**Green Pace Developer: Security Policy Guide**



# Green Pace Secure Development Policy

## Contents

[Green Pace Secure Development Policy 0](#_Toc114934358)

[Contents 1](#_Toc114934359)

[Overview 3](#_Toc114934360)

[Purpose 3](#_Toc114934361)

[Scope 3](#_Toc114934362)

[Module Three Milestone 3](#_Toc114934363)

[Ten Core Security Principles 3](#_Toc114934364)

[C/C++ Ten Coding Standards 4](#_Toc114934365)

[Data Type Coding Standard 5](#_Toc114934366)

[Data Value Coding Standard 7](#_Toc114934367)

[String Correctness Coding Standard 9](#_Toc114934368)

[SQL Injection Coding Standard 11](#_Toc114934369)

[Memory Protection Coding Standard 13](#_Toc114934370)

[Assertions Coding Standard 14](#_Toc114934371)

[Exceptions Coding Standard 15](#_Toc114934372)

[OOP Coding Standard 16](#_Toc114934373)

[File Input/Out Coding Standard 17](#_Toc114934374)

[Data Coding Standard 18](#_Toc114934375)

[Defense-in-Depth Illustration 19](#_Toc114934376)

[Project One 19](#_Toc114934377)

[Revise the C/C++ Standards 19](#_Toc114934378)

[Risk Assessment 19](#_Toc114934379)

[Automated Detection 19](#_Toc114934380)

[Automation 19](#_Toc114934381)

[Summary of Risk Assessments 20](#_Toc114934382)

[Create Policies for Encryption and Triple A 20](#_Toc114934383)

[Map the Principles 21](#_Toc114934384)

[Audit Controls and Management 22](#_Toc114934385)

[Enforcement 22](#_Toc114934386)

[Exceptions Process 22](#_Toc114934387)

[Distribution 23](#_Toc114934388)

[Policy Change Control 23](#_Toc114934389)

[Policy Version History 23](#_Toc114934390)

[Appendix A Lookups 23](#_Toc114934391)

[Approved C/C++ Language Acronyms 23](#_Toc114934392)

## 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 | Prevent improperly formatted data from entering an information system. Input data should be both syntactically and semantically correct. That is, input data validation should be enforced to both the correct syntax of structured fields and enforce the correctness of their values. |
| 1. Heed Compiler Warnings | Update your compiler to display the highest warning levels. Update your code to clear those warnings. Use static and dynamic analysis to detect other software defects and update your code accordingly. |
| 1. Architect and Design for Security Policies | Design your architecture and software design to enforce security policies. Ensure that the design of your architecture is based on security policies and not an afterthought. Not only does this make your system more secure but saves you time and costs later when implementing new functionality or security policies. |
| 1. Keep It Simple | Keep the software architecture and design as simple as possible. The more complex a system gets, the harder it is to maintain and ensure the system is secure. Complex designs increase the likelihood of software security defects are made. |
| 1. Default Deny | Deny any access request by default and only allow access based on permissions instead of exclusion. That is, the default decision is to deny unless the requester has the appropriate permissions for access. |
| 1. Adhere to the Principle of Least Privilege | Employ the principle of least privilege in your design and only allow access/permissions to users that they need to perform their duties. For example, an administrative assistant may only need to permissions to create and updated some form of data, but not have the permission to delete it. |
| 1. Sanitize Data Sent to Other Systems | Sanitize data sent to other systems such as shells and databases to prevent attacks such as SQL injection. These kinds of attacks may be able to use functionality provided by the system to perform a process that is understood by the system, but not intended for in its design. For example, an attacker may perform an injection attack by returning the entire contents and a user database. |
| 1. Practice Defense in Depth | Make use of multiple different layers of defense so that if one layer fails to prevent an attack or security flaw, then another security layer can catch and prevent it. For example, using multiple security layers such as a firewall, frequently installing software updates and patches, installing antivirus and antimalware software, using secure runtime environments, and using the other security techniques outlined here create a combined security net that can detect and prevent different kinds of attacks. |
| 1. Use Effective Quality Assurance Techniques | New and modified code should always be tested and have more than one pair of eyes on it. Unit tests, automated tests, and penetration tests are a few testing strategies that should be utilized. Code audits, both internal and external, should be incorporated to ensure that multiple different perspectives are present in quality assurance. |
| 1. Adopt a Secure Coding Standard | The secure coding principles and standards outlined in this document should be adopted across the organization to minimize risk and ensure that more secure and maintainable software is developed by the organization. |

### 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.

#### Data Type Coding Standard

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not define a C-style variadic function. A C-style variadic function is unable to check the type safety of arguments being passed or unable to check that the number of arguments being passed matches the function definition. |

| **Noncompliant Code** |
| --- |
| The below code block is a C-style variadic function that adds a series of integers together until the value 0 is found. Calling this function without passing a 0 value, or passing any data type other than int, results in undefined behavior. |
| #include <cstdarg>    int add(int first, int second, ...) {  int r = first + second;  va\_list va;  va\_start(va, second);  while (int v = va\_arg(va, int)) {  r += v;  }  va\_end(va);  return r;  } |

| **Compliant Code** |
| --- |
| The below compliant solution makes use of a built in add() function that that doe not result in undefined behavior if the passed arguments don’t end with a 0 value. As this is a templated function, if any other data type other than int is passed as an argument it also does not result in undefined behavior. |
| #include <type\_traits>    template <typename Arg, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  int add(Arg f, Arg s) { return f + s; }    template <typename Arg, typename... Ts, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  int add(Arg f, Ts... rest) {  return f + add(rest...);  } |

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

| **Principles(s):**   * Validate Input Data: Ensure user input is sanitized and input data is assigned to correct data types. * Heed Compiler Warnings: Compiler will warn of potential data type mismatches in function calls. * Sanitize Data Sent to Other Systems: Ensure all user input is sanitized before assignment and sent to other systems. * Practice Defense in Depth: Adopting an additional layer of security that overlaps other layers helps create a more secure system. * Adopt a Secure Coding Standard: adopting a secure coding standard upfront helps decrease overall vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **function-ellipsis** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-dcl50-cpp | Checked by clang-tidy. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **LANG.STRUCT.ELLIPSIS** | Ellipsis |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++2012, C++2625** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2022.3 | **MISRA.FUNC.VARARG** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **41 S** | Fully Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL50-a** | Functions shall not be defined with a variable number of arguments |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl50cpp.html) | Checks for function definition with ellipsis notation (rule fully covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2012, 2625** |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | **function-ellipsis** | Fully checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**FunctionEllipsis**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-923) |  |

#### Data Value Coding Standard

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-C] | Ensure that unsigned integer operations do not wrap. Unsigned integer operations can occur when a resulting integer value cannot be represented by the declared integer data type. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example can result in unsigned integer wrapping for sufficiently large int values. This may lead to undefined behavior or allow for some other kind of vulnerability the can be exploited elsewhere. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The below compliant code makes use of a precondition test to guarantee that the sum of the two unsigned int values do not exceed the establish limit of the unsigned integer maximum value. |
| #include <limits.h>    void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum;  if (UINT\_MAX - ui\_a < ui\_b) {  /\* Handle error \*/  } else {  usum = ui\_a + ui\_b;  }  /\* ... \*/  } |

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

| **Principles(s):**   * Validate Input Data: Ensure input data is checked against limits.h as needed to prevent wrapping. * Heed Compiler Warnings: Some compilers will warn of potential unsigned integer wrapping. * Practice Defense in Depth: Adopting an additional layer of security that overlaps other layers helps create a more secure system. * Adopt a Secure Coding Standard: adopting a secure coding standard upfront helps decrease overall vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | **integer-overflow** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | **CertC-INT30** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **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 |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2022.3 | **C2910, C2911, C2912, C2913, C3383, C3384, C3385, C3386**  **C++2910, C++2911, C++2912, C++2913** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2022.3 | **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) | 2022.1 | **CERT\_C-INT30-a** **CERT\_C-INT30-b** **CERT\_C-INT30-c** | Avoid integer overflows 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) | R2022b | [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. |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **2910 [C], 2911 [D], 2912 [A],**  **2913 [S], 3383, 3384, 3385, 3386** | Partially implemented |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2910, 2911, 2912, 2913** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.20 | [**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. |

#### String Correctness Coding Standard

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator. Copying data to a buffer that is not large enough to hold that data results in buffer overflow. C-style strings require a null character to indicate the end of the string, while C++ *std::string* does not have this requirement. |

| **Noncompliant Code** |
| --- |
| Because the below code initializes a finite char value and allows for unbounded input into that value, this code can easily result in buffer overflow if the user provides more input than can be fit into the char value. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| To ensure that data is not truncated or results in buffer overflow, *std::string* should be used instead of a bounded char array. As *std::string* is not bounded, these results will not occur. |
| #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):**   * Architect and Design for Security Policies: Do not use outdated functionality and/or use cases. * Heed Compiler Warnings: Some compilers may warn of possible buffer overflow. * Keep It Simple: Use tools/libraries that are available to you. Do not reinvent the wheel. Don’t use a char array when using a string would work as well. * Practice Defense in Depth: Adopting an additional layer of security that overlaps other layers helps create a more secure system. * Adopt a Secure Coding Standard: adopting a secure coding standard upfront helps decrease overall vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++2835, C++2836, C++2839, C++5216** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.3 | **NNTS.MIGHT** **NNTS.TAINTED** **NNTS.MUST** **SV.UNBOUND\_STRING\_INPUT.CIN** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation * Insufficient destination buffer size   Rule partially covered. |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3519**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3519) |  |

#### SQL Injection Coding Standard

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C] | Exclude user input from format strings. Never use data that has not been sanitized. An attacker can taint data that could then be used in such a way that could crash a vulnerable process, view the contents of the stack, view memory content, or write to an arbitrary memory location. |

| **Noncompliant Code** |
| --- |
| The below noncompliant function accepts a string parameter provided by the user. This is untrusted data that is provided by an unauthenticated user. The amount of memory that is allocated is determined by the length of the user input, which is a vulnerability that can be exploited. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {  /\* Handle error \*/  } else if (ret >= len) {  /\* Handle truncated output \*/  }  fprintf(stderr, msg);  free(msg);  } |

| **Compliant Code** |
| --- |
| The below code does not allocated memory based on user input and passes the parameter through the fprintf() function, rather than placing it into memory as part of a formatted string. This also prevents a formatted string vulnerability. |
| #include <stdio.h>    void incorrect\_password(const char \*user) {  static const char msg\_format[] = "%s cannot be authenticated.\n";  fprintf(stderr, msg\_format, user);  } |

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

| **Principles(s):**   * Validate Input Data: Ensure untrusted data is validated for type correctness, is sanitized, and does not contain malicious SQL statements. * Architect and Design for Security Policies: Design for database queries that disallow for SQL injection attacks. * Default Deny: Use prepared statements to deny by default any statement that is not expected under normal operation. * Sanitize Data Sent to Other Systems: Validate and sanitize user input before using it in a database query. Also use prepared statements for database queries. * Practice Defense in Depth: Adopting an additional layer of security that overlaps other layers helps create a more secure system. * Adopt a Secure Coding Standard: adopting a secure coding standard upfront helps decrease overall 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 via stubbing/taint analysis |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-FIO30** | Partially implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **IO.INJ.FMT MISC.FMT** | Format string injection Format string |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **TAINTED\_STRING** | Implemented |
| [GCC](https://wiki.sei.cmu.edu/confluence/display/c/GCC) | 4.3.5 |  | Can detect violations of this rule when the -Wformat-security flag is used |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2022.3 | **C4916, C4917, C4918**  **C++4916, C++4917, C++4918** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2022.3 | **SV.FMTSTR.GENERIC SV.TAINTED.FMTSTR** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **86 D** | Partially Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | **CERT\_C-FIO30-a** **CERT\_C-FIO30-b** **CERT\_C-FIO30-c** | Avoid calling functions printf/wprintf with only one argument other than string constant Avoid using functions fprintf/fwprintf with only two parameters, when second parameter is a variable Never use unfiltered data from an untrusted user as the format parameter |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **592** | Partially supported: reports non-literal format strings |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022b | [CERT C: Rule FIO30-C](https://www.mathworks.com/help/bugfinder/ref/certcrulefio30c.html) | Checks for tainted string format (rule partially covered) |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | **4916, 4917, 4918** |  |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **4916, 4917, 4918** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.20 | [**V618**](https://pvs-studio.com/en/docs/warnings/v618/) |  |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/c/Splint) | 3.1.1 |  |  |

#### Memory Protection Coding Standard

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Properly deallocate dynamically allocated resources. C++ provides multiple ways to allocate and deallocate memory resources, such as new, new[], delete, and delete[]. It is important to allocate and deallocate memory when needed, no longer needed, and to do so properly. |

| **Noncompliant Code** |
| --- |
| In the below example two allocations are attempted in the try block and if they fail, the catch block then attempts to free those allocations. If the try block fails, then the allocations never take place and the catch block tries to deallocated data that was never allocated, resulting in undefined behavior. |
| #include <new>    void f() {  int \*i1, \*i2;  try {  i1 = new int;  i2 = new int;  } catch (std::bad\_alloc &) {  delete i1;  delete i2;  }  } |

| **Compliant Code** |
| --- |
| The below example fixes the above example by first initializing the two pointers (before the try block) to nullptr, which is a valid value to pass to the delete operator. |
| #include <new>    void f() {  int \*i1 = nullptr, \*i2 = nullptr;  try {  i1 = new int;  i2 = new int;  } catch (std::bad\_alloc &) {  delete i1;  delete i2;  }  } |

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

| **Principles(s):**   * Heed Compiler Warnings: some compilers may provide potential warnings when objects and resources do not properly deallocate their resources. * Architect and Design for Security Policies: design your classes to automatically allocate and deallocate memory when possible. * Keep It Simple: automate the allocation/deallocation of classes and memory where applicable. * Practice Defense in Depth: Adopting an additional layer of security that overlaps other layers helps create a more secure system. * Adopt a Secure Coding Standard: adopting a secure coding standard upfront helps decrease overall 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=222953724) | 20.10 | **invalid\_dynamic\_memory\_allocation dangling\_pointer\_use** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MEM51** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks -Wmismatched-new-delete clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **ALLOC.FNH ALLOC.DF ALLOC.TM ALLOC.LEAK** | Free non-heap variable Double free Type mismatch Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++2110, C++2111, C++2112, C++2113, C++2118, C++3337, C++3339, C++4262, C++4263, C++4264** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2022.3 | **CL.FFM.ASSIGN** **CL.FFM.COPY** **CL.FMM** **CL.SHALLOW.ASSIGN** **CL.SHALLOW.COPY** **FMM.MIGHT** **FMM.MUST** **FNH.MIGHT** **FNH.MUST** **FUM.GEN.MIGHT** **FUM.GEN.MUST** **UNINIT.CTOR.MIGHT** **UNINIT.CTOR.MUST** **UNINIT.HEAP.MIGHT** **UNINIT.HEAP.MUST** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-MEM51-a** **CERT\_CPP-MEM51-b** **CERT\_CPP-MEM51-c** **CERT\_CPP-MEM51-d** | Use the same form in corresponding calls to new/malloc and delete/free Always provide empty brackets ([]) for delete when deallocating arrays Both copy constructor and copy assignment operator should be declared for classes with a nontrivial destructor Properly deallocate dynamically allocated resources |
| [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) | R2022b | [CERT C++: MEM51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem51cpp.html) | Checks for:   * Invalid deletion of pointer * Invalid free of pointer * Deallocation of previously deallocated pointer   Rule partially covered. |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2110, 2111, 2112, 2113, 2118,**  **3337, 3339, 4262, 4263, 4264** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.20 | [**V515**](https://pvs-studio.com/en/docs/warnings/v515/), [**V554**](https://pvs-studio.com/en/docs/warnings/v554/), [**V611**](https://pvs-studio.com/en/docs/warnings/v611/), [**V701**](https://pvs-studio.com/en/docs/warnings/v701/), [**V748**](https://pvs-studio.com/en/docs/warnings/v748/), [**V773**](https://pvs-studio.com/en/docs/warnings/v773/), [**V1066**](https://pvs-studio.com/en/docs/warnings/v1066/) |  |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S1232**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-1232) |  |

#### Assertions Coding Standard

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Avoid information leakage when passing a class object across a trust boundary. When passing a pointer to a class object across a trust boundary, your must ensure that the padding bits of an object do not contain sensitive information. |

| **Noncompliant Code** |
| --- |
| In the below noncompliant code example, an object is copied from a kernel space to a user space over a trust boundary. This object may contain padding bits that may contain sensitive information that may be leaked when the data is copied to the user space. |
| #include <cstddef>    struct test {  int a;  char b;  int c;  };    // Safely copy bytes to user space  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void do\_stuff(void \*usr\_buf) {  test arg{};    arg.a = 1;  arg.b = 2;  arg.c = 3;    copy\_to\_user(usr\_buf, &arg, sizeof(arg));  } |

| **Compliant Code** |
| --- |
| The below code fixes this by using static assertions to ensure that the object contains no padding bits. These assertions take a constant expression and an error message. The assertion is evaluated at compile time and terminated the compilation if false and displays the error message. |
| #include <cstddef>    struct test {  int a;  char b;  char padding\_1, padding\_2, padding\_3;  int c;    test(int a, char b, int c) : a(a), b(b),  padding\_1(0), padding\_2(0), padding\_3(0),  c(c) {}  };  // Ensure c is the next byte after the last padding byte.  static\_assert(offsetof(test, c) == offsetof(test, padding\_3) + 1,  "Object contains intermediate padding");  // Ensure there is no trailing padding.  static\_assert(sizeof(test) == offsetof(test, c) + sizeof(int),  "Object contains trailing padding");        // Safely copy bytes to user space.  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void do\_stuff(void \*usr\_buf) {  test arg{1, 2, 3};  copy\_to\_user(usr\_buf, &arg, sizeof(arg));  } |

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

| **Principles(s):**   * Sanitize Data Sent to Other Systems: ensure class object data sent to other systems does not contain padding bits with sensitive information. * Architect and Design for Security Policies: design for data in transit to be stateless and encrypted where necessary. * Practice Defense in Depth: Adopting an additional layer of security that overlaps other layers helps create a more secure system. * Adopt a Secure Coding Standard: adopting a secure coding standard upfront helps decrease overall vulnerabilities. |
| --- |

**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/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL55** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **MISC.PADDING.POTB** | Padding Passed Across a Trust Boundary |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++4941, C++4942, C++4943** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL55-a** | A pointer to a structure should not be passed to a function that can copy data to the user space |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL55-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl55cpp.html) | Checks for information leakage due to structure padding (rule partially covered) |

#### Exceptions Coding Standard

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions. When an exception is thrown, control is transferred to the nearest exception handler that matches the type of exception thrown. If no matching handler is found, the function *std::terminate()* is called and abnormally terminates the process. All exceptions thrown by an application must be caught by a matching exception handler to prevent this. |

| **Noncompliant Code** |
| --- |
| The below noncompliant code does not catch exceptions thrown by *throwing\_func*(). This results in *std::terminate()* being called. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In this example, the main function catches all exceptions thrown, which ensures that the stack is unwound up to the main function. |
| 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):**   * Heed Compiler Warnings: compilers will warn for potential uncaught exceptions. * Architect and Design for Security Policies: handling all potential exceptions leads to an overall more secure design. * Use Effective Quality Assurance Techniques: proper QA involves ensuring that unexpected input/behavior in a program is handled gracefully. * Practice Defense in Depth: Adopting an additional layer of security that overlaps other layers helps create a more secure system. * Adopt a Secure Coding Standard: adopting a secure coding standard upfront helps decrease overall vulnerabilities. |
| --- |

**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) | 20.10 | **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) | 7.1p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.3 | **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) | 2022.1 | **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) | R2022b | [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) | 20.10 | **main-function-catch-all early-catch-all** | Partially checked |

#### OOP Coding Standard

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| OOP | [STD-008-CPP] | Do not invoke virtual functions from constructors or destructors. As the order of construction starts from base classes and moves to more derived classes calling a virtual function from a derived class while still under construction in the base class is dangerous. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example the base class B attempts to allocate and deallocate its resources by calls to virtual functions from the constructor and destructor. However, the B::B() constructor calls B::seize() instead of D::seize(). The result of this code is that no derived classes will be seized or released during construction or destruction of object type D. |
| struct B {  B() { seize(); }  virtual ~B() { release(); }    protected:  virtual void seize();  virtual void release();  };  struct D : B {  virtual ~D() = default;    protected:  void seize() override {  B::seize();  // Get derived resources...  }    void release() override {  // Release derived resources...  B::release();  }  }; |

| **Compliant Code** |
| --- |
| To fix this, the constructors and destructors call a nonvirtual private member function, resulting in each class being responsible for its own creating and deleting of resources. |
| class B {  void seize\_mine();  void release\_mine();    public:  B() { seize\_mine(); }  virtual ~B() { release\_mine(); }    protected:  virtual void seize() { seize\_mine(); }  virtual void release() { release\_mine(); }  };    class D : public B {  void seize\_mine();  void release\_mine();    public:  D() { seize\_mine(); }  virtual ~D() { release\_mine(); }    protected:  void seize() override {  B::seize();  seize\_mine();  }    void release() override {  release\_mine();  B::release();  }  }; |

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

| **Principles(s):**   * Heed Compiler Warnings: some compilers may warn when a virtual function declaration should be used. * Architect and Design for Security Policies: design and declare your classes and derived classes in the most secure way possible. * Keep It Simple: design your classes and derived classes well for a simpler system that is easier to maintain. * Practice Defense in Depth: Adopting an additional layer of security that overlaps other layers helps create a more secure system. * Adopt a Secure Coding Standard: adopting a secure coding standard upfront helps decrease overall vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **virtual-call-in-constructor invalid\_function\_pointer** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-OOP50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-alpha.cplusplus.VirtualCall | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **LANG.STRUCT.VCALL\_IN\_CTOR**  **LANG.STRUCT.VCALL\_IN\_DTOR** | Virtual Call in Constructor  Virtual Call in Destructor |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++4260, C++4261, C++4273, C++4274, C++4275, C++4276, C++4277, C++4278, C++4279, C++4280, C++4281, C++4282** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2022.3 | **CERT.OOP.CTOR.VIRTUAL\_FUNC** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **467 S, 92 D** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-OOP50-a** **CERT\_CPP-OOP50-b** **CERT\_CPP-OOP50-c** **CERT\_CPP-OOP50-d** | Avoid calling virtual functions from constructors Avoid calling virtual functions from destructors Do not use dynamic type of an object under construction Do not use dynamic type of an object under destruction |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: OOP50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop50cpp.html) | Checks for virtual function call from constructors and destructors (rule fully covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **4260, 4261, 4273, 4274, 4275, 4276, 4277, 4278, 4279, 4280, 4281, 4282** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.20 | [**V1053**](https://pvs-studio.com/en/docs/warnings/v1053/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | **virtual-call-in-constructor** | Fully checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S1699**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-1699) |  |

#### File Input/Out Coding Standard

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| File Input/Output | [STD-009-CPP] | Close files when they are no longer needed. A call to the *std::basic\_filebuf<T>::open()* function must be matched with a call to *std::basic\_filebuf<T>::close() before* normal program termination. Failing to do so may result in leaking to object in memory. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an *std::fstream* file object is constructed and not closed before *std::terminate()* is called. As *std::terminate()* does not call destructors, the file object is not closed properly. |
| #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** |
| --- |
| To fix the above example *std::fstream::close()* is called before *std::terminate()* which ensures that the file is closed properly. |
| #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):**   * Heed Compiler Warnings: some compilers will warn when an open() call is not matched with a close() call for file I/O operations. * Keep It Simple: open a file only when you need it and close it immediately when no longer in use. Do not keep it open even if it is likely to be used somewhere else in the program. * Practice Defense in Depth: Adopting an additional layer of security that overlaps other layers helps create a more secure system. * Adopt a Secure Coding Standard: adopting a secure coding standard upfront helps decrease overall vulnerabilities. |
| --- |

**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) | 7.1p0 | **ALLOC.LEAK** | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++4786, C++4787, C++4788** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.3 | **RH.LEAK** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **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) | R2022b | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |

#### Data Coding Standard

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data | [STD-010-CPP] | Detect errors when converting a string to a number. Parsing an integer or floating-point number from a string can result in many different errors. The string may not contain a number or the number may be out of range for the receiving data type. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, multiple numeric values are converted from the input stream. If the text received from the input stream cannot be converted to numeric value represented by the int data type, the result values stored into the variables may be unexpected. |
| #include <iostream>    void f() {  int i, j;  std::cin >> i >> j;  // ...  } |

| **Compliant Code** |
| --- |
| To fix the above code, the *std::istream::failure* exception is caught for any conversion failure arising from the input stream. It is worth noting that while this solution allows the exception to be handled, it cannot distinguish between valid and invalid values and must assume that all values from the input stream are invalid. |
| #include <iostream>    void f() {  int i, j;    std::cin.exceptions(std::istream::failbit | std::istream::badbit);  try {  std::cin >> i >> j;  // ...  } catch (std::istream::failure &E) {  // Handle error  }  } |

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

| **Principles(s):**   * Validate Input Data: validate input by throwing exceptions if there is a data type mismatch. * Heed Compiler Warnings: some compilers may warn of possible type assignment error. * Architect and Design for Security Policies: implement into your design to expect the unexpected and use try-catch blocks to handle unexpected input gracefully. * Default Deny: throw exceptions when unexpected input is received to ensure only correct input is accepted and allow for normal program flow. * Use Effective Quality Assurance Techniques: use exception handling, try-catch blocks, to ensure unexpected input is handled gracefully. * Practice Defense in Depth: Adopting an additional layer of security that overlaps other layers helps create a more secure system. * Adopt a Secure Coding Standard: adopting a secure coding standard upfront helps decrease overall vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR62** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-err34-c | Checked by clang-tidy; only identifies use of unsafe C Standard Library functions corresponding to ERR34-C |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **BADFUNC.ATOF BADFUNC.ATOI BADFUNC.ATOL BADFUNC.ATOLL** | Use of atof Use of atoi Use of atol Use of atoll |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++3161** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.3 | **CERT.ERR.CONV.STR\_TO\_NUM** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-ERR62-a** | The library functions atof, atoi and atol from library stdlib.h shall not be used |

### 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.

Security and automation tools should be present throughout the DevOps process for it to truly be DevSecOps. Therefore, it makes sense to automate as much security enforcement and compliance in the DevOps pipeline as possible. With regards to automation tools, some areas of the DevOps pipeline can be automated. On the pre-production side, parts of design phase can be automated by utilizing the OWASP dependency checker. This will list all known vulnerabilities of each dependency in your design. Test driven development is also an important part of the design phase. In the build phase, static analysis tools should be used in the developer’s IDE and the compiler flags for errors and warnings should be set to their highest level. This will help developers catch common errors and warnings in their code that could lead to a security vulnerability. In the verify and test phase, automated unit and integration tests should be performed before the modified codebase is released to production. This will likely be done during off hours, typically at night.

On the production side, more automatic security tests should be performed in the transition and health check phase. Specifically, security and configuration settings should be automatically deployed. Automatic network and penetration testing should also be continuously performed. During the monitor and detect phase, logs should be collected from all areas and automatically analyzed. If a threat is detected, it should trigger an automatic response. This leads into the response phase, where some threat is detected automatically, it should trigger either a notification to the appropriate team(s) for action or trigger some automatic defenses. These automatic defenses could be blocking the attacks, turning off certain services, or rolling back a new release to an older, more stable, version.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| [STD-001-CPP] | High | Probable | Medium | P12 | L1 |
| [STD-002-C] | High | Likely | High | P9 | L2 |
| [STD-003-CPP] | High | Likely | Medium | P18 | L1 |
| [STD-004-C] | High | Likely | Medium | P18 | L1 |
| [STD-005-CPP] | High | Likely | Medium | P18 | L1 |
| [STD-006-CPP] | Low | Unlikely | High | P1 | L3 |
| [STD-007-CPP] | Low | Probable | Medium | P4 | L3 |
| [STD-008-CPP] | Low | Unlikely | Medium | P2 | L3 |
| [STD-009-CPP] | Medium | Unlikely | Medium | P4 | L3 |
| [STD-010-CPP] | Medium | Unlikely | Medium | P4 | L3 |

### 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 | Encryption for data at rest means encrypting your data when it is being stored on a server or in some other kind of physical or cloud storage. The most used type of encryption used for data at rest is symmetric encryption. The most popular symmetric encryption cipher is the Advanced Encryption Standard (AES). Symmetric ciphers like AES use a private key to encrypt and decrypt data. All confidential information, no matter where it is stored, should be encrypted with AES. The private keys should also be stored in a different location, ideally in a secure key store. |
| Encryption at flight | Encryption for data in flight means encrypting data that is traveling over a network, typically an unsecure network such as the public Internet. This is the most widely used case of encryption as encryption in flight is the bedrock of our Internet-based economy. The most used type of encryption for data in flight is asymmetric encryption. Asymmetric encryption is when data is encrypted using a public key and then decrypted using a private key. Typically, when a user sends confidential information over the Internet, the client-side code will encrypt their data using the public key and send the encrypted data over the Internet. Without the private key, it is impossible to decrypt the data (even with the public key). The private key is kept secure and private on the server back end and is used to decrypt data received from the client-side. |
| Encryption in use | Encryption in use is when data is always encrypted, regardless of where it comes from or whether the data is at rest or in flight. This means that data is, ideally, never left unsecured – even when it is ‘in use’. This is accomplished by utilizing some form of identity and access management, typically through the administration of role-based access control (RBAC). RBAC should be administer through a central governing body, with access to unencrypted data highly controlled. Access the unencrypted data should be denied by default. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process to determine that user is who they say they are. Any secure system requires a user to have an ‘account’ to be granted access to it. You are ‘authenticating’ the credentials of their account. Typically, this comes in the form of verifying a combination of their username or email, password, and MFA token matches what is on file. Once verified, the user has been ‘authenticated’. |
| Authorization | Once authenticated, the user has confirmed they are who they say they are. Now, we need to determine what they are ‘authorized’ to do. Typically, this is done through some form of role-based access control which should adhere to the principle of least privilege. In other words, an account may have one or more roles tied to it, determining what information they’re allowed to see or actions they’re allowed to take. Example permissions might include create, read, update, or delete. Under the principle of least privilege, every account should have a role with the bare minimum number of permissions for them to perform their duties. |
| Accounting | Once a user has been authenticated and has been authorized to access certain data/systems, the activity of that user should be logged. Ideally, everything the user does should be logged. Whenever a user takes an action, at least the following should be logged: what access was requested, what user is requesting it, whether access was granted or not, what role(s) the user possesses, and what actions were performed. In the event of a security breach, these logs will be reviewed to help determine how a bad actor gained unauthorized access by tracing the logs, replicating their steps, until the defect is found. |

**\***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 | 09/24/2022 | Added security principles coding standards | Austin Fuchs |  |
| 1.2 | 10/09/2022 | Added risk assessment, automated security tools, and encryption and triple AAA security policies | Austin Fuchs |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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