**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 | All data from outside or untrusted sources will be validated for quality and security. Untrusted data is one of the largest sources for security exploits, so that data must always be validated. |
| 1. Heed Compiler Warnings | Compilers may allow code to be compiled with potential flaws that are still syntactically correct. These are reported as warnings rather than errors. Although it is possible to compile such code, all warnings should be resolved when possible. |
| 1. Architect and Design for Security Policies | The organization must have rules in place to govern how systems should be architected and allowed to interact so that concepts such as authentication, authorization, and data boundaries can be maintained. |
| 1. Keep It Simple | Designs, systems, algorithms, and code should be kept simple whenever possible. Unnecessary complexity lowers understanding and increases the chance of flaws, which can increase the chance of security vulnerabilities in a system. |
| 1. Default Deny | When access to an object, system, or functionality is attempted the system should assume that access is denied unless it is explicitly granted. Default values for access flags should be false and set to true when confirmed not the other way around. |
| 1. Adhere to the Principle of Least Privilege | When accessing resources each part of a system should only be granted the minimum authorization necessary to complete its functionality. |
| 1. Sanitize Data Sent to Other Systems | Data that crosses a security boundary may expose secrets or sensitive data to untrusted actors. It’s important to make sure that all data which crosses a security boundary is properly sanitized to remove any data to which the outside actor should not have access. |
| 1. Practice Defense in Depth | Multiple layers of security should be used to create a web that can provide greater coverage. This is superior to relying on a single layer of security and trying to make it cover all areas. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance is critical in finding potential flaws in a system which might create vulnerabilities. Quality assurance should be an inherent part of the development process. |
| 1. Adopt a Secure Coding Standard | In order to provide the best security an organization should have a consistent approach to coding which emphasizes secure programming. Without standards around many coding patterns code can be difficult to understand between coders and some coders may not be aware of flaws in their own code. By working together as a team the development team is more likely to produce more secure code. |

### 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** | **Never qualify a reference type with const or volatile** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | C++ does not allow you to change the value of a reference type, effectively treating all references as being const qualified. The C++ Standard, [dcl.ref], paragraph 1 [ISO/IEC 14882-2014], states the following:  Cv-qualified references are ill-formed except when the cv-qualifiers are introduced through the use of a typedef-name (7.1.3, 14.1) or decltype-specifier (7.1.6.2), in which case the cv-qualifiers are ignored.  Thus, C++ prohibits or ignores the cv-qualification of a reference type. Only a value of non-reference type may be cv-qualified. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example correctly declares p to be a reference to a const-qualified char. The subsequent modification of p makes the program ill-formed. |
| #include <iostream>    void f(char c) {  const char &p = c;  p = 'p'; // Error: read-only variable is not assignable  std::cout << c << std::endl;  } |

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

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

| **Principles(s):** **Heed compiler warnings.** Conforming compilers will issue a warning for this issue. Some compilers may still allow the code to compile with the warning message, but the results will be undefined. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL52 |  |
| Helix QAC | 2023.1 | C++0014 |  |
| Klocwork | 2023.1 | CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-DCL52-a | Never qualify a reference type with ‘const’ or ‘volatile’ |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [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. |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **0014** |  |
| [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** | **Ensure that integer conversions do not result in lost or misinterpreted data** |
| --- | --- | --- |
| **Data Value** | [STD-002- CPP] | Integer conversions, both implicit and explicit (using a cast), must be guaranteed not to result in lost or misinterpreted data. This rule is particularly true for integer values that originate from untrusted sources and are used in any of the following ways:   * Integer operands of any pointer arithmetic, including array indexing * The assignment expression for the declaration of a variable length array * The postfix expression preceding square brackets [] or the expression in square brackets [] of a subscripted designation of an element of an array object * Function arguments of type size\_t or rsize\_t (for example, an argument to a memory allocation function) |

| **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):** **Validate input data.** Data entering a function or area of code must be checked to ensure that it will not cause invalid math operations or operations that might result in overflow errors.  **Defense in depth.** Security must be considered at all levels, including potential code specific vulnerabilities such as potential overflow 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) | 22.04 |  |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **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** |  |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity)\* | 2017.07 | **NEGATIVE\_RETURNS**  **REVERSE\_NEGATIVE**  **MISRA\_CAST** |  |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | **memsetValueOutOfRange** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **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) | 2023.1 | **PORTING.CAST.SIZE** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **93 S, 433 S, 434 S** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **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** |  |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rule INT31-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint31c.html) |  |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **2850, 2851, 2852, 2853, 2855, 2856, 2857, 2858,**  **2890, 2891, 2892, 2893, 2895, 2896, 2897, 2898**  **2900, 2901, 2902, 2903, 2905, 2906, 2907, 2908** |  |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2850, 2851, 2852, 2853, 2855, 2856, 2857, 2858,**  **2890, 2891, 2892, 2893, 2895, 2896, 2897, 2898,**  **2900, 2901, 2902, 2903, 2905, 2906, 2907, 2908,**  **3000, 3010** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.24 | [**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) | 22.04 |  |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **signed\_downcast** |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | [STD-003- CPP] | Copying data to a buffer that is not large enough to hold that data results in a buffer overflow. Buffer overflows occur frequently when manipulating strings [Seacord 2013]. To prevent such errors, either limit copies through truncation or, preferably, ensure that the destination is of sufficient size to hold the data to be copied. C-style strings require a null character to indicate the end of the string, while the C++ std::basic\_string template requires no such character. |

| **Noncompliant Code** |
| --- |
| Because the input is unbounded, the following code could lead to a buffer overflow. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The best solution for ensuring that data is not truncated and for guarding against buffer overflows is to use std::string instead of a bounded array, as in this compliant solution. |
| #include <iostream>  #include <string>    void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):** **Validate input data.** All outside data should be checked to make sure that it will not overflow strings in the code which could cause overflow vulnerabilities. |
| --- |

**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) | 22.04 |  | Supported  Astrée reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-STR31** | Detects calls to unsafe string function that may cause buffer overflow Detects potential buffer overruns, including those caused by unsafe usage of fscanf() |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.MEM.BO LANG.MEM.TO MISC.MEM.NTERM BADFUNC.BO.\*** | Buffer overrun Type overrun No space for null terminator A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| [Compass/ROSE](https://www.securecoding.cert.org/confluence/display/seccode/Rose) |  |  | Can detect violations of the rule. However, it is unable to handle cases involving strcpy\_s() or manual string copies such as the one in the first example |
| [Coverity](https://www.securecoding.cert.org/confluence/display/seccode/Coverity) | 2017.07 | **STRING\_OVERFLOW**  **BUFFER\_SIZE**  **OVERRUN**  **STRING\_SIZE** | Fully implemented |
| [Fortify SCA](https://www.securecoding.cert.org/confluence/display/seccode/Fortify) | 5.0 |  |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **C2840,  C5009, C5038**  **C++0145, C++5009, C++5038**  **DF2840, DF2841, DF2842, DF2843, DF2845, DF2846, DF2847, DF2848, DF2930, DF2931, DF2932, DF2933, DF2935, DF2936, DF2937, DF2938** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2023.1 | **SV.FMT\_STR.BAD\_SCAN\_FORMAT** **SV.UNBOUND\_STRING\_INPUT.FUNC** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152309) | 9.7.1 | **489 S, 109 D, 66 X, 70 X, 71 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-STR31-a** **CERT\_C-STR31-b** **CERT\_C-STR31-c** **CERT\_C-STR31-d** **CERT\_C-STR31-e** | Avoid accessing arrays out of bounds Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Avoid using unsafe string functions which may cause buffer overflows |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **421, 498** | Partially supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rule STR31-C](https://www.mathworks.com/help/bugfinder/ref/certcrulestr31c.html) | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation * Insufficient destination buffer size   Rule partially covered. |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **5009, 5038, 2840, 2841, 2842, 2843, 2845, 2846, 2847, 2848, 2930, 2931, 2932, 2933, 2935, 2936, 2937, 2938** | Partially implemented |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **0145, 2840, 2841, 2842, 2843, 2845, 2846, 2847, 2848, 2930, 2931, 2932, 2933, 2935, 2936, 2937, 2938, 5006, 5038** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.24 | [**V518**](https://pvs-studio.com/en/docs/warnings/v518/), [**V645**](https://pvs-studio.com/en/docs/warnings/v645/), [**V727**](https://pvs-studio.com/en/docs/warnings/v727/), [**V755**](https://pvs-studio.com/en/docs/warnings/v755/) |  |
| [Splint](https://www.securecoding.cert.org/confluence/display/seccode/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/144ae03a)). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Sanitize data passed to complex subsystems** |
| --- | --- | --- |
| **SQL Injection** | [STD-004- CPP] | String data passed to complex subsystems may contain special characters that can trigger commands or actions, resulting in a software vulnerability. As a result, it is necessary to sanitize all string data passed to complex subsystems so that the resulting string is innocuous in the context in which it will be interpreted. |

| **Noncompliant Code** |
| --- |
| Data sanitization requires an understanding of the data being passed and the capabilities of the subsystem. |
| 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):** **Validate input data.** All data from outside sources should be considered untrusted and potentially dangerous unless it has been appropriately validated. |
| --- |

**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) | 22.04 |  | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **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) | 2023.1 | **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) | 2022.2 | **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) | R2023a | [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** | **Allocate and free memory in the same module, at the same level of abstraction** |
| --- | --- | --- |
| **Memory Protection** | [STD-005- CPP] | Dynamic memory management is a common source of programming flaws that can lead to security vulnerabilities. Poor memory management can lead to security issues, such as heap-buffer overflows, dangling pointers, and double-free issues [Seacord 2013]. From the programmer's perspective, memory management involves allocating memory, reading and writing to memory, and deallocating memory.  Allocating and freeing memory in different modules and levels of abstraction may make it difficult to determine when and if a block of memory has been freed, leading to programming defects, such as memory leaks, double-free vulnerabilities, accessing freed memory, or writing to freed or unallocated memory.  To avoid these situations, memory should be allocated and freed at the same level of abstraction and, ideally, in the same code module. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows a double-free vulnerability resulting from memory being allocated and freed at differing levels of abstraction. In this example, memory for the list array is allocated in the process\_list() function. The array is then passed to the verify\_size() function that performs error checking on the size of the list. If the size of the list is below a minimum size, the memory allocated to the list is freed, and the function returns to the caller. The calling function then frees this same memory again, resulting in a double-free and potentially exploitable vulnerability. |
| enum { MIN\_SIZE\_ALLOWED = 32 };    int verify\_size(char \*list, size\_t size) {  if (size < MIN\_SIZE\_ALLOWED) {  /\* Handle error condition \*/  free(list);  return -1;  }  return 0;  }    void process\_list(size\_t number) {  char \*list = (char \*)malloc(number);  if (list == NULL) {  /\* Handle allocation error \*/  }    if (verify\_size(list, number) == -1) {  free(list);  return;  }    /\* Continue processing list \*/    free(list);  } |

| **Compliant Code** |
| --- |
| To correct this problem, the error-handling code in verify\_size() is modified so that it no longer frees list. This change ensures that list is freed only once, at the same level of abstraction, in the process\_list() function. |
| enum { MIN\_SIZE\_ALLOWED = 32 };    int verify\_size(const char \*list, size\_t size) {  if (size < MIN\_SIZE\_ALLOWED) {  /\* Handle error condition \*/  return -1;  }  return 0;  }    void process\_list(size\_t number) {  char \*list = (char \*)malloc(number);    if (list == NULL) {  /\* Handle allocation error \*/  }    if (verify\_size(list, number) == -1) {  free(list);  return;  }    /\* Continue processing list \*/    free(list);  } |

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

| **Principles(s):** **Architect and design for security policies.** Systems should be architected in such a way that memory leaks and other memory issues such as double freeing memory are less likely to occur. This includes allocating and freeing memory in the same module.  **Adopt a coding standard.** By always handling memory in the same module developers do not need to wonder whether or not they are responsible for part of the memory handling. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **ALLOC.DF ALLOC.LEAK** | Double free Leak |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect possible violations by reporting any function that has malloc() or free() but not both. This would catch some false positives, as there would be no way to tell if malloc() and free() are at the same level of abstraction if they are in different functions |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | **RESOURCE\_LEAK** | Fully implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2023.1 | **FREE.INCONSISTENT** **UFM.FFM.MIGHT** **UFM.FFM.MUST** **UFM.DEREF.MIGHT** **UFM.DEREF.MUST** **UFM.RETURN.MIGHT** **UFM.RETURN.MUST** **UFM.USE.MIGHT** **UFM.USE.MUST** **MLK.MIGHT** **MLK.MUST** **MLK.RET.MIGHT** **MLK.RET.MUST** **FNH.MIGHT** **FNH.MUST** **FUM.GEN.MIGHT** **FUM.GEN.MUST** **RH.LEAK** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **50 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-MEM00-a** **CERT\_C-MEM00-b** **CERT\_C-MEM00-c** **CERT\_C-MEM00-d** **CERT\_C-MEM00-e** | Do not allocate memory and expect that someone else will deallocate it later Do not allocate memory and expect that someone else will deallocate it later Do not allocate memory and expect that someone else will deallocate it later Do not use resources that have been freed Ensure resources are freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) |  |  | Runtime analysis |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **449, 2434** | Partially supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rec. MEM00-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.mem00c.html) | Checks for:   * Invalid free of pointer * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rec. partially covered. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Assertions** | [STD-006- CPP] | Assertions are a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities (see MSC11-C. Incorporate diagnostic tests using assertions). The runtime assert() macro has some limitations, however, in that it incurs a runtime overhead and because it calls abort(). Consequently, the runtime assert() macro is useful only for identifying incorrect assumptions and not for runtime error checking. As a result, runtime assertions are generally unsuitable for server programs or embedded systems. |

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly: |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution: |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** **Practice defense in depth.** Using asserts properly, as a compile-time check, can provide another layer of protection for a system.  **Use effective quality assurance techniques.** As part of automated QA for code these compile-time checks can catch some errors that might otherwise slip through during code changes. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007- CPP] | When an exception is thrown, control is transferred to the nearest handler with a type that matches the type of the exception thrown. If no matching handler is directly found within the handlers for a try block in which the exception is thrown, the search for a matching handler continues to dynamically search for handlers in the surrounding try blocks of the same thread. The C++ Standard, [except.handle], paragraph 9 [ISO/IEC 14882-2014], states the following:  If no matching handler is found, the function std::terminate() is called; whether or not the stack is unwound before this call to std::terminate() is implementation-defined.  The default terminate handler called by std::terminate() calls std::abort(), which abnormally terminates the process. When std::abort() is called, or if the implementation does not unwind the stack prior to calling std::terminate(), destructors for objects may not be called and external resources can be left in an indeterminate state. Abnormal process termination is the typical vector for denial-of-service attacks. For more information on implicitly calling std::terminate(), see ERR50-CPP. Do not abruptly terminate the program.  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.  As per ERR50-CPP-EX1, a program that encounters an unrecoverable exception may explicitly catch the exception and terminate, but it may not allow the exception to remain uncaught. One possible solution to comply with this rule, as well as with ERR50-CPP, is for the main() function to catch all exceptions. While this does not generally allow the application to recover from the exception gracefully, it does allow the application to terminate in a controlled fashion. |

| **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):** **Architect and design for security policies.** It’s important that any system is able to appropriately handle inevitable exceptions in code to prevent more severe vulnerabilities.  **Adopt a secure coding standard.** Consistent exception handling, using the same system, improves the security of a system. |
| --- |

**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) | 7.3p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **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) | 2022.2 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | **4035, 4036, 4037** |  |
| [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** | **Close files when they are no longer needed** |
| --- | --- | --- |
| Input/Output | [STD-008- CPP] | A call to the std::basic\_filebuf<T>::open() function must be matched with a call to std::basic\_filebuf<T>::close() before the lifetime of the last pointer that stores the return value of the call has ended or before normal program termination, whichever occurs first.  Note that std::basic\_ifstream<T>, std::basic\_ofstream<T>, and std::basic\_fstream<T> all maintain an internal reference to a std::basic\_filebuf<T> object on which open() and close() are called as needed. Properly managing an object of one of these types (by not leaking the object) is sufficient to ensure compliance with this rule. Often, the best solution is to use the stream object by value semantics instead of via dynamic memory allocation, ensuring compliance with MEM51-CPP. Properly deallocate dynamically allocated resources. However, that is still insufficient for situations in which destructors are not automatically called. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::fstream object file is constructed. The constructor for std::fstream calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. Consequently, the underlying std::basic\_filebuf<T> object maintained by the object is not properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  std::terminate();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  file.close();  if (file.fail()) {  // Handle error  }  std::terminate();  } |

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

| **Principles(s):** **Adhere to the principle of least privilege.** Files should only be available during the lifespan of applicable code. The system should not maintain access to the file when it is no longer needed. In addition, this improves on performance and decreases the chance of unnecessary locks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **ALLOC.LEAK** | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **DF4786, DF4787, DF4788** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **RH.LEAK** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-FIO51-a** | Ensure resources are 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) | R2023a | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Copy operations must not mutate the source object** |
| --- | --- | --- |
| Object Oriented Programming | [STD-009- CPP] | Copy operations (copy constructors and copy assignment operators) are expected to copy the salient properties of a source object into the destination object, with the resulting object being a "copy" of the original. What is considered to be a salient property of the type is type-dependent, but for types that expose comparison or equality operators, includes any properties used for those comparison operations. This expectation leads to assumptions in code that a copy operation results in a destination object with a value representation that is equivalent to the source object value representation. Violation of this basic assumption can lead to unexpected behavior. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the copy operations for A mutate the source operand by resetting its member variable m to 0. When std::fill() is called, the first element copied will have the original value of obj.m, 12, at which point obj.m is set to 0. The subsequent nine copies will all retain the value 0. |
| #include <algorithm>  #include <vector>    class A {  mutable int m;    public:  A() : m(0) {}  explicit A(int m) : m(m) {}    A(const A &other) : m(other.m) {  other.m = 0;  }    A& operator=(const A &other) {  if (&other != this) {  m = other.m;  other.m = 0;  }  return \*this;  }    int get\_m() const { return m; }  };    void f() {  std::vector<A> v{10};  A obj(12);  std::fill(v.begin(), v.end(), obj);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the copy operations for A no longer mutate the source operand, ensuring that the vector contains equivalent copies of obj. Instead, A has been given move operations that perform the mutation when it is safe to do so. |
| #include <algorithm>  #include <vector>    class A {  int m;    public:  A() : m(0) {}  explicit A(int m) : m(m) {}    A(const A &other) : m(other.m) {}  A(A &&other) : m(other.m) { other.m = 0; }    A& operator=(const A &other) {  if (&other != this) {  m = other.m;  }  return \*this;  }    A& operator=(A &&other) {  m = other.m;  other.m = 0;  return \*this;  }    int get\_m() const { return m; }  };    void f() {  std::vector<A> v{10};  A obj(12);  std::fill(v.begin(), v.end(), obj);  } |

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

| **Principles(s):** **Keep it simple.** If an object might change during a copy operation, this adds unnecessary complexity to the system. There may be unexpected behavior which could cause problems when future developers who use the object are not aware of this behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.FUNCS.COPINC** | Copy Operation Parameter Is Not const |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++4075** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2023.1 | **CERT.OOP.COPY\_MUTATES** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-OOP58-a** | Copy operations must not mutate the source object |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: OOP58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop58cpp.html) | Checks for copy operation modifying source operand (rule partially covered) |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | **4075** |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Ensure actively held locks are released on exceptional conditions** |
| --- | --- | --- |
| Concurrency | [STD-010- CPP] | Mutexes that are used to protect accesses to shared data may be locked using the lock() member function and unlocked using the unlock() member function. If an exception occurs between the call to lock() and the call to unlock(), and the exception changes control flow such that unlock() is not called, the mutex will be left in the locked state and no critical sections protected by that mutex will be allowed to execute. This is likely to lead to deadlock. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example manipulates shared data and protects the critical section by locking the mutex. When it is finished, it unlocks the mutex. However, if an exception occurs while manipulating the shared data, the mutex will remain locked. |
| #include <mutex>    void manipulate\_shared\_data(std::mutex &pm) {  pm.lock();    // Perform work on shared data.    pm.unlock();  } |

| **Compliant Code** |
| --- |
| This compliant solution catches any exceptions thrown when performing work on the shared data and unlocks the mutex before rethrowing the exception. |
| #include <mutex>    void manipulate\_shared\_data(std::mutex &pm) {  pm.lock();  try {  // Perform work on shared data.  } catch (...) {  pm.unlock();  throw;  }  pm.unlock(); // in case no exceptions occur  } |

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

| **Principles(s):** **Adhere to the principle of least privilege.** Objects should only be available during the lifespan of applicable code. The system should not maintain access to the object when it is no longer needed. In addition, this improves on performance and decreases the chance of unnecessary locks. If exceptions cause sustained locks it could introduce a vulnerability to the system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **CONCURRENCY.LOCK.NOUNLOCK** | Missing Lock Release |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++5018** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-CON51-a** | Do not call lock() directly on a mutex |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **5018** |  |

### 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



The standards above are coding standards. As such, they should be caught early on in the development process. Ideally, they will be caught during an automated scan during the build process. As a developer checks code into source control it will be compiled and built on a development server and during this build any missed standards will be reported so that the development team can correct them before the code moves any further through the system.

Any standards that cannot be adequately captured during automated builds should be checked for during the verify and test portion of the development cycle.

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

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Sensitive data that is being stored, for example in a database, must be encrypted. Encryption should happen on the server with a private key so that only the server itself can decrypt the data. |
| Encryption at flight | Sensitive data that is being sent from one system or part of a system to another system (also often called “in transit”) must be encrypted before being sent. A shared key or certificate can be used by the sender and receiver so that the data cannot be discovered if it is intercepted while in transit. This will often be over the internet but could also be when it is sent by other methods. |
| Encryption in use | When sensitive data is in use by an application it should also be encrypted. This can prevent physical attacks or other breaches that might give an attacker access to an application or direct access to a machine’s memory. The user’s security can be checked at a frequency that is appropriate for the data being accessed. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is when the system verifies who a user is. The user must be authenticated. If the system does not know for certain who a user is then it cannot decide to what that user should and should not have access. |
| Authorization | Once the system knows who a user is it is important to know to what that user can have access. This is authorization. Given knowledge of who a user is and what resource they are requesting, should they be granted access. |
| Accounting | Accounting or auditing is tracking which resources are used, by which users, and what those users have done with the resource. This is an important part of security, especially when it comes to post-incident forensics. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 03/19/2023 | Added 10 principles and coding standards | Thomas R. Hummel |  |
| 1.2 | 04/09/2023 | Expanded on coding standards, added encryption and triple-A standards | Thomas R. Hummel |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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