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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_heading=h.gjdgxs)

[Purpose 2](#_heading=h.30j0zll)

[Scope 2](#_heading=h.1fob9te)

[Module Three Milestone 2](#_heading=h.3znysh7)

[Ten Core Security Principles 2](#_heading=h.4d34og8)

[C/C++ Ten Coding Standards 3](#_heading=h.2s8eyo1)

[Coding Standard 1 4](#_heading=h.3rdcrjn)

[Coding Standard 2 5](#_heading=h.26in1rg)

[Coding Standard 3 6](#_heading=h.lnxbz9)

[Coding Standard 4 7](#_heading=h.35nkun2)

[Coding Standard 5 8](#_heading=h.1ksv4uv)

[Coding Standard 6 9](#_heading=h.2jxsxqh)

[Coding Standard 7 10](#_heading=h.z337ya)

[Coding Standard 8 11](#_heading=h.28h4qwu)

[Coding Standard 9 13](#_heading=h.nmf14n)

[Coding Standard 10 14](#_heading=h.37m2jsg)

[Defense-in-Depth Illustration 15](#_heading=h.1mrcu09)

[Project One 15](#_heading=h.46r0co2)

[1.](#_heading=h.2lwamvv) Revise the C/C++ Standards 15

[2.](#_heading=h.111kx3o) Risk Assessment 15

[3.](#_heading=h.3l18frh) Automated Detection 15

[4.](#_heading=h.206ipza) Automation 15

[5.](#_heading=h.4k668n3) Summary of Risk Assessments 16

[6.](#_heading=h.2zbgiuw) Create Policies for Encryption and Triple A 16

[7.](#_heading=h.1egqt2p) Map the Principles 17

[Audit Controls and Management 18](#_heading=h.3ygebqi)

[Enforcement 18](#_heading=h.2dlolyb)

[Exceptions Process 18](#_heading=h.sqyw64)

[Distribution 19](#_heading=h.3cqmetx)

[Policy Change Control 19](#_heading=h.1rvwp1q)

[Policy Version History 19](#_heading=h.4bvk7pj)

[Appendix A Lookups 19](#_heading=h.2r0uhxc)

[Approved C/C++ Language Acronyms 19](#_heading=h.1664s55)

## 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 | Verify all inputs are of the correct size and format before processing. This will help prevent SQL injection attacks, for example. |
| 1. Heed Compiler Warnings | Your compiler will sometimes notify you of problems in your code. Your code will still run, but it may be incorrect or unsafe. A best practice is to solve these warnings when they appear. |
| 1. Architect and Design for Security Policies | Design your code with security as a top priority. |
| 1. Keep It Simple | Make your code as easy to understand as possible. |
| 1. Default Deny | Deny access to your program by default. Continually verify the information and connections to your program are clean. |
| 1. Adhere to the Principle of Least Privilege | Users and processes should have only the minimum privileges needed to do their jobs. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data sent to other programs. |
| 1. Practice Defense in Depth | Defense in Depth is a security practice that uses multiple layers of defense to protect against attacks. If one layer of defence is compromised, several more can protect your assets. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance, if done right, can prevent attacks before they are attempted. |
| 1. Adopt a Secure Coding Standard | Find a secure coding standard that applies for your application, language and system. |

### 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 | DCL52-CPP. Never qualify a reference type with const or volatile |

| **Noncompliant Code** |
| --- |
| The reference 'const' is qualified instead of the non-reference value 'p'. |
| char &const p; |

| **Compliant Code** |
| --- |
| 'p' is correctly qualified. |
| char const &p; // Or: const char &p; |

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

| **Principles(s):**  1. Heed Compiler Warnings:  Misusing const or volatile qualifiers on reference types can lead to compiler warnings or errors. Addressing these warnings ensures code correctness and prevents unintended behavior.  2. Adopt a Secure Coding Standard:  This policy is part of the CERT C Coding Standard, which promotes secure, reliable, and predictable behavior in code. Adopting such standards helps developers avoid common pitfalls and improve overall code quality. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | 3 | 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) | 2024.2 | **C++0014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | **CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **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) |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | INT30-C. Ensure that unsigned integer operations do not wrap |

| **Noncompliant Code** |
| --- |
| Unsigned integers have a maximum and minimum value. If they try to go over the max or under the min, they will wrap to the min or max values respectively. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The code is checked for wrapping before the equation happens. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

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

| **Principles(s):**  1. Validate Input Data:  Unsigned integer wraparounds can occur due to unexpected or invalid input data, such as overly large values. Ensuring that input data is within expected bounds can help prevent these conditions.  2. Heed Compiler Warnings:  Compilers may issue warnings when they detect potential overflow or wraparound issues with unsigned integers. Paying attention to these warnings can help developers identify and address these problems early in the development process.  3. Keep It Simple:  Preventing wraparounds often involves using straightforward checks to ensure values remain within expected ranges. Keeping the code simple makes it easier to understand, test, and maintain, which reduces the risk of introducing bugs.  4. Adopt a Secure Coding Standard:  This policy is part of the CERT C Coding Standard, which promotes secure, reliable, and predictable behavior in code. Adopting such standards helps developers avoid common pitfalls and improve overall code quality. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | 9 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **integer-overflow** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | **CertC-INT30** | Implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this rule by ensuring that operations are checked for overflow before being performed (Be mindful of exception INT30-EX2 because it excuses many operations from requiring [validation](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-validation), including all the operations that would validate a potentially dangerous operation. For instance, adding two unsigned ints together requires validation involving subtracting one of the numbers from UINT\_MAX, which itself requires no validation because it cannot wrap.) |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **INTEGER\_OVERFLOW** | Implemented |
| [Cppcheck Premium](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck+Premium) | 24.9.0 | **premium-cert-int30-c** | Partially implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.2 | **C2910, C3383, C3384, C3385, C3386**  **C++2910**  **DF2911, DF2912, DF2913,** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.2 | **NUM.OVERFLOW CWARN.NOEFFECT.OUTOFRANGE NUM.OVERFLOW.DF** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **493 S, 494 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-INT30-a CERT\_C-INT30-b CERT\_C-INT30-c** | Avoid wraparounds when performing arithmetic integer operations Integer overflow or underflow in constant expression in '+', '-', '\*' operator Integer overflow or underflow in constant expression in '<<' operator |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rule INT30-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint30c.html) | Checks for:   * Unsigned integer overflow * Unsigned integer constant overflow   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.33 | [**V658**](https://pvs-studio.com/en/docs/warnings/v658/)**,** [**V1012**](https://pvs-studio.com/en/docs/warnings/v1012/)**,** [**V1028**](https://pvs-studio.com/en/docs/warnings/v1028/)**,** [**V5005**](https://pvs-studio.com/en/docs/warnings/v5005/)**,** [**V5011**](https://pvs-studio.com/en/docs/warnings/v5011/) |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **unsigned overflow** | Exhaustively verified. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | STR30-C. Do not attempt to modify string literals |

| **Noncompliant Code** |
| --- |
| Modifying string literals causes undefined behavior. |
| char \*str = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| If you need to modify a string, use an array initilizer. |
| char str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):**  1. Heed Compiler Warnings:  Modifying string literals can lead to undefined behavior, and many compilers will issue warnings or errors if they detect an attempt to do so. Paying attention to these warnings helps prevent such dangerous operations.  2. Keep It Simple:  Avoiding the modification of string literals simplifies code by ensuring that data intended to be immutable remains so. This makes the code more predictable and easier to reason about.  3. Adopt a Secure Coding Standard:  This policy is part of the CERT C Coding Standard, which promotes secure, reliable, and predictable behavior in code. Adopting such standards helps developers avoid common pitfalls and improve overall code quality. |
| --- |

**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) | 2024.2 | **C0556, C0752, C0753, C0754**  **C++3063, C++3064, C++3605, C++3606, C++3607** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.2 | **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) | 7.33 | [**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** | STD-004-JAV | IDS00-J. Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| This exerpt allows an user to input a value such as "validuser' OR '1'='1" which will return all database entries. |
| String sqlString = "SELECT \* FROM db\_user WHERE username = '"  + username +  "' AND password = '" + pwd + "'"; |

| **Compliant Code** |
| --- |
| setString is used here for input sanitization. |
| String sqlString =  "select \* from db\_user where username=? and password=?";  PreparedStatement stmt = connection.prepareStatement(sqlString);  stmt.setString(1, username);  stmt.setString(2, pwd); |

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

| **Principles(s):**  1. Validate Input Data:  Preventing SQL injection starts with validating and sanitizing user input. Ensuring that inputs conform to expected patterns (e.g., numeric, email formats) reduces the likelihood of malicious SQL commands being executed.  2. Sanitize Data Sent to Other Systems:  Use parameterized queries (prepared statements) to ensure that input data is properly escaped and does not alter the structure of SQL commands. This prevents attackers from injecting harmful SQL code through user input.  3. Adopt a Secure Coding Standard:  This policy is part of the CERT Java Coding Standard, which promotes secure, reliable, and predictable behavior in code. Adopting such standards helps developers avoid common pitfalls and improve overall code quality. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | 18 | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [The Checker Framework](https://wiki.sei.cmu.edu/confluence/display/java/The+Checker+Framework) | 2.1.3 | **Tainting Checker** | Trust and security errors (see Chapter 8) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **JAVA.IO.INJ.SQL** | SQL Injection (Java) |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/java/Coverity) | 7.5 | **SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_ FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [Findbugs](https://wiki.sei.cmu.edu/confluence/display/java/Findbugs) | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [Fortify](https://wiki.sei.cmu.edu/confluence/display/java/Fortify) | 1.0 | **HTTP\_Response\_Splitting SQL\_Injection\_\_Persistence SQL\_Injection** | Implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/java/Klocwork) | 2024.2 | **SV.DATA.DB SV.SQL SV.SQL.DBSOURCE** | Implemented |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2024.1 | **CERT.IDS00.TDSQL** | Protect against SQL injection |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 9.9 | [**S2077**](https://rules.sonarsource.com/java/RSPEC-2077)  [**S3649**](https://rules.sonarsource.com/java/RSPEC-3649) | [Executing SQL queries is security-sensitive](https://rules.sonarsource.com/java/RSPEC-2077)  [SQL queries should not be vulnerable to injection attacks](https://rules.sonarsource.com/java/RSPEC-3649) |
| [SpotBugs](https://wiki.sei.cmu.edu/confluence/display/java/SpotBugs) | 4.6.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | MEM35-C. Allocate sufficient memory for an object |

| **Noncompliant Code** |
| --- |
| sizeof(int) is used instead of sizeof(long) when allocating memory for a long. Because a long is 32bit and int is 16, writing to the variable can cause a buffer overflow. |
| #include <stdint.h>  #include <stdlib.h>    void function(size\_t len) {  long \*p;  if (len == 0 || len > SIZE\_MAX / sizeof(long)) {  /\* Handle overflow \*/  }  p = (long \*)malloc(len \* sizeof(int));  if (p == NULL) {  /\* Handle error \*/  }  free(p);  } |

| **Compliant Code** |
| --- |
| sizeof(long) is used correctly. Alternatively, sizeof(\*p) can be used instead of sizeof(long). |
| #include <stdint.h>  #include <stdlib.h>    void function(size\_t len) {  long \*p;  if (len == 0 || len > SIZE\_MAX / sizeof(long)) {  /\* Handle overflow \*/  }  p = (long \*)malloc(len \* sizeof(long));  if (p == NULL) {  /\* Handle error \*/  }  free(p);  } |

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

| **Principles(s):**  1. Validate Input Data:  When allocating memory based on input (e.g., user-specified sizes), ensure the input is validated. This prevents situations where insufficient memory is allocated due to unexpected or malicious input, leading to overflow conditions.  2. Heed Compiler Warnings:  Compilers can sometimes warn about suspicious memory allocation patterns or potential buffer overflows. Addressing these warnings can help catch issues where memory allocation might be insufficient.  3. Keep It Simple:  Proper memory allocation practices simplify code by ensuring that objects have enough space for all their data. This reduces the likelihood of subtle bugs and vulnerabilities that can arise from incorrect memory calculations.  4. Adopt a Secure Coding Standard:  This policy is part of the CERT C Coding Standard, which promotes secure, reliable, and predictable behavior in code. Adopting such standards helps developers avoid common pitfalls and improve overall code quality. |
| --- |

**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 | **malloc-size-insufficient** | Partially checked  Besides direct rule violations, all undefined behaviour resulting from invalid memory accesses is reported by Astrée. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | **CertC-MEM35** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW ALLOC.SIZE.TRUNC IO.TAINT.SIZE MISC.MEM.SIZE.BAD LANG.MEM.BO LANG.MEM.BU LANG.STRUCT.PARITH LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.MEM.TBA LANG.MEM.TO LANG.MEM.TU** | Addition overflow of allocation size Addition overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Truncation of allocation size Tainted allocation size Unreasonable size argument Buffer Overrun Buffer Underrun Pointer Arithmetic Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could check violations of this rule by examining the size expression to malloc() or memcpy() functions. Specifically, the size argument should be bounded by 0, SIZE\_MAX, and, unless it is a variable of type size\_t or rsize\_t, it should be bounds-checked before the malloc() call. If the argument is of the expression a\*b, then an appropriate check is   | **if** (a < SIZE\_MAX / b && a > 0) ... | | --- | |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **BAD\_ALLOC\_STRLEN**  **SIZECHECK (deprecated)** | Partially implemented  Can find instances where string length is miscalculated (length calculated may be one less than intended) for memory allocation purposes. Coverity Prevent cannot discover all violations of this rule, so further verification is necessary  Finds memory allocations that are assigned to a pointer that reference objects larger than the allocated block |
| [Cppcheck Premium](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck+Premium) | 24.9.0 | **premium-cert-mem35-c** | Partially implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.2 | **C0696, C0701, C1069, C1071, C1073, C2840**  **DF2840, DF2841, DF2842, DF2843, DF2935, DF2936, DF2937, DF2938** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.2 | **INCORRECT.ALLOC\_SIZE SV.TAINTED.ALLOC\_SIZE** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **400 S, 487 S, 115 D** | Enhanced enforcement |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/c/Splint) | 3.1.1 |  |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-MEM35-a** | Do not use sizeof operator on pointer type to specify the size of the memory to be allocated via 'malloc', 'calloc' or 'realloc' function |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **433, 826** | Partially supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rule MEM35-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemem35c.html) | Checks for:   * Pointer access out of bounds * Memory allocation with tainted size   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.33 | [**V531**](https://pvs-studio.com/en/docs/warnings/v531/), [**V635**](https://pvs-studio.com/en/docs/warnings/v635/), [**V781**](https://pvs-studio.com/en/docs/warnings/v781/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 | **malloc-size-insufficient** | Partially checked |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **mem\_access** | Exhaustively detects undefined behavior (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/77590559)). |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | ERR06-C. Understand the termination behavior of assert() and abort() |

| **Noncompliant Code** |
| --- |
| Assert will call abort(), which will cause cleanup() to not run. |
| void cleanup(void) {  /\* Delete temporary files, restore consistent state, etc. \*/  }    int main(void) {  if (atexit(cleanup) != 0) {  /\* Handle error \*/  }    /\* ... \*/    assert(/\* Something bad didn't happen \*/);    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| cleanup() will still run even if "something bad happened". |
| void cleanup(void) {  /\* Delete temporary files, restore consistent state, etc. \*/  }    int main(void) {  if (atexit(cleanup) != 0) {  /\* Handle error \*/  }    /\* ... \*/    if (/\* Something bad happened \*/) {  exit(EXIT\_FAILURE);  }    /\* ... \*/  } |

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

| **Principles(s):**  1. Keep It Simple:  Using assert() and abort() correctly ensures the program terminates in a controlled manner during critical failures. Knowing when and how they cause program termination simplifies error handling and makes the code easier to understand and maintain.  2. Architect and Design for Security Policies:  Understanding the behavior of assert() and abort() helps in designing systems that handle failures securely. For instance, it can help avoid revealing sensitive information through abrupt crashes or providing inconsistent program states.  3. Adopt a Secure Coding Standard:  This policy is part of the CERT C Coding Standard, which promotes secure, reliable, and predictable behavior in code. Adopting such standards helps developers avoid common pitfalls and improve overall code quality. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
|  |  |  |  |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | CERT\_C-ERR06-a | Do not use assertions |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **586** | Fully supported |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio_V) | 7.33 | [**V2021**](https://pvs-studio.com/en/docs/warnings/v2021/) |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | ERR51-CPP. Handle all exceptions |

| **Noncompliant Code** |
| --- |
| No exception handling is used. This will cause the program to terminate before other potentially-important actions can be taken. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| Exception handling is used. |
| 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):**  1. Keep It Simple:  Handling exceptions consistently simplifies code by providing a clear path for error management. This makes the control flow easier to follow and reduces the risk of unhandled exceptions leading to crashes or undefined behavior.  2. Adhere to the Principle of Least Privilege:  By catching exceptions at the appropriate level, you can ensure that only the necessary code is executed during error recovery. This minimizes the exposure of sensitive resources and operations when an error occurs.  3. Practice Defense in Depth:  Comprehensive exception handling acts as an additional layer of defense against unexpected situations. By preparing for potential errors at various points in the code, you can prevent vulnerabilities that might arise from unhandled exceptions.  4. Adopt a Secure Coding Standard:  This policy is part of the CERT C++ Coding Standard, which promotes secure, reliable, and predictable behavior in code. Adopting such standards helps developers avoid common pitfalls and improve overall code quality. |
| --- |

**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) | 2024.2 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | **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** |
| --- | --- | --- |
| Randomness | STD-008-CPP | MSC50-CPP. Do not use std::rand() for generating pseudorandom numbers |

| **Noncompliant Code** |
| --- |
| std::rand() is too predictable to be secure. |
| #include <cstdlib>  #include <string>    void f() {  std::string id("ID"); // Holds the ID, starting with the characters "ID" followed  // by a random integer in the range [0-10000].  id += std::to\_string(std::rand() % 10000);  // ...  } |

| **Compliant Code** |
| --- |
| This code uses a Mersenne Twister algorithm to randomize. |
| #include <random>  #include <string>    void f() {  std::string id("ID"); // Holds the ID, starting with the characters "ID" followed  // by a random integer in the range [0-10000].  std::uniform\_int\_distribution<int> distribution(0, 10000);  std::random\_device rd;  std::mt19937 engine(rd());  id += std::to\_string(distribution(engine));  // ...  } |

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

| **Principles(s):**  1. Architect and Design for Security Policies:  Using std::rand() can lead to predictable pseudorandom number sequences, which may be exploited in security-sensitive applications. Employing secure random number generators in design prevents vulnerabilities that could arise from weak randomness.  2. Keep It Simple:  Relying on well-established libraries or functions for generating random numbers simplifies the implementation and avoids the complexities and pitfalls of using std::rand(), which may not provide adequate randomness or security.  3. Practice Defense in Depth:  Utilizing secure random number generators adds an additional layer of defense by ensuring that the randomness used in cryptographic or security-sensitive contexts is robust, thus mitigating the risks associated with predictable outputs.  4. Adopt a Secure Coding Standard:  This policy is part of the CERT C++ Coding Standard, which promotes secure, reliable, and predictable behavior in code. Adopting such standards helps developers avoid common pitfalls and improve overall code quality. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **bad-function (AUTOSAR.26.5.1A)** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MSC50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 4.0 (prerelease) | cert-msc50-cpp | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **BADFUNC.RANDOM.RAND** | Use of rand |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  |  |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.MSC30** | Fully implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++5028** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | **CERT.MSC.STD\_RAND\_CALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **44 S** | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MSC50-a** | Do not use the rand() function for generating pseudorandom numbers |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: MSC50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc50cpp.html) | Checks for use of vulnerable pseudo-random number generator (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **bad-function (AUTOSAR.26.5.1A)** | Fully checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Function Return | STD-009-CPP | MSC52-CPP. Value-returning functions must return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| Failing to return a value in one path of a function results in undefined behavior. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| Ensure all paths return a value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  return a;  } |

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

| **Principles(s):**  1. Keep It Simple:  Ensuring that every exit path in a function returns a value simplifies the code’s logic and flow. This reduces ambiguity about what a function will return and makes it easier for other developers to understand and maintain the code.  2. Heed Compiler Warnings:  Compilers often issue warnings when a value-returning function might not return a value. Addressing these warnings helps catch potential bugs early in the development process and prevents undefined behavior.  3. Practice Defense in Depth:  By guaranteeing that all exit paths return a value, you prevent scenarios where a function could leave the program in an inconsistent state. This acts as an additional layer of defense against unexpected behavior or crashes that could arise from missing return values.  4. Adopt a Secure Coding Standard:  This policy is part of the CERT C++ Coding Standard, which promotes secure, reliable, and predictable behavior in code. Adopting such standards helps developers avoid common pitfalls and improve overall code quality. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **return-implicit** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MSC52** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | **-Wreturn-type** | Does not catch all instances of this rule, such as *function-try-blocks* |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.STRUCT.MRS LANG.STRUCT.NVNR** | Missing return statement  Non-void noreturn, |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **DF2888** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | **FUNCRET.GEN**  **FUNCRET.IMPLICIT** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **2 D, 36 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MSC52-a** | All exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: MSC52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc52cpp.html) | Checks for missing return statements (rule partially covered) |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S935**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-935) |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.33 | [**V591**](https://pvs-studio.com/en/docs/warnings/v591/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **return-implicit** | Fully checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Termination | STD-010-CPP | ERR50-CPP. Do not abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| The code may terminate unexpectedly because f(), which is called at exit, may cause the program to terminate because throwing\_func() calls an exception. |
| #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** |
| --- |
| f() handles exceptions which prevents unexpected termination. |
| #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):**  1. Architect and Design for Security Policies:  Designing systems to handle errors without abrupt termination enhances overall stability and security. This approach prevents the application from entering an inconsistent state, which could lead to vulnerabilities or data loss.  2. Practice Defense in Depth:  By ensuring that errors are managed appropriately and not resulting in abrupt termination, you create multiple layers of defense in your application. This allows for recovery strategies and cleanup processes to run, mitigating potential negative impacts of failures.  3. Keep It Simple:  Implementing structured error handling simplifies the code by providing a clear strategy for managing exceptional cases. This makes the program more maintainable and reduces complexity, helping other developers understand how to handle errors.  4. Adopt a Secure Coding Standard:  This policy is part of the CERT C++ Coding Standard, which promotes secure, reliable, and predictable behavior in code. Adopting such standards helps developers avoid common pitfalls and improve overall code quality. |
| --- |

**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) | 8.1p0 | **BADFUNC.ABORT BADFUNC.EXIT** | Use of abort Use of exit |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.2 | **C++5014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.2 | **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) | 2023.1 | **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 (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) | 7.33 | [**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) |  |

### 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.

[Insert your written explanations here.]

### 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 | 3 | 3 |
| STD-002-CPP | High | Likely | High | 9 | 2 |
| STD-003-CPP | Low | Likely | Low | 9 | 2 |
| STD-004-CPP | High | Likely | Medium | 18 | 1 |
| STD-005-CPP | High | Probable | High | 6 | 2 |
| STD-006-CPP | Medium | Unlikely | Medium | 4 | 3 |
| STD-007-CPP | Low | Probable | Medium | 4 | 3 |
| STD-008-CPP | Medium | Unlikely | Low | 6 | 2 |
| STD-009-CPP | Medium | Probable | Medium | 8 | 2 |
| STD-010-CPP | Low | Probable | Medium | 4 | 3 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest refers to the process of encrypting data that is stored on storage media, such as hard drives or cloud storage. This type of encryption is crucial for protecting sensitive information, ensuring that even if the data is stolen or accessed without authorization, it remains unreadable without the appropriate decryption keys. By implementing encryption at rest, organizations can enhance their data security and comply with regulatory requirements, reducing the risk of data breaches. |
| Encryption in flight | Encryption in flight refers to the process of encrypting data while it is being transmitted over a network. This type of encryption is essential for protecting sensitive information from interception and unauthorized access during transmission. By encrypting data in flight, organizations can effectively prevent threats such as man-in-the-middle attacks and eavesdropping. Implementing encryption protocols ensures that data remains secure as it travels between devices, helping to maintain confidentiality and integrity throughout the communication process. |
| Encryption in use | Encryption in use refers to the protection of data while it is being processed or utilized by applications. Unlike encryption at rest or in flight, which secures data when it is stored or transmitted, encryption in use safeguards sensitive information during active operations, such as computations or analytics. This type of encryption is critical for ensuring that data remains confidential even when it is accessed by applications, helping to prevent unauthorized access and data breaches. Techniques such as homomorphic encryption allow for computations to be performed on encrypted data without needing to decrypt it first, thereby maintaining security throughout the entire data lifecycle. Implementing encryption in use is especially important in environments that handle highly sensitive information, ensuring compliance with regulations and enhancing overall data protection. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying the identity of a user, device, or system, ensuring that the entity attempting to access a resource is who they claim to be. This typically involves credential submission, where users provide identifiers like usernames and passwords during user logins, which are then verified against a secure database. Effective authentication mechanisms help prevent unauthorized access and establish accountability by linking users to their actions. This is particularly important when changes are made to the database, as tracking who accessed the system and when can provide valuable insights for security audits and incident investigations. |
| Authorization | Once a user is authenticated, Authorization determines what that user is allowed to do within the system. This process defines the permissions and access levels granted based on the user's role or attributes, utilizing methods like Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC). A key aspect of authorization is defining the user level of access, ensuring that users can only access files and resources necessary for their specific roles. By carefully controlling access to sensitive files and functionalities, organizations can minimize risks and adhere to the principle of least privilege, thus protecting critical data from unauthorized exposure. |
| Accounting | Accounting, or auditing, refers to the tracking of user activity within a system. This includes logging user actions such as user logins, any changes made to the database, the addition of new users, and files accessed by users. Maintaining detailed logs of these activities is vital for incident response, as it provides the necessary information to identify and investigate security incidents. Accounting promotes accountability by linking users to their activities, thereby deterring potential malicious behavior. Additionally, maintaining detailed logs is often a requirement for compliance with various regulations and standards. |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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 |