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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | This principle emphasizes how essential it is to verify all data inputs to ensure firstly that they’re secure and then of course that they’re correct and complete. Most software vulnerabilities can be eliminated with proper input validation. |
| 1. Heed Compiler Warnings | Like most security principles, heeding compiler warnings is incredibly relevant during the development phase. It pertains to the importance of paying close attention to the warnings generated by the compiler during development. These can be uninitialized variables, type mismatches, etc. left unaddressed that can lead to vulnerabilities. Resolving these promptly can lead to more robust and stable applications. |
| 1. Architect and Design for Security Policies | Creating the architecture and design with security policies in mind is this principles proactive approach. A couple examples would be, designing with an access control policy to ensure only authorized users can access proprietary data or designing with a data encryption policy to protect data from authorized access. |
| 1. Keep It Simple | This principle is as simple as it sounds. Keeping it simple allows for system designs to enhance security. The very nature of a complex system is that with complexity, there is more room for errors, more room for vulnerabilities, and thus much more difficult to keep secure. This can be accomplished by minimizing unnecessary features, designing a minimalistic user interface, as well adopting the single responsibility principle. |
| 1. Default Deny | Deny by default is almost an addition of keeping it simple. This security approach is implemented in the early development phase and denies all users access to resources unless it is explicitly and purposefully granted. This reduces the attack surface greatly and ultimately creates a more secure environment. |
| 1. Adhere to the Principle of Least Privilege | This principle also has to do with granting access to users. To enhance overall security, this security measure mandates that all users only be granted the absolute minimum level of access that’s required to perform their tasks and responsibilities. This removes the risk of any accidental or malicious use of resources. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing, cleaning, and validating data is a necessary step to ensure malicious code like SQL injection or XSS attacks can be prevented from spreading. This helps industry organizations protect their internal systems as well as other systems they plan on interacting with. There’s various software’s that assist with this principle as well as best practices like certifying data destruction methods, maintaining a detailed audit trail, and training personnel. |
| 1. Practice Defense in Depth | This is the onion principle. In theory, by utilizing many layers of defense, you improve the security of your system. This way, if one layer fails, there are additional layers in place to continuously and further protect the system. This can be accomplished with various methods including firewalls, network segmentation, antivirus or anti-malware software, access controls, etc. |
| 1. Use Effective Quality Assurance Techniques | Incorporating rigorous quality assurance techniques highlights the importance of testing and validating processes throughout the development cycle. It helps to ensure the software is meeting security standards and protecting against potential threats. This can be accomplished through frequent code reviews, vulnerability assessments, automated testing, etc. By finding flaws early through quality assurance, it helps to improve overall performance of the software. |
| 1. Adopt a Secure Coding Standard | Another proactive principle that keeps security in mind from the ground up. This involves setting guidelines for writing secure code throughout the development process. By following these standards and guidelines developers can avoid the common vulnerabilities such as SQL injection, buffer overflow, etc. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Never qualify a reference type with const or volatile. |
| Source: [DCL52-CPP. Never qualify a reference type with const or volatile - SEI CERT C++ Coding Standard - Confluence](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL52-CPP.+Never+qualify+a+reference+type+with+const+or+volatile) | | |

| **Noncompliant Code** |
| --- |
| This noncompliant code example correctly declares *p* to be a reference to a *const-*qualified *char.* This means that *p* cannot be modified which can cause a compilation error when assigning results to *p.* |
| #include <iostream>  void f(char c) {  const char &p = c;  P = ‘p’;  Std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| This compliant solution removes the *const* qualifier. |
| #include <iostream>  void f(char c) {  char &p = c;  P = ‘p’;  Std::cout << c << std::endl;  } |

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

| **Principles(s):**  2. Heed Compiler Warnings: Qualifying a reference type with const or volatile can directly lead to compiler warnings.  4. Keep It Simple: Simplifying the code by avoiding const or volatile reference types will prevent confusion.  10. Adopt a Secure Coding Standard: Following this standard ensures our code adheres to best practices and adopts a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL52** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++0014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: DCL52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl52cpp.html) | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 |  | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options. |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3708**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3708) | [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Do not declare or define a reserved identifier. |
| Source: [DCL51-CPP. Do not declare or define a reserved identifier - SEI CERT C++ Coding Standard - Confluence](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL51-CPP.+Do+not+declare+or+define+a+reserved+identifier) | | |

| **Noncompliant Code** |
| --- |
| This noncompliant code declares a user-defined literal operator”” x. Literal suffixes without the underscore prefix are reserved for future library implementations. Utilizing these identifiers can lead to conflicts with standard library updates. Furthermore, this can be problematic in larger libraries and codebases as these conflicts will be harder to identify and ultimately resolve. |
| #include <cstddef>  unsigned int operator”” x(const char \*, std::size\_t); |

| **Compliant Code** |
| --- |
| This compliant solution names the user-defined literal operator”” \_x, which is not a reserved identifier. |
| #include <cstddef>  unsigned int operator”” \_x(const char \*, std::size\_t); |

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

| **Principles(s):**  2. Heed Compiler Warnings: Declaring or defining reserved identifiers can directly lead to compiler warnings.  4. Keep It Simple: Simplifying the code by avoiding reserved identifiers can mitigate risk of conflicts with library identifiers.  10. Adopt a Secure Coding Standard: Following this standard ensures our code adheres to best practices and adopts a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **reserved-identifier** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL51** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreserved-id-macro -Wuser-defined-literals | The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverything. This flag does not catch all instances of this rule, such as redefining reserved names. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.3p0 | **LANG.ID.NU.MK**  **LANG.STRUCT.DECL.RESERVED** | Macro name is C keyword  Declaration of reserved name |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++5003** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.4 | **MISRA.DEFINE.WRONGNAME** **MISRA.DEFINE.WRONGNAME.UNDERSCORE** **MISRA.UNDEF.WRONGNAME** **MISRA.UNDEF.WRONGNAME.UNDERSCORE** **MISRA.STDLIB.WRONGNAME** **MISRA.STDLIB.WRONGNAME.UNDERSCORE** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **86 S, 218 S, 219 S, 580 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-DCL51-a** **CERT\_CPP-DCL51-b** **CERT\_CPP-DCL51-c** **CERT\_CPP-DCL51-d** **CERT\_CPP-DCL51-e** **CERT\_CPP-DCL51-f** | Do not #define or #undef identifiers with names which start with underscore Do not redefine reserved words Do not #define nor #undef identifier 'defined' The names of standard library macros, objects and functions shall not be reused The names of standard library macros, objects and functions shall not be reused (C90) The names of standard library macros, objects and functions shall not be reused (C99) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: DCL51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl51cpp.html) | Checks for redefinitions of reserved identifiers (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.35 | [**V1059**](https://pvs-studio.com/en/docs/warnings/v1059/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **reserved-identifier** | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**978**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-978) |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Do not attempt to create a std::string from a null pointer. |
| Source: [STR51-CPP. Do not attempt to create a std::string from a null pointer - SEI CERT C++ Coding Standard - Confluence](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR51-CPP.+Do+not+attempt+to+create+a+std%3A%3Astring+from+a+null+pointer) | | |

| **Noncompliant Code** |
| --- |
| This noncompliant code creates a std::string object from the results of a call to std::getenv(). But because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior such as program crashes or the producing of incorrect results. When programs crash due to string creation from null pointers, the undefined behavior can introduce vulnerabilities and open windows for attackers to exploit. |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| This compliant solutions results from the call 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):**  1. Validate Input Data: Ensuring that pointers are not null before string creation, input data is validated.  2. Heed Compiler Warnings: Creating a string from a null pointer can directly lead to compiler warnings.  4. Keep It Simple: Simplifying the code by avoiding string creation from null pointers reduces the risk of errors.  10. Adopt a Secure Coding Standard: Following this standard ensures our code adheres to best practices and adopts a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **assert\_failure** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.3p0 | **LANG.MEM.NPD** | Null Pointer Dereference |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **DF4770, DF4771, DF4772, DF4773, DF4774** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.4 | **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** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-STR51-a** | Avoid null pointer dereferencing |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C++: STR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr51cpp.html) | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Guarantee that storage for strings has sufficient space for character data and the null terminator. |
| Source: [STR50-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator - SEI CERT C++ Coding Standard - Confluence](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator) | | |

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the unformatted input function std::basic\_istream<T>::read() to read an unformatted character array of 32 characters. However, the read() function does not guarantee that the string will be null terminated. |
| #include <fstream>  #include <string>    void f(std::istream &in) {    char buffer[32];    try {      in.read(buffer, sizeof(buffer));    } catch (std::ios\_base::failure &e) {      // Handle error    }      std::string str(buffer);    // ...  } |

| **Compliant Code** |
| --- |
| This compliant solution constructs the std::string object based on the number of characters read from the input stream instead of inserting a null terminator. Validating all user inputs form the input stream and properly escaping any special characters can prevent SQL injection attempts. |
| #include <fstream>  #include <string>    void f(std::istream &in) {    char buffer[32];    try {      in.read(buffer, sizeof(buffer));    } catch (std::ios\_base::failure &e) {      // Handle error    }    std::string str(buffer, in.gcount());    // ...  } |

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

| **Principles(s):**  1. Validate Input Data: Ensuring sufficient space prevents buffer overflows which occur when input data exceeds the spaces allocation.  2. Heed Compiler Warnings: Failure to allocate proper space can directly lead to compiler warnings and runtime errors.  4. Keep It Simple: Simplifying the code by properly managing string storage makes it easier to understand and maintain.  7. Sanitize Data Sent to Other Systems: Ensuring sufficient space and proper string termination prevents incomplete data being sent to other systems.  10. Adopt a Secure Coding Standard: Following this standard ensures our code adheres to best practices and adopts a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description** |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stream-input-char-array** | Partially checked + soundly supported |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++5216**  **DF2835, DF2836, DF2839,** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **NNTS.MIGHT** **NNTS.TAINTED** **NNTS.MUST** **SV.UNBOUND\_STRING\_INPUT.CIN** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation * Insufficient destination buffer size   Rule partially covered. |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **stream-input-char-array** | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3519**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3519) |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Do not access freed memory. |
| Source: [MEM50-CPP. Do not access freed memory - SEI CERT C++ Coding Standard - Confluence](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory) | | |

| **Noncompliant Code** |
| --- |
| This noncompliant code dereferences s after it has been deallocated. The vulnerability could be exploited if the access results in a write-after-free. This can lead to serious real world consequences such as program crashes, write-after-free vulnerability exploits, data corruption and leaked resources. |
| #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):**  1. Validate Input Data: Ensuring pointers aren’t used after memory is freed validates the integrity of our data.  2. Heed Compiler Warnings: Accessing freed memory can directly lead to compiler warnings and runtime errors.  4. Keep It Simple: Simplifying the code by avoiding the use of freed memory reduces risk.  8. Practice Defense in Depth: Ensuring the memory is accessed after being freed adds an additional layer of protection.  10. Adopt a Secure Coding Standard: Following this standard ensures our code adheres to best practices and adopts a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **dangling\_pointer\_use** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MEM50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.3p0 | **ALLOC.UAF** | Use after free |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++4303, C++4304** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.4 | **UFM.DEREF.MIGHT** **UFM.DEREF.MUST** **UFM.FFM.MIGHT** **UFM.FFM.MUST** **UFM.RETURN.MIGHT** **UFM.RETURN.MUST** **UFM.USE.MIGHT** **UFM.USE.MUST** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **483 S, 484 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.35 | [**V586**](https://pvs-studio.com/en/docs/warnings/v586/), [**V774**](https://pvs-studio.com/en/docs/warnings/v774/) |  |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/cplusplus/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. |
| Source: [DCL03-C. Use a static assertion to test the value of a constant expression - SEI CERT C Coding Standard - Confluence](https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression) | | |

| **Noncompliant Code** |
| --- |
| This noncompliant code utilizes the assert() macro to assert a property for the code to behave correctly. However, it needs to be placed in a function and executed. |
| #include <assert.h>    struct timer {    unsigned char MODE;    unsigned int DATA;    unsigned int COUNT;  };    int func(void) {    assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| In this compliant solution, status\_assert is utilized to allow incorrect assumptions to be diagnosed during compilation rather than in runtime. |
| #include <assert.h>    struct timer {    unsigned char MODE;    unsigned int DATA;    unsigned int COUNT;  };    static\_assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int),                "Structure must not have any padding"); |

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

| **Principles(s):**  2. Heed Compiler Warnings: Utilizing static assertions can help catch errors early during compile time which can assist in avoiding compiler warnings.  4. Keep It Simple: Simplifying the code by utilizing static assertions provides clear feedback during debugging.  10. Adopt a Secure Coding Standard: Following this standard ensures our code adheres to best practices and adopts a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.3p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/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 |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle all exceptions. |
| Source: [ERR51-CPP. Handle all exceptions - SEI CERT C++ Coding Standard - Confluence](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions) | | |

| **Noncompliant Code** |
| --- |
| This noncompliant code does not catch exceptions thrown by throwing\_func(). |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution the main entry point handles all exceptions. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    try {      f();    } catch (...) {      // Handle error    }  } |

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

| **Principles(s):**  2. Heed Compiler Warnings: Failing to handle exceptions can directly lead to complier warnings.  4. Keep It Simple: Simplifying the code by properly handling exceptions provides clear feedback during debugging.  8. Practice Defense in Depth: By handling all exceptions, an additional layer of protection is added against potential errors.  9. Use Effective Quality Assurance Techniques: Identifying and addressing potential issues early by handling all exceptions is a key aspect of effective quality assurance.  10. Adopt a Secure Coding Standard: Following this standard ensures our code adheres to best practices and adopts a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR51** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.3p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **MISRA.CATCH.ALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory Protection | STD-008-CPP | Detect and handle memory allocation errors. |
| Source: [MEM52-CPP. Detect and handle memory allocation errors - SEI CERT C++ Coding Standard - Confluence](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors) | | |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, an array is created and the results of the allocation are not checked. This can lead to undefined behavior as the null pointer will dereference in all subsequent operations. Ultimately the program could crash, data can be corrupted and vulnerabilities can therefore be exploited. |
| #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** |
| --- |
| In this compliant solution the error condition is handled appropriately when the returned pointer is nullptr. |
| #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: Effectively handling and detecting memory allocation errors is part of validating input data.  2. Heed Compiler Warnings: Failing to handle memory allocation errors can directly lead to complier warnings.  4. Keep It Simple: Simplifying the code by properly handling memory allocation errors provides clear feedback during debugging.  8. Practice Defense in Depth: By detecting and handling all memory allocation errors, an additional layer of protection is added against potential vulnerabilities.  9. Use Effective Quality Assurance Techniques: Identifying and addressing potential issues early by handling all memory allocation errors is a key aspect of effective quality assurance.  10. Adopt a Secure Coding Standard: Following this standard ensures our code adheres to best practices and adopts a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | 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) | 2024.4 | **C++3225, C++3226, C++3227, C++3228, C++3229, C++4632** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.4 | **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) | 2024.2 | **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) | 7.35 | [**V522**](https://pvs-studio.com/en/docs/warnings/v522/)**,**[**V668**](https://pvs-studio.com/en/docs/warnings/v668/) |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-009-CPP | Do not abruptly terminate the program. |
| Source: [ERR50-CPP. Do not abruptly terminate the program - SEI CERT C++ Coding Standard - Confluence](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR50-CPP.+Do+not+abruptly+terminate+the+program) | | |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, the call to f() may result in a call to std::terminate() as the throwing\_func() may throw an exception. This will lead to program instability and opens the possibility for resource leaks. And it also will lead to a poor user experience as repeated abrupt crashes will be frustrating. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    int main() {    if (0 != std::atexit(f)) {      // Handle error    }    // ...  } |

| **Compliant Code** |
| --- |
| In this compliant solution, f() handles all exceptions and does not rethrow. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.    try {      throwing\_func();    } catch (...) {      // Handle error    }  }    int main() {    if (0 != std::atexit(f)) {      // Handle error    }    // ...  } |

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

| **Principles(s):**  2. Heed Compiler Warnings: Abruptly terminating programs can directly lead to complier warnings.  4. Keep It Simple: Simplifying the code by avoiding abrupt termination language makes it more predictable.  8. Practice Defense in Depth: By ensuring the program doesn’t terminate abruptly, an additional layer of protection is added against leaked resources or other undefined behavior.  9. Use Effective Quality Assurance Techniques: Avoiding abrupt termination is a key aspect of effective quality assurance.  10. Adopt a Secure Coding Standard: Following this standard ensures our code adheres to best practices and adopts a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stdlib-use** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.3p0 | **BADFUNC.ABORT BADFUNC.EXIT** | Use of abort Use of exit |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++5014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.4 | **MISRA.TERMINATE** **CERT.ERR.ABRUPT\_TERM** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **122 S** | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-ERR50-a** **CERT\_CPP-ERR50-b** **CERT\_CPP-ERR50-c** **CERT\_CPP-ERR50-d** **CERT\_CPP-ERR50-e** **CERT\_CPP-ERR50-f** **CERT\_CPP-ERR50-g** **CERT\_CPP-ERR50-h** **CERT\_CPP-ERR50-i** **CERT\_CPP-ERR50-j** **CERT\_CPP-ERR50-k** **CERT\_CPP-ERR50-l** **CERT\_CPP-ERR50-m CERT\_CPP-ERR50-n** | The execution of a function registered with 'std::atexit()' or 'std::at\_quick\_exit()' should not exit via an exception Never allow an exception to be thrown from a destructor, deallocation, and swap Do not throw from within destructor There should be at least one exception handler to catch all otherwise unhandled exceptions An empty throw shall only be used in the compound-statement of a catch handler Exceptions shall be raised only after start-up and before termination of the program Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) Function called in global or namespace scope shall not throw unhandled exceptions Always catch exceptions Properly define exit handlers The 'abort()' function from the 'stdlib.h' or 'cstdlib' library shall not be used Avoid throwing exceptions from functions that are declared not to throw The 'quick\_exit()' and '\_Exit()' functions from the 'stdlib.h' or 'cstdlib' library shall not be used |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: ERR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr50cpp.html) | Checks for implicit call to terminate() function (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.35 | [**V667**](https://pvs-studio.com/en/docs/warnings/v667/)**,**[**V2014**](https://pvs-studio.com/en/docs/warnings/v2014/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **stdlib-use** | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S990**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-990) |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| String Correctness | STD-010-CPP | Range check element access. |
| Source: [STR53-CPP. Range check element access - SEI CERT C++ Coding Standard - Confluence](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access) | | |

| **Noncompliant Code** |
| --- |
| In this noncompliant code the value returned by get\_index() may be greater than the number of elements stored in the string. Which can lead to out of bounds access that are problematic and cause stability issues. Accessing out of bounds data can corrupt it, overwrite, and potentially crash the program. |
| #include <string>    extern std::size\_t get\_index();    void f() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| In this compliant solution a std::out\_of\_range exception is thrown if pos >= size(). |
| #include <stdexcept>  #include <string>  extern std::size\_t get\_index();    void f() {    std::string s("01234567");    try {      s.at(get\_index()) = '1';    } catch (std::out\_of\_range &) {      // Handle error    }  } |

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

| **Principles(s):**  1. Validate Input Data: A form of input validation is ensuring element access is within proper range.  2. Heed Compiler Warnings: Accessing elements outside the given range can directly lead to complier warnings.  4. Keep It Simple: Simplifying the code by properly checking element access makes it more readable and maintainable.  8. Practice Defense in Depth: Range checks add an additional layer of protection against vulnerabilities and potentially exploits.  9. Use Effective Quality Assurance Techniques: Ensuring elements are within proper ranges is a key aspect of effective quality assurance.  10. Adopt a Secure Coding Standard: Following this standard ensures our code adheres to best practices and adopts a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **assert\_failure** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.3p0 | **LANG.MEM.BO** **LANG.MEM.BU** **LANG.MEM.TBA** **LANG.MEM.TO** **LANG.MEM.TU** | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++3162, C++3163, C++3164, C++3165** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the valid range |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule 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.

The earliest we’re going to implement automation to ensure enforcement of and compliance to the standards defined in this policy are the design and build phases. We’ll start integrating early into the design phase to validate our architectural decisions against our standards. And during the design review is when we’ll begin threat modeling to identify potential security threats and vulnerabilities in the design. In the build phase, we can begin to utilize static code analysis; more specifically, we can begin using tools like SonarQube in our CI pipeline that will automatically analyze our code for bugs and vulnerabilities. And throughout the build phase and into the verify and test phase, we’ll implement automated testing frameworks that’ll run our unit and integration tests to ensure code quality and standard compliance.

Once design, build, and testing are completed, the transition and health check phase will automate the deployment process, configuration management, and environment provisioning. We’ll also utilize automated rollback mechanisms using CD tools in the event of deployment failures we can revert to a stabilized version and reduce impact. After our transition and health check, we head into our monitoring, detecting, and responding phases. During and after deployment, automated tools can periodically check application status and alert the necessary personnel to address issues that are detected. It’s essential that when we are running periodic health checks that we have robust log management. These automated log management tools will need to collect, analyze, and visualize data to assist developers with pattern recognition, detecting outliers, and troubleshooting quickly.

And lastly, in our final maintain and stabilize phase we’ll want to ensure ongoing reliability, performance and security of our application. We’re going to accomplish this through a regularly scheduled maintenance window for necessary patching and updates. This will ensure our app is always up to date with the latest features and quality of life improvements. We’ll need to continuously monitor performance issues and optimize where required. We’ll monitor security and incident reports in real-time. And we’ll run regular compliance audits to enforce compliance with regulatory standards as well as the standards outlined in this policy.

### 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 | Low | Unlikely | Low | P3 | 3 |
| STD-002-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | High | Likely | Medium | P18 | L1 |
| STD-009-CPP | Low | Probable | Medium | P4 | L3 |
| STD-010-CPP | High | Unlikely | Medium | P6 | L2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest is a security measure and refers to protecting data that is in a stored state, such as on a hard drive, SSD, etc. What encryption at rest accomplishes is ensuring the data is unreadable to any unauthorized users. But as the ‘encryption’ name suggests, the data can be decrypted with the correct decryption key. This policy applies to enhance our security measures; in the event there was a breach, and an unauthorized user was to gain access to the physical storage device, the data would remain protected and safe. It mitigates risk and is a requirement in several industry standards such as GDPR and HIPAA. |
| Encryption in flight | Encryption in flight is another security measure and refers to protecting data that is being transmitted from one location to another. This could be within a private network or over the internet. And much like encryption at rest, encryption in flight can be decrypted with the correct decryption key. What the policy accomplishes is making the data unreadable to any unauthorized users who intercept the data in transit. Again, this policy mitigates risk, enhances security, ensures data integrity, and is also a requirement in several industry standards such as GDPR and HIPAA. |
| Encryption in use | Encryption in use is the third encryption security measure and refers to protecting data while it is in use or actively being processed. This goes beyond the measures of protection in stored and transit states, and ensures data remains protected when it’s manipulated or accessed in memory. This can include the data being used by applications, by the CPU, or in memory. Technologies such as trusted execution environments are often used to implement this policy. And like encryption at rest and in flight, encryption in use is a compliance requirement to industry standards. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the first process of the Triple-A Framework, and it refers to verifying the identity of a user or device. Authentication is a common practice that is usually accomplished through a credential system such as a username and password. This typically occurs upon account creation as in new users or when users attempt to login from new devices. Applying this policy ensures only legitimate users are accessing your system. |
| Authorization | Authorization is the second process of the Triple-A Framework, and it refers to determining what authenticated users are allowed to do. This common practice is accomplished through limiting what unique users have access to and what actions they are allowed to perform by setting user permissions. Applying this policy further enhances security and protects sensitive information through access restrictions. |
| Accounting | Accounting is the third process that completes the Triple-A Framework, and it refers to tracking and logging all activities performed by users within the network. These activities can include but aren’t limited to actions performed, resources accessed, login times, changes made, etc. Applying this policy improves security monitoring, operational analysis, and provides an audit trail. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.5 | 01/26/2025 | 3-2 Milestone Requirements Added | Ryan Branchaud | Ryan Branchaud |
| 2.0 | 02/13/2025 | 6-2 Project One Requirements Added | Ryan Branchaud | Ryan Branchaud |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

## References

*SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence*. (n.d.). Wiki.sei.cmu.edu. <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

Magnusson, A. (2023, October 5). *What is AAA Security? Authentication, Authorization, and Accounting*. Discover.strongdm.com. <https://www.strongdm.com/blog/aaa-security>