and Design for Security Policies | Architects and designs for security policies should be implemented at the beginning of a system or process design. The purpose is to mitigate potential threats and attacks during the design phase while aligning with security requirements. Not only is it cheaper to fix a vulnerability that is identified before the later stages of development or after deployment, but it can also enhance the system’s reliability and improve compliance with industry and security standards. |
| 1. Keep It Simple | The keep it simple principle emphasizes code should have simplicity and to avoid unnecessary complexity for systems to be more effective, efficient, and user-friendly. To keep it simple, one should focus on the core elements to obtain the desired outcome and to eliminate redundant steps. Adapting to this principle can ensure maintainability and clarity for all developers involved in the process. |
| 1. Default Deny | Default deny automatically assumes that all access or actions are denied, and only specific authorized users, devices, applications, or actions are granted permission. When assuming all access or actions are denied, there is a smaller risk of vulnerabilities occurring and ensures more control over who is designated to access management. One of the best practices to implement default deny is by using an operating system for user accounts with limited permissions by default and only providing more access when needed. |
| 1. Adhere to the Principle of Least Privilege | Adhering to the principle of least privilege (PoLP) means providing the minimum of privileges to the lowest-level account to perform the given tasks. PoLP can not only minimize errors but also reduce the attack surface. Suppose an unauthorized user has access to a limited privileged account. This means the attacker will cause less damage compared to an account with full access. Utilizing PoLP helps to confine any potential issues to a smaller area. |
| 1. Sanitize Data Sent to Other Systems | Sanitized data sent to other systems cleans and validates data before transferring it to another system. This will prevent issues like SQL injection and data corruption from transferring from one system to another. The best practices for sanitizing data are to ensure data validation, encoding, and removal are used to identify potential threats, improve security, and enhance data integrity. |
| 1. Practice Defense in Depth | Defense in Depth (DiD) means creating multiple layers of security control to protect the system, data, and users. The layers of security can include physical, network, system, and data security. If one layer is breached, the attacker will not have complete access because of the additional layers being utilized. DiD helps enhance security and reduce the overall risk of a successful attack. It is always best to continuously monitor suspicious activity to identify and address possible threats. |
| 1. Use Effective Quality Assurance Techniques | Effective quality assurance techniques ensure the quality of a product or service by identifying, preventing, and fixing defects throughout its lifecycle. Ways to ensure quality assurance include planning, continuous testing, test automation, and usability testing. Applying these techniques can help catch problems early and improve overall quality, making it more reliable and user-friendly. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard means building secure software that promotes best practices within the industry and security standards. Standards reduce the risk of vulnerabilities while raising awareness by showcasing knowledge of security risks. Good coding standards can be defined as comprehensive, displaying concrete examples of good and bad practices. |

### 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** | **Obey the one-definition rule (ODR)** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | ODR means only one definition of a non-inline function or variable can be used across all units. ODR manages how named objects are defined and used in a program to ensure predictable behavior after linking across files. Using ODR helps to facilitate code organization and maintenance by enabling definitions to belong to different files. |

| **Noncompliant Code** |
| --- |
| The noncompliant code below defines S as a struct and a class. Since S has two definitions, it violates the ODR coding standard because the program will not know which definition to use, causing undefined behavior such as crashes or unexpected results. |
| // a.cpp  **struct** S {  **int** a;  };    // b.cpp  **class** S {  **public**:  **int** a;  }; |

| **Compliant Code** |
| --- |
| In the compliant solution below, a header file was used to enable the class S with an integer a to be used in both files. While S is defined as one definition, it also promotes code reusability and maintainability to keep the class definition centralized. |
| // S.h  **struct** S {  **int** a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):**   * 2. Heed Compiler Warnings:  Maps ODR because it can help reduce compiler warnings from inconsistent code definitions. * 3. Architect and Design for Security Policies:  Map ODR because it ensures a clear and consistent codebase that prevents uncertainty and potential security vulnerabilities. * 4. Keep It Simple: Since ODR reduces code complexity, it aligns with the principle of enforcing that each element should have one definition. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | P3 (Low) | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **type-compatibility** **definition-duplicate** **undefined-extern** **undefined-extern-pure-virtual** **external-file-spreading** **type-file-spreading** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL60** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.STRUCT.DEF.FDH** **LANG.STRUCT.DEF.ODH** | Function defined in header file Object defined in header file |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++1067, C++1509, C++1510** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **286 S, 287 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-DCL60-a** | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: DCL60-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl60cpp.html) | Checks for inline constraints not respected (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **type-compatibility** **definition-duplicate** **undefined-extern** **undefined-extern-pure-virtual** **external-file-spreading** **type-file-spreading** | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Value-returning functions must return a value from all exit paths** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | A value-returning function must return a value from all code paths or else it will lead to an undefined behavior. Return-value functions are expected to provide a value of the declared return type such as normal paths, code blocks, and try-catch blocks. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code below, the programmer fails to return the positive input value. Without inputting the positive input value, the code does not handle the non-negative number, meaning not all code paths will return a value. |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;    }  } |

| **Compliant Code** |
| --- |
| In the compliant solution below, the programmer includes the positive input return, enabling the code to correctly calculate absolute values for negative numbers. |
| int absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;    }  **return** a;  } |

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

| **Principles(s):**   * 2. Heed Compiler Warnings:  The principle maps the standard code because the principle highlights potential issues and can reduce warnings if the value is missing on the declared return type. * 4. Keep it simple:  A code with clear and consistent return types is not just readable and easier to maintain, it also reassures you that you're in sync with the best practices, aligning with both the coding standard and principle. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **return-implicit** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MSC52** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | **-Wreturn-type** | Does not catch all instances of this rule, such as *function-try-blocks* |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.STRUCT.MRS** **LANG.STRUCT.NVNR** | Missing return statement Non-void noreturn, |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **DF2888** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | **FUNCRET.GEN**  **FUNCRET.IMPLICIT** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **2 D, 36 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MSC52-a** | All exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: MSC52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc52cpp.html) | Checks for missing return statements (rule partially covered) |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [S935](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-935) |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.31 | [V591](https://pvs-studio.com/en/docs/warnings/v591/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **return-implicit** | Fully checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Use valid references, pointers, and iterators to reference elements of a basic\_string** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | String can store a sequence of characters that can provide iterators like other containers. However, string references, pointers, and iterators can become invalidated when passing a non-const reference or calling a non-const member function. If the string is accessed using invalidated references, the program’s outcome can be unpredictable and provide unfavored results. |

| **Noncompliant Code** |
| --- |
| In the code below, the code attempts to create a new string but copies the input and replaces the semicolon with spaces. Unfortunately, insert() causes an undefined behavior because it invalidates loc, pointing with the string. This means that every insertion in the loop manipulates email to make loc point at the wrong location after the first insertion causing an unexpected behavior. |
| #include <string>    **void** f(**const** std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();  **for** (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

| **Compliant Code** |
| --- |
| In the code below, insert() is assigned back to loc. When assigning the return value or insert() back to loc, the code ensures that loc points to the correct direction after each insertion. Having a loc point in the proper direction means it remains valid throughout the loop, enabling the code to insert characters with predictable behavior. |
| #include <string>    **void** f(**const** std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();  **for** (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      loc = email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

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

| **Principles(s):**   * 1. Validate Input Data:  The coding standard is a form of the principle because it ensures the validity of references, pointers, and iterators. Incorrect values result in out-of-bounce access, which can lead to weaknesses or security breaches. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **ALLOC.UAF** | Use After Free |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **DF4746, DF4747, DF4748, DF4749** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-STR52-a** | Use valid references, pointers, and iterators to reference elements of a basic\_string |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C++: STR52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr52cpp.html) | Checks for use of invalid string iterator (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Do not use deprecated or obsolescent functions** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | It is best to avoid using deprecated or obsolescent functions because of security risks, compatibility issues, and maintainability issues. Deprecated functions are typically marked for removal due to security concerns or better alternatives. Obsolescent functions are less desirable because there are better alternatives within the C standard documentation or compiler warnings. |

| **Noncompliant Code** |
| --- |
| The code below uses obsolescent functions strcat and strcpy in C. Unfortunately, both functions can lead to buffer overflow because they do not perform bounds checking. If the resulting string exceeds the size of the buffer, overflow would occur, corrupting the memory and leading to undefined behavior. |
| #include <string.h>  #include <stdio.h>    **enum** { BUFSIZE = 32 };  **void** complain(**const** **char** \*msg) {    **static** **const** **char** prefix[] = "Error: ";  **static** **const** **char** suffix[] = "\n";  **char** buf[BUFSIZE];      strcpy(buf, prefix);    strcat(buf, msg);    strcat(buf, suffix);    fputs(buf, stderr);  } |

| **Compliant Code** |
| --- |
| The code below shows that functions strcat and strcpy were replaced with better alternatives, such as strcpy\_s and strcat\_s. The other options mitigate potential overflow risks by taking the buffer size as an argument, enabling them to check if overflow would occur. If overflow is detected, the program is terminated, preventing undefined behavior from occurring. It is compliant because the alternatives can avert buffer overflow from happening and because it is safer to use than the functions strcat and strcpy. |
| #define \_\_STDC\_WANT\_LIB\_EXT1\_\_  #include <string.h>  #include <stdio.h>    **enum** { BUFFERSIZE = 256 };    **void** complain(**const** **char** \*msg) {  **static** **const** **char** prefix[] = "Error: ";  **static** **const** **char** suffix[] = "\n";  **char** buf[BUFFERSIZE];      strcpy\_s(buf, BUFFERSIZE, prefix);    strcat\_s(buf, BUFFERSIZE, msg);    strcat\_s(buf, BUFFERSIZE, suffix);  **fputs**(buf, stderr);  } |

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

| **Principles(s):**   * 2. Heed Compiler Warnings:  Using a deprecated or obsolescent function generates compiler warnings indicating issues with the code. Avoiding deprecated or obsolescent functions means it will align with the principle. * 4. Keep It Simple:  Using nondeprecated or obsolescent functions makes the code more reliable, easier to maintain, and clearer, mapping out the principle of keeping it simple. * 9. Use Effective Quality Assurance Techniques:  Using nondeprecated or obsolescent functions can align with the principle of using better, more secure codes that improve the overall quality of the code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probably | Medium | P12 (High) | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **stdlib-use-ato**  **stdlib-macro-ato**  **stdlib-use-atoll**  **stdlib-macro-atoll** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MSC24** | Fully implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **BADFUNC.\***  **(customization)** | A number of CodeSonar's "Use of \*" checks are for deprecated/obsolescent functions CodeSonar also provides a mechanism for users to create custom checks for uses of specified functions |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.MSC34** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-MSC24-a** **CERT\_C-MSC24-b** **CERT\_C-MSC24-c** **CERT\_C-MSC24-d** | The library functions atof, atoi and atol from library stdlib.h shall not be used The 'getenv()' function from the 'stdlib.h' or 'cstdlib' library shall not be used Avoid using unsafe string functions which may cause buffer overflows Don't use unsafe C functions that do write to range-unchecked buffers |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **586** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rec. MSC24-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.msc24c.html) | Checks for use of obsolete standard function (rec. fully covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.31 | [**V513**](https://pvs-studio.com/en/docs/warnings/v513/), [**V2001**](https://pvs-studio.com/en/docs/warnings/v2001/), [**V2002**](https://pvs-studio.com/en/docs/warnings/v2002/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 | **stdlib-use-ato**  **stdlib-macro-ato**  **stdlib-use-atoll**  **stdlib-macro-atoll** | Partially checked |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Detect and handle memory allocation errors** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | When requesting memory using a new operator, the system may not have enough resources to fulfill the request, leading to unexpected behavior. Throwing new and new[] operators adds additional ways to allocate memory and checks with try-catch and nullptr to prevent program crashes. Also, for best practice, the system should not be abruptly terminated due to memory allocation failures and should use alternative options or notifications. |

| **Noncompliant Code** |
| --- |
| The code below is considered noncompliant because an unexpected program termination can occur from not checking allocation results. The code uses ::operator new[](std::size\_t), but the allocation will fail if there is not enough memory. Also, another is the issue with noexcept. Noexcept in this scenario assumes there will not be any thrown exceptions and can cause a false promise to the caller, leading to either an unexpected behavior and the program will crash or a compiler issue. |
| #include <cstring>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** **int**[size];    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

| **Compliant Code** |
| --- |
| The code below is compliant because of it’s proper handling of using new(std::nothrow) to allocate memory and returns a nullptr instead of throwing an exception. Using nullptr can indicate allocation failures before using the pointer, preventing crashes or undefined behavior. |
| #include <cstring>  #include <new>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** (std::**nothrow**) **int**[size];  **if** (!copy) {      // Handle error  **return**;    }    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

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

| **Principles(s):**   * 1. Validate Input Data:  The principle maps the coding standard because detecting and handling memory allocation ensures input data is validated and prevents vulnerabilities such as buffer overflows and memory leaks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Coverity) | 7.5 | **CHECKED\_RETURN** | Finds inconsistencies in how function call return values are handled |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++3225, C++3226, C++3227, C++3228, C++3229, C++4632** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.2 | **NPD.CHECK.CALL.MIGHT** **NPD.CHECK.CALL.MUST** **NPD.CHECK.MIGHT** **NPD.CHECK.MUST** **NPD.CONST.CALL** **NPD.CONST.DEREF** **NPD.FUNC.CALL.MIGHT** **NPD.FUNC.CALL.MUST** **NPD.FUNC.MIGHT** **NPD.FUNC.MUST** **NPD.GEN.CALL.MIGHT** **NPD.GEN.CALL.MUST** **NPD.GEN.MIGHT** **NPD.GEN.MUST** **RNPD.CALL** **RNPD.DEREF** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **45 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MEM52-a** **CERT\_CPP-MEM52-b** | Check the return value of new Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: MEM52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem52cpp.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.31 | [**V522**](https://pvs-studio.com/en/docs/warnings/v522/)**,**[**V668**](https://pvs-studio.com/en/docs/warnings/v668/) |  |

#### 

#### Coding Standard 6

| **Coding Standard** | **Label** | **Understand the termination behavior of assert() and abort()** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Assert() can be helpful for debugging and catching unexpected conditions, but abrupt termination can prevent crucial cleanup actions to occur. Using static assertions for compile-time checks or custom runtime checks can assist with proper error handling. |

| **Noncompliant Code** |
| --- |
| The noncompliant code below shows a program termination and error handling of using atexit and improper use of assert(). Because the If statement checks the return value of atexit, a registration fail can occur. If this happens and the error is not handled, the cleanup function will not be called leading to recourse leaks, inconsistent program state, or unexpected behavior. |
| **void** cleanup(**void**) {    /\* Delete temporary files, restore consistent state, etc. \*/  }    **int** main(**void**) {  **if** (**atexit**(cleanup) != 0) {      /\* Handle error \*/    }      /\* ... \*/    **assert**(/\* Something bad didn't happen \*/);      /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The compliant code below shows an example of proper handling. The assert call is replaced with an If statement to check if something wrong happens. If the condition is true, then it calls the code exit(EXIT\_FAILURE). The exit can control the program’s termination. It can also register the cleanup function, successfully enabling the cleanup to execute before the program exits. |
| **void** cleanup(**void**) {    /\* Delete temporary files, restore consistent state, etc. \*/  }    **int** main(**void**) {  **if** (**atexit**(cleanup) != 0) {      /\* Handle error \*/    }      /\* ... \*/    **if** (/\* Something bad happened \*/) {  **exit**(EXIT\_FAILURE);    }    /\* ... \*/  } |

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

| **Principles(s):**   * 9. Use Effective Quality Assurance Techniques:  The principle maps the coding standard because planning and designing during the development phase will help to understand the behavior of assert() and abort(). While assert() and abort() can be helpful, using them wrong can lead to unexpected behaviors and an inconsistent program state. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-ERR06-a** | Do not use assertions |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **586** | Fully supported |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Catch exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | The coding standard for catch exceptions is to ensure proper error recovery. When an exception is thrown, a catch handler is involved. However, suppose the thrown object is a derived class, and the base class has a non-trivial destructor or copy constructor. In that case, slicing can occur, causing improper error handling. To avoid these situations, using catch exceptions such as const T& or T& can ensure the object is copied and derived from the class information. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code, slicing occurred because the class S is derived from std::exception when the catch block tries copying object e using copy initialization. As a result, only parts of the base class are copied, causing the custom message within the class S information to be lost. |
| #include <exception>  #include <iostream>    **struct** S : std::exception {  **const** **char** \*what() **const** noexcept override {  **return** "My custom exception";    }  };    **void** f() {  **try** {  **throw** S();    } **catch** (std::exception e) {      std::cout << e.what() << std::endl;    }  } |

| **Compliant Code** |
| --- |
| In the compliant code, the catch block’s exception declaration is changed to std::exception (std::exception& e).  This alteration creates a reference to the exception object that is thrown. Now that the entire thrown object is referenced instead of copied, avoid the slicing to ensure the S object behaves correctly with the catch block. |
| #include <exception>  #include <iostream>    **struct** S : std::exception {  **const** **char** \*what() **const** noexcept override {  **return** "My custom exception";    }  };    **void** f() {  **try** {  **throw** S();    } **catch** (std::exception &e) {      std::cout << e.what() << std::endl;    }  } |

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

| **Principles(s):**   * 9. Use Effective Quality Assurance Techniques:  Having proper exception handling can ensure its effectiveness for testing and debugging to detect vulnerabilities. The principle maps out the coding standing because it can prevent unexpected program termination and potential vulnerabilities, showcasing code quality. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 (Low) | P3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | catch-class-by-value | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-ERR61 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-err61-cpp | Checked by clang-tidy; also checks for VOID ERR09-CPP. Throw anonymous temporaries by default |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | LANG.STRUCT.EXCP.CATCH  LANG.STRUCT.EXCP.THROW | Use of catch  Use of throw |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | C++4031 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | MISRA.CATCH.BY\_VALUE |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 455 S | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-ERR61-a CERT\_CPP-ERR61-b | A class type exception shall always be caught by reference Throw by value, catch by reference |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: ERR61-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr61cpp.html) | Checks for exception object initialized by copy in catch statement (rule fully covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.31 | [**V746**](https://pvs-studio.com/en/docs/warnings/v746/)**,**[**V816**](https://pvs-studio.com/en/docs/warnings/v816/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **catch-class-by-value** | Fully checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S1044**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-1044) |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not copy a FILE object** |
| --- | --- | --- |
| Input Output (FIO) | [STD-008-CPP] | When copying a FILE object, there may not be enough memory to create a functional replica. Unexpected behaviors can occur because the operation may not interact with the intended file if it is in a different state than its original form. Also, copying FILE objects can lead to data corruption if operations are performed on the wrong file or with an inconsistent state. |

| **Noncompliant Code** |
| --- |
| Below, an access violation occurred because copying stdout might create an unusable copy. Also, the copy of my\_stdout may not have permissions or point to the same underlying source as the original. In this scenario, the access violation error can occur because of the copied object's limitations. |
| #include <stdio.h>    **int** main(**void**) {  **FILE** my\_stdout = \*stdout;  **if** (**fputs**("Hello, World!\n", &my\_stdout) == EOF) {      /\* Handle error \*/    }  **return** 0;  } |

| **Compliant Code** |
| --- |
| Once the stdout is assigned to the variable my\_stdout of type File\*, it avoids directly copying the FILE object. It reduces the risk of a denial-of-service attack. |
| #include <stdio.h>    **int** main(**void**) {  **FILE** \*my\_stdout = stdout;  **if** (**fputs**("Hello, World!\n", my\_stdout) == EOF) {      /\* Handle error \*/    }  **return** 0;  } |

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

| **Principles(s):**   * 9. Use Effective Quality Assurance Techniques:  The principle maps out the coding standard because proper filing handling can make a code more effective during testing and analysis. With efficient coding, the code is more accessible and readable to maintain, reducing errors and improving performance overall. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | P4 (Low) | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **file-dereference** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-FIO38** | Fully implemented |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-non-copyable-objects | Checked with clang-tidy |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect simple violations of this rule |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **MISRA C 2012 Rule 22.5** | Partially implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.2 | **C1485, C5028**  **C++3113, C++3114** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.2 | **MISRA.FILE\_PTR.DEREF.2012** **MISRA.FILE\_PTR.DEREF.CAST.2012** **MISRA.FILE\_PTR.DEREF.INDIRECT.2012** **MISRA.FILE\_PTR.DEREF.RETURN.2012** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **591 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-FIO38-a** | A pointer to a FILE object shall not be dereferenced |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **9047** | Partially supported: reports when a FILE pointer is dereferenced |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rule FIO38-C](https://www.mathworks.com/help/bugfinder/ref/certcrulefio38c.html) | Checks for misuse of a FILE object (rule fully covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 | **file-dereference** | Partially checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do not modify the standard namespaces** |
| --- | --- | --- |
| Declarations and Initialization (DCL) | [STD-009-CPP] | Namespace creates separate "zones" for code elements to reduce the chances of duplicate naming conflicts. It's best to avoid modifying standard namespaces like std (the standard library) or posix (another standard library) because it can lead to namespace pollution, naming conflicts, or even unintended behaviors. Using user-defined types can be designed best to meet specific needs in the program. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code below, the declaration of x is added to the namespace std, resulting in undefined behavior. An undefined behavior can lead to the program failing to compile, executing incorrectly, crashing, or generating incorrect results. |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| In the compliant solution below, the declaration of x is placed into a namespace to prevent collision with other identifiers. Therefore, it provides a clear and organized way to manage code variables and functions to improve readability and maintainability. |
| namespace nonstd {  int x;  } |

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

| **Principles(s):**   * 4. Keep it simple: The principle maps out the coding standard by ensuring that modifications to the standard namespace make the code predictable and meet the needs of the program. When adhering to the guide, the code’s readability is enhanced and reduces conflicts with other libraries. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL58** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.STRUCT.DECL.SNM** | Modification of Standard Namespaces |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | C++3180, C++3181, C++3182 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | CERT.DCL.STD\_NS\_MODIFIED |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-DCL58-a | Do not modify the standard namespaces 'std' and 'posix' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: DCL58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl58cpp.html) | Checks for modification of standard namespaces (rule fully covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.31 | [**V1061**](https://pvs-studio.com/en/docs/warnings/v1061/) |  |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3470**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3470) |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do not access an object outside of its lifetime** |
| --- | --- | --- |
| Expressions (EXP) | [STD-010-CPP] | Every object has a lifetime. The lifetime starts when sufficient memory is allocated to the object and ends when the memory allocation for the object is freed or reused. Using an object outside the lifetime expectancy can result in undefined behavior. If the object’s lifetime needs to be maintained manually, it's best to use smart pointers to handle the memory allocation and deallocation automatically. |

| **Noncompliant Code** |
| --- |
| The noncompliant code shows the pointer s with the non-static member mem\_fn(). The code will lead to undefine behavior because the pointer is not pointing to the valid S object, making it point to nowhere and causing the undefine behavior. |
| **struct** S {  **void** mem\_fn();  };    **void** f() {    S \*s;    s->mem\_fn();  } |

| **Compliant Code** |
| --- |
| The memory allocation is properly handled since the pointers now point to the valid S object before calling the member function. Since the object’s lifetime starts with the memory allocation and initialization, the member function call happens within the valid lifetime, mitigating the undefined behavior from occurring. |
| **struct** S {  **void** mem\_fn();  };    **void** f() {    S \*s = **new** S;    s->mem\_fn();  **delete** s;  } |

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

| **Principles(s):**   * 1. Validate Input Data:  The principle maps out the coding standard because the objects are only accessed within their valid lifetime, refraining from any access to invalid memory locations. Pointers that only call member functions are a form of input validation, and the data is accurate without any unexpected behaviors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probably | High | P6 (Medium) | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **return-reference-local** **dangling\_pointer\_use** | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | **-Wdangling-initializer-list** | Catches some lifetime issues related to incorrect use of std::initializer\_list<> |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **IO.UAC** **ALLOC.UAF** | Use after close Use after free |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++4003, C++4026**  **DF2812, DF2813, DF2814, DF2930, DF2931, DF2932, DF2933, DF2934,** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | **CL.FFM.ASSIGN** **CL.FFM.COPY** **LOCRET.ARG** **LOCRET.GLOB** **LOCRET.RET** **UFM.DEREF.MIGHT** **UFM.DEREF.MUST** **UFM.FFM.MIGHT** **UFM.FFM.MUST** **UFM.RETURN.MIGHT** **UFM.RETURN.MUST** **UFM.USE.MIGHT** **UFM.USE.MUST** **UNINIT.HEAP.MIGHT** **UNINIT.HEAP.MUST** **UNINIT.STACK.ARRAY.MIGHT** **UNINIT.STACK.ARRAY.MUST** **UNINIT.STACK.ARRAY.PARTIAL.MUST** **UNINIT.STACK.MIGHT** **UNINIT.STACK.MUST** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **42 D, 53 D, 77 D, 1 J, 71 S, 565 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-EXP54-a** **CERT\_CPP-EXP54-b** **CERT\_CPP-EXP54-c** | Do not use resources that have been freed The address of an object with automatic storage shall not be returned from a function The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: EXP54-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp54cpp.html) | Checks for:   * Non-initialized variable or pointer * Use of previously freed pointer * Pointer or reference to stack variable leaving scope * Accessing object with temporary lifetime * Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.31 | [**V758**](https://pvs-studio.com/en/docs/warnings/v758/)**,**[**V1041**](https://pvs-studio.com/en/docs/warnings/v1041/)**,**[**V1099**](https://pvs-studio.com/en/docs/warnings/v1099/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **return-reference-local** | Partially checked |

### 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.

While DevOps prioritizes speed and efficiency, DevSecOps can help add layers of security throughout the entire developmental process by incorporating security from the outset and creating a more robust and secure SDLC. One way to create a more secure application is to use automation throughout the DevSecOps pipeline to enforce security standards and compliance.

In the pre-production phase, automation can be helpful because it can accelerate development and deployment and enhance security. During the assess and plan phase, automation can examine the current DevOps process, such as workflow, repetitive tasks, or bottlenecks, to identify what needs to be addressed. Having an organized and solid foundation with automation can streamline tasks, reduce errors, and improve security. In the design phase, automation can generate security requirements and system components. Incorporating automation during this phase, instead of leaving security as an afterthought, can create a more resilient system architecture, making the system more efficient and facilitating better risk management. Automation within the build phase can implement static code analysis to identify coding errors and security vulnerabilities while being integrated into the CI/CD pipeline to test and validate each stage within the building process. Once deployed, the automation could check for common exploits to quickly identify and solve problems early in the development cycle, mitigating vulnerabilities that can occur before production. Lastly, within the verify and test phase, automation can be helpful for test cases that can review functional and security testing. During this phase, automation can improve how the application should behave while assessing the system’s vulnerability to enhance the overall quality and security of the software. By incorporating automation in all pre-production phases, security and overall efficiency can be improved throughout SDLC.

While in the production phase, automation can benefit DevSecOps because it can ensure ongoing security and reliability for the application. During the transition and health phase, automation can be established to perform benchmarks to assess the system’s performance over time while identifying anomalies that may occur. Automation can collect response times, compare performances through a baseline, and notify teams of abnormal behaviors, improving the reliability of the application. While in the monitor and detect phase, automation can monitor the infrastructure to measure bottlenecks and other resource constraints. Utilizing automation would optimize the system’s performance and quickly respond to incidents to improve the application. Since the response phase focuses on reacting to incidents, automation can develop automated playbooks to handle common incidents like security breaches. Utilizing automation can accelerate and improve the rapid response time during production. Lastly, in the maintain and stabilize phase that keeps the system running optimally, automation can be used to scan for vulnerabilities and generate lists of required patches for various system types. Automation is beneficial because it could reduce the time and effort needed to manage the patch while improving system security. Automation in the production phase can significantly reduce downtime and improve security and optimization utilization.

By incorporating automation at each stage, security, efficiency, and reliability can be enhanced. With automation in both pre-production and production, security is not a second thought but a fundamental asset of the development and operation processes for the existing DevOps framework.

### 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 | High | Unlikely | High | P3 (Low) | 3 |
| STD-002-CPP | Medium | Probable | Medium | P8 (Medium) | 2 |
| STD-003-CPP | High | Probable | High | P6 (Medium) | 2 |
| STD-004-CPP | High | Probable | Medium | P12 (High) | 1 |
| STD-005-CPP | High | Likely | Medium | P18 (High) | 1 |
| STD-006-CPP | Medium | Unlikely | Medium | P4 (Low) | 3 |
| STD-007-CPP | Low | Unlikely | Low | P3 (Low) | 3 |
| STD-008-CPP | Low | Probable | Medium | P4 (Low) | 3 |
| STD-009-CPP | High | Unlikely | Medium | P6 (Medium) | 2 |
| STD-010-CPP | High | Probable | High | P6 (Medium) | 2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest refers to the data that is protected when it is stored and converted into an unreadable format to prevent unauthorized users from getting access. Encryption at rest uses algorithms applied to the data before it is stored. If the data needs to be accessed, the data can be decrypted using a decryption key. The policy applies because sensitive data should be at rest to mitigate potential data breaches or other exposures. If the data were taken, the information would still be safe because unauthorized users cannot read the data without the key. An example of encryption at rest would be in finance industries with sensitive client data that must be protected to meet regulatory compliances. |
| Encryption in flight | Encryption in flight protects sensitive data while being transmitted from one network to another, ensuring unauthorized users cannot intercept or read the data. Like encryption at rest, when the encryption is in flight, an algorithm is used on the data before it is transmitted. However, the receiver can only read the sensitive data if they have the decryption key. The policy applies because if an unauthorized user steals the data, the information is still safe since the user cannot read the data without the decryption key. An example of when to use encryption in flight can be hospitals that transfer client information from one hospital to another, securing personal information. |
| Encryption in use | Encryption in use refers to the encryption of data as it is being processed within the system. With specialized hardware or software encryptions, the data can be protected while being used, making it secure at all times. The policy applies because it provides the highest level of protection against advanced persistent threats and safeguards sensitive data from unauthorized users in both outside and insider threats. An example of encryption in use can be government or military organizations that handle classified information. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of confirming the identity of a user or a system to ensure that only authorized individuals can access the system's resources. Authentication can help prevent identity theft, maintain accessed data records, and secure networks. Authentication can be verified by combining passwords, security tokens, biometric data, such as fingerprints or facial recognition, and multi-factor authentication. An example of authentication is a student who logs into their student portal using a unique username and password to prove their identity in the system.  The policy applies when:   * User logins: Every system user should be authenticated to prevent unauthorized users from accessing the system. * Changes to the database: Data integrity must be properly executed when a database is modified. When a user is authenticated, it ensures that the database remains accurate and reliable, highlighting the importance of authentication in maintaining data quality. * Addition of new users: Individuals adding new users should be authenticated to protect accounts, networks, and systems from unauthorized users. |
| Authorization | Authorization determines what access rights, such as permission and privileges, a user can have once authenticated to utilize for their roles and responsibilities. Authorization assigns specific permissions to a particular group or individual based on job roles, security clearance, etc. Ways to manage approval is possible by Access Control Lists (ACLs) that define specific permission for various resources, Role-Based Access Control (RBAC) access assigned based on a user’s role in an organization, and Policy-Based Access Control (PBAC) that uses policies to determine the most suited access permissions to grant. An example of authorization is a student who can view their grades and submit assignments. At the same time, the teacher has the same access as the students but can also grade assignments and manage class rosters. In contrast, an administrator has both student and teacher access and system settings. The policy applies because granting users’ permissions based on their needs can mitigate potential security breaches.  The policy applies when:   * User level of access: Authorization must be verified to ensure a user can perform tasks such as reading, writing, or deleting data. * Files accessed by users: Authorization can control the files users can access and the type of operation they can perform on the files. * Changes to the database: Authorization also controls who can modify the database to ensure data integrity. |
| Accounting | Accounting, a vital process, tracks user activity within the organization or system by recording information such as what resources were accessed, when, and by whom. It uses a log to monitor actions such as logging in or out, file access, and data modification. An example is the use of Key Performance Indicators (KPIs) for monitoring data storage efficiency and determining disk space utilization for log files or the average data growth rate. Accounting is important because it provides essential data such as behavior analysis, performance management, and auditing to identify security breaches or other unauthorized accesses.  The policy applies when:   * User logins: Timestamps, user IDs, and status should be logged to ensure the system is functioning correctly and to identify unauthorized users. * Changes to the database: Modification to the database should log who, when, and what was changed for security and compliance measures. * Addition of new users: Adding new users should be recorded because it provides modified information and can determine accountability and potential insider threats. * Files accessed by users: Logging files accessed by users can track patterns that assist with security and performance analysis. |

**\***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 | 07/21/2024 | Ten Core Security Principles & C/C++ Ten Coding Standards | Winnie Kwong | Mimi Tam |
| 3.0 | 08/11/2024 | C/C++ Ten Coding Standards (Threat Level & Automation Tools), Automation Summary, Summary of Risk Assessments, Encryption, and Triple-A Framework | Winnie Kwong |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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