Green Pace Developer: Security Policy Guide Template



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validating input data is a common strategy used to sanitize user data before it is used in the program. By doing this, it ensures that the program is only letting in data that is appropriate and usable. If anything out of bounds is accepted, it can lead to unexpected results and vulnerabilities such as SQL injection, command injection, and cross-site scripting. |
| 1. Heed Compiler Warnings | Heeding compiler warnings means to take all types of warning seriously, no matter how big or small. Sometimes a program will display some warning but will still compile, which probably means that some library has deprecated or something similar. However, not resolving these warning can lead to having to deal with bugs later down the road and having unreadable and unmaintainable code. |
| 1. Architect and Design for Security Policies | Designing and architecting software so it adheres to security policies is good for making sure it is secure, and the data is being protected. |
| 1. Keep It Simple | Keeping software simple can lower the chances of introducing errors and bugs and can increase the maintainability. Doing so also keeps time and money costs low, as more complex systems require more developers and longer working times to keep up with the security measures. |
| 1. Default Deny | Default deny means that all users have denied access to the system by default unless otherwise given permission. This ensures that no users are being explicitly excluded and are not accidentally given access privileges when they are not supposed to. |
| 1. Adhere to the Principle of Least Privilege | All users should only be given the right amount of privilege in order to successfully complete the tasks that they need done. There is no circumstance where a user should be given more access privileges than necessary. |
| 1. Sanitize Data Sent to Other Systems | Similar to sanitizing input data from users, data being sent to other systems should also be sanitized. This way, it doesn’t introduce security vulnerabilities that malicious hackers can use to tamper with or view sensitive data such as different types of injection attacks. |
| 1. Practice Defense in Depth | Applying multiple layers of security to protect the systems' sensitive data is a great way to prevent malicious attacks. If one layer of security is broken into, there are still other layers in which the data is being protected. For example, sensitive user input data can be encrypted along with proper input validation. This way, if an attacker is somehow able to access the data with an injection attack, they still need to decipher the encrypted data. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance is all about improving the security and integrity of a software system. Testing the code is a big part of quality assurance, but it’s not the only part. It also puts emphasis on improving the process in which the software is being created, making sure that no major issues are being introduced in the development process. |
| 1. Avoid undefined behavior | Undefined behavior will most always result in a bad outcome or introduce vulnerabilities. Buffer overflow, the use of uninitialized variables, and trying to access an array that is out-of-bounds can all lead to undefined behavior. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Name: Do not define a C-style variadic function  A variadic function is a function that can take a variable number of arguments from a caller. C++ provides two implementations for this, using an ellipsis (...) for the final paramter or using a function parameter pack. |

| **Noncompliant Code** |
| --- |
| This C++ C-style variadic function adds a series of integers together. |
| #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** |
| --- |
| This is a C++ variadic function that uses a function pack to implement the *add()* function to add a series of integers together. This is much safer than the noncompliant code because it does not result in undefined behavior if the list of parameters in not terminated with 0. |
| #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...); } |
| Resources: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL50-CPP.+Do+not+define+a+C-style+variadic+function> |

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

| **Principles(s):** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Function-ellipsis | Fully Checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL50 |  |
| Clang | 3.9 | Cert-dc150-cpp | Checked by clang-tidy |
| CodeSonar | 9.0p0 | LANG.STRUCT.ELLIPSIS | Ellipsis |
| Helix QAC | 2025.1 | C++2012, C++2625 |  |
| Klocwork | 2025.1 | MISRA.FUNC.VARARG |  |
| LDRA tool suite | 9.7.1 | 41 S | Fully Implemented |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-DCL50-a | Functions shall not be defined with a variable number of arguments |
| Polyspace Bug Finder | R2024b | CERT C++: DCL50-CPP | Checks for function definition with ellipsis notation (rule fully covered) |
| RuleChecker | 22.10 | Function-ellispsis | Fully Checked |
| SonarQube C/C++ Plugin | 4.10 | FunctionEllipsis |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data Value | STD-002-CPP | Name: Do not read uninitialized memory  Data must not be read before the initialization of the value |

| **Noncompliant Code** |
| --- |
| In this code example, an uninitialized variable part of a function is trying to print its value to the terminal, which will result in undefined behavior. |
| #include <iostream>   void f() {  int i;  std::cout << i; } |

| **Compliant Code** |
| --- |
| In this example, the variable in the function is being initialized before its value is printed to the terminal. |
| #include <iostream>   void f() {  int i = 0;  std::cout << i; } |
| Resources: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory> |

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

| **Principles(s):** Avoid undefined behavior. Undefined behavior can lead to unexpected outcomes in a program and other bad security vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Uninitialized-read | Partially checked |
| Clang | 3.9 | -Wuninitialized  clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory. |
| CodeSonar | 9.0p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local Uninitialized variable |
| Helix QAC | 2025.1 | DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978 |  |
| Klocwork | 2025.1 | UNINIT.CTOR.MIGHT UNINIT.CTOR.MUST UNINIT.HEAP.MIGHT UNINIT.HEAP.MUST UNINIT.STACK.ARRAY.MIGHT UNINIT.STACK.ARRAY.MUST UNINIT.STACK.ARRAY.PARTIAL.MUST UNINIT.STACK.MIGHT UNINIT.STACK.MUST |  |
| LDRA tool suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially implemented |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-EXP53-a | Avoid use before initialization |
| Parasoft Insure++ |  |  | Runtiime detection |
| Polyspace Bug Finder | R2024b | CERT C++: EXP53-CPP | Checks for:   * Non-initialized variable * Non-initialized pointer   Rule partially covered |
| PVS-Studio | 7.37 | V546, V573, V614, V670, V679, V730, V788, V1007, V1050 |  |
| RuleChecker | 22.10 | Initialized-read | Partially checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| String Correctness | STD-003-CPP | Name: Guarantee that storage for strings has sufficient space for data and the null terminator  Not making sure that strings have sufficient space will result in a buffer overflow, which will produce unexpected changes in other parts of the program. |

| **Noncompliant Code** |
| --- |
| This code will lead to a buffer overflow if the user a string that is more than 12 characters |
| #include <iostream>   void f() {  char buf[12];  std::cin >> buf; } |

| **Compliant Code** |
| --- |
| One of the best solutions to the noncompliant code example is by using the *std::string* data type instead of fixed size arrays. |
| #include <iostream> #include <string>   void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo; } |
| Resources: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator> |

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

| **Principles(s):** Practice Defense in Depth. The reason I chose this principle is because it could be a way of applying an extra layer of security by making sure that all strings have the correct amount of space. That way, there is no way that a string can surpass its assigned memory amount. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Stream-input-char-array | Partially checked + soundly supported |
| CodeSonar | 9.0p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Helix QAC | 2025.1 | C++5216  DF2835, DF2836, DF2839 |  |
| Klocwork | 2025.1 | NNTS.MIGHT  NNTS.TAINTED  NNTS.MUST  SV.UNBOUND\_STRING\_INPUT.CIN |  |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the ‘char’ buffer to store input from ‘std::cin’ |
| Polyspace Bug Finder | R2024b | CERT C++: STR50-CPP | 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 |
| RuleChecker | 22.10 | Stream-input-char-array | Partially checked |
| SonarQube C/C++ Plugin | 4.10 | S3519 |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-C | Name: Sanitize data passed to complex subsystems  If string data is not sanitized before it is passed to other systems, they can contain special characters to trigger certain actions or commands. This can introduce software vulnerabilities. |

| **Noncompliant Code** |
| --- |
| This noncompliant code does not sanitize the data before it is passed to the *system()* function. |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr); system(buffer); |

| **Compliant Code** |
| --- |
| This compliant code uses a pre-defined list of acceptable user input and checks against them. It removes any characters that are not allowed. |
| 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 = '\_'; } |
| Resources: <https://wiki.sei.cmu.edu/confluence/display/c/STR02-C.+Sanitize+data+passed+to+complex+subsystems> |

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

| **Principles(s): Sanitize Data Sent to Other Systems. Sanitizing the data, especially for an SQL query, can help prevent SQL injection.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 |  | Supported by stubbing/taint analysis |
| CodeSonar | 9.0p0 | 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 | 6.5 | TAINTED\_STRING | Fully Implemented |
| Klocwork | 2024.4 | NNTS.TAINTED  SV.TAINTED.INJECTION |  |
| LDRA tool suite | 9.7.1 | 108 D, 109 D | Partially Implemented |
| Parasoft C/C++ test | 2024.2 | 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 | R2024b | CERT C: Rec. STR02-C | Checks for:   * Execution of externally controlled command * Command executed from externally controlled path * Library loaded from externally controlled path   Rec. Partially covered |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory Protection | STD-005-CPP | Name: Detect and handle memory allocation errors  Memory allocations need to be checked just in case they fail. If they do, and it is not checked, it could cause abnormalities in the program such as early termination. |

| **Noncompliant Code** |
| --- |
| This code contains an *int* pointer being allocated, and the allocation results are not being checked for errors. |
| #include <cstring>   void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new int[size];  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy; } |

| **Compliant Code** |
| --- |
| In this compliant code, the allocated *int* pointer is being checked for errors, just in case the allocation failed. |
| #include <cstring> #include <new>   void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new (std::nothrow) int[size];  if (!copy) {  // Handle error  return;  }  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy; } |
| Resources: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors> |

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

| **Principles(s):** Architect and Design for Security Policies. The reason why I chose this one is because handling memory allocation errors, no matter if it was successful or not, is a good practice and should always be done by default. Doing so helps implement good coding practices and security. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE |  |  |  |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| Helix QAC | 2025.1 | C++3225, C++3226, C++3227, C++3228, C++3229, C++4632 |  |
| Klocwork | 2025.1 | NPD.CHECK.CALL.MIGHT NPD.CHECK.CALL.MUST NPD.CHECK.MIGHT NPD.CHECK.MUST NPD.CONST.CALL NPD.CONST.DEREF NPD.FUNC.CALL.MIGHT NPD.FUNC.CALL.MUST NPD.FUNC.MIGHT NPD.FUNC.MUST NPD.GEN.CALL.MIGHT NPD.GEN.CALL.MUST NPD.GEN.MIGHT NPD.GEN.MUST RNPD.CALL RNPD.DEREF |  |
| LDRA tool suite | 9.7.1 | 45 D | Partially implemented |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function’s parameters is undefined |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2024b | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |
| PVS-Studio | 7.37 | V522, V668 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-C | Name: Use a static assertion to test the value of a constant expression  Assertions are a diagnostic tool for C/C++ that helps find and eliminate software bugs that may result in exposed vulnerabilities. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, it uses the *assert()* macro to assert a property regarding a mapped structure that helps the code behave properly. However, the *assert* macro should be placed inside the function and executed. |
| #include <assert.h>   struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT; };   int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)); } |

| **Compliant Code** |
| --- |
| For this compliant code, a preprocessor conditional statement is used. The *#error* type provides clear diagnostic messages. |
| 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 |
| Resources: <https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression> |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhuas Suite | 7.2.0 | CertC-DCL03 |  |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 9.0p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| Compass/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 | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Name: Handle all exceptions  All exceptions thrown by a program must be caught and handled properly. Doing so will make sure the stack is properly unwound, and any external resources are managed before the program is terminated. |

| **Noncompliant Code** |
| --- |
| In the code below, both functions do not catch any exceptions thrown by *throwing\_func().* |
| void throwing\_func() noexcept(false);   void f() {  throwing\_func(); }   int main() {  f(); } |

| **Compliant Code** |
| --- |
| In this compliant code, the *main()* function is properly handling any exceptions thrown by *throwing\_func()*. Doing so makes sure the stack is properly unwound and any external resources are managed. |
| void throwing\_func() noexcept(false);   void f() {  throwing\_func(); }   int main() {  try {  f();  } catch (...) {  // Handle error  } } |
| Resources: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions> |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 | [Insert text.] |
| CodeSonar | 9.0p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Helix QAC | 2025.1 | C++4035, C++4036, C++4037 |  |
| Klocwork | 2025.1 | MiSRA.CATCH.ALL |  |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exception  Each exception explicitly thrown in the code shall have a handler of a compatible type in call paths that could lead to that point |
| Polyspace Bug Finder | R2024b | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| RuleChecker | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | STD-008-CPP | Name: Write constructor member initializers in the canonical order  The order in which members are initialized matters. They are determined by the declaration order of the variables of the class members or the base class specifier list. Not writing member initializers in canonical order can result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, *dependsOnSomVal* is initialized before *someVal*, which will result in an unspecified value being stored in *dependsOnSomeVal.* |
| class C {  int dependsOnSomeVal;  int someVal;   public:  C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {} }; |

| **Compliant Code** |
| --- |
| This compliant code changed the declaration order of the class member variables, which will make sure the variable dependencies are ordered properly. |
| class C {  int someVal;  int dependsOnSomeVal;   public:  C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {} }; |
| Resources: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP53-CPP.+Write+constructor+member+initializers+in+the+canonical+order> |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Initializer-list-order | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP53 |  |
| Clang | 3.9 | -Wreorder |  |
| CodeSonar | 9.0p0 | LANG.STRUCT.INIT.OOMI | Out of Order Member Initializers |
| Helix QAC 2025.1 | C++4053 |  |  |
| Klocwork | 2025.1 | CERT.OOP.CTOR.INIT\_ORDER |  |
| LDRA tool suite | 9.7.1 | 206 S | Fully implemented |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-OOP53-1 | List members in an initialization list in the order in which they are declared |
| Polyspace Bug Finder | R2024b | CERT C++: OOP53-CPP | Checks for members not initalized in canonical order (rule fully covered) |
| RuleChecker | 22.10 | Initializer-list-order | Fully checked |
| SonarQube C/C++ Plugin | 4.10 | S3229 |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Concurrency | STD-009-CPP | Name: Do not destroy a mutex while it is locked.  Mutex objects are used to protect shared data from being accessed concurrently. If a mutex object is destroyed while a blocked thread is waiting for the lock, then shared data and critical sections are not protected. |

| **Noncompliant Code** |
| --- |
| The code below creates several threads that each invoke the *do\_work* function. What makes it noncompliant is that it contains a race condition, which allows the mutex to be destroyed while it is still owned. |
| #include <mutex> #include <thread>  const size\_t maxThreads = 10;  void do\_work(size\_t i, std::mutex \*pm) {  std::lock\_guard<std::mutex> lk(\*pm);   // Access data protected by the lock. }  void start\_threads() {  std::thread threads[maxThreads];  std::mutex m;   for (size\_t i = 0; i < maxThreads; ++i) {  threads[i] = std::thread(do\_work, i, &m);  } } |

| **Compliant Code** |
| --- |
| This compliant code extends the lifetime of the mutex, eliminating the race condition. |
| #include <mutex> #include <thread>  const size\_t maxThreads = 10;  void do\_work(size\_t i, std::mutex \*pm) {  std::lock\_guard<std::mutex> lk(\*pm);   // Access data protected by the lock. }  std::mutex m;  void start\_threads() {  std::thread threads[maxThreads];   for (size\_t i = 0; i < maxThreads; ++i) {  threads[i] = std::thread(do\_work, i, &m);  } } |
| Resources: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/CON50-CPP.+Do+not+destroy+a+mutex+while+it+is+locked> |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.0p0 | CONCURRENCY.LOCALARG | Local Variable Passed to Thread |
| Helix QAC | 2025.1 | DF961, DF4962 |  |
| Klocwork | 2025.1 | CERT.CONC, MUTEX.DESTROY\_WHILE\_LOCKED |  |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-CON50-a | Do not destory another thread’s mutex |
| Polyspace Bug Finder | R2024b | CERT C++: CON50-CPP | Checks for destruction of locked mutex (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers | STD-010-CPP | Name: Do not cast to an out-of-range enumeration value  Arithmitic values must be within the range of values the enumeration can represent, doing so avoids operating on unspecified values. |

| **Noncompliant Code** |
| --- |
| This noncompliant code checks to see if a value is within the range of enumeration values, but it is doing so after casting the enumeration type. |
| enum EnumType {  First,  Second,  Third };  void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);   if (enumVar < First || enumVar > Third) {  // Handle error  } } |

| **Compliant Code** |
| --- |
| This compliant code uses a scoped enumeration, which has a fixed underlying *int* data type by default. This allows any value in the parameter to be converted to an acceptable enumeration value. |
| enum EnumType {  First,  Second,  Third };  void f(int intVar) {  if (intVar < First || intVar > Third) {  // Handle error  }  EnumType enumVar = static\_cast<EnumType>(intVar); } |
| Resources: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/INT50-CPP.+Do+not+cast+to+an+out-of-range+enumeration+value> |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Cast-integer-to-enum | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| CodeSonar | 9.0p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Ceorcion Alters Value  Cast Alters Value |
| Helix QAC | 2025.1 | C++3013 |  |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| PVS-Studio | 7.37 | V1026 |  |
| RuleChecker | 22.10 | Cast-integer-to-enum | Partially checked |
| Polyspace Bug Finder | R2024b | CERT C++: INT50-CPP | Checks for casting to out-of-range enumeration value (rule fully covered) |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

Automation in DevSecOps is an important process that allows developers to test their code as it gets produced to more quickly ship updates to production. A good rule that many organizations follow is to test early and often, which is what automation tools help them achieve. In the DevSecOps diagram above, automation would best fit on the Pre-production side with the *Design* and *Build* processes. The reason for this is because, as I mentioned earlier, automation testing should be completed as the code is being written, making sure that any bugs and errors are caught early and often.

### 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 (High) | 2 |
| STD-002-CPP | High | Probable | Medoum | P12 (High) | 1 |
| STD-003-CPP | High | Likely | Medium | P18 (High) | 1 |
| STD-004-C | High | Likely | Medium | P9 (High) | 1 |
| STD-005-CPP | High | Likely | Medium | P18 (High) | 1 |
| STD-006-C | Low | Unlikely | Medium | P3 (Low) | 4 |
| STD-007-CPP | Low | Probable | Medium | P4 (Low) | 4 |
| STD-008-CPP | Medium | Unlikely | Medium | P4 (Low) | 3 |
| STD-009-CPP | Medium | Probable | High | P4 (Low) | 3 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 (Low) | 3 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest is properly securing data while it is stored on a physical device, such as a hard drive or solid-state drive. This includes encrypting the data with an encryption algorithm such as the AES-256. This ensures that even when the device is stolen, the data is still protected. |
| Encryption in flight | Encryption in flight is the process of encrypting data while it is being transferred between two different systems or locations. This way, if the data gets intercepted while in transit, it will not be able to be viewed or decrypted. This policy is crucial because there is a ton of data being sent over the internet and making sure that it’s secure is important to keeping the integrity and confidentiality of the data and organization or individual. |
| Encryption in use | Encryption in use in the process of making sure that data is still encrypted while it’s being used in an application. This way, if the data is accessed by a malicious user, then it still cannot be viewed. |

| 1. Triple-A Framework\* | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of correctly identifying a user and making sure that they are who they say to be. Each user has their own unique identification credentials, most likely a numeric user ID. This way, the server can use these valid credentials to check against whenever a user tries to log in to their respective account. This policy applies because it prevents malicious users from being able to access other users' personal information. |
| Authorization | In the Triple-A Framework, authorization is the process of giving a user the right number of privileges to be able to complete their tasks. They are given a certain level of authorization that is respective to their role. For example, in a web application a normal user should not be able to perform administrator duties on other user accounts; only administrators should be able to do that. This policy applies because it helps keep the integrity of the system high by protecting certain internal processes that may try to be taken advantage of by a malicious user. |
| Accounting | In the Triple-A Framework, accounting is the process of keeping track of certain types of data and statistics such as logging information, database changes, user session duration, privilege setting changes, user file access, creation of new users, user logins, etc. This policy applies and is important because if anything were to go wrong, such as a malicious user giving themselves unauthorized administrator priviledges, then it can be traced back to exactly when and where it happened. |

\*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 |  |
| 1.2 | 06/11/2025 | Defined security policies and coding standards | Braden Whitcher |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

Resources

*Default encryption at rest.* (n.d.). Google Cloud.

<https://cloud.google.com/docs/security/encryption/default-encryption#:~:text=Encryption%20at%20rest%20is%20encryption,)%20algorithm%2C%20AES%2D256>.