**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 | Whenever a program takes in user input data ensure that it fits within the guidelines of security. Create code that protects against buffer overflows, SQL injection queries, etc. |
| 1. Heed Compiler Warnings | Analyze compiler warnings to mitigate potential vulnerabilities. They help prevent mistakes and lead to more reliable code. Compilers help debug the code to reduce potential code clogs like unused variables, or faulty code. |
| 1. Architect and Design for Security Policies | When designing a project, architect policies to adhere to security principles. Building security into the design of the project proactively identifies potential threats. |
| 1. Keep It Simple | The best way to design code is to keep it simple. Improving readability allows others to help improve code and find errors. Simple code helps developers maintain and update the code. |
| 1. Default Deny | To maintain a security tight program, only allow specific actions and auto default to deny. Reducing access helps reduce the areas of potential vulnerabilities. |
| 1. Adhere to the Principle of Least Privilege | Whitelist user actions manually. Only allow users access to the most basic actions needed for them to perform. This is also applied to systems and components. Reduce points of contact based on necessity. |
| 1. Sanitize Data Sent to Other Systems | Remove potential harm from data before sending it to other systems. Remove or encode sensitive data to limit potential attack points. Having sensitive data in only one area allows the developer to prioritize security to that area. |
| 1. Practice Defense in Depth | When creating security for a program identify points of vulnerability and layer defense. Adding multiple different protections helps protect against different threats. Techniques such as firewalls, antivirus software, etc. address different vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | Always test code multiple times in multiple different ways and do so consistently throughout the project. Testing often and in various ways helps identify threats or potential vulnerabilities. Employ the use of quality testing teams to gain different perspectives. |
| 1. Adopt a Secure Coding Standard | Choosing and following a coding standard ensures consistency. Consistent code is easier to work with and enables future developers to expand upon. Coding standards also have practices that reduce known vulnerabilities. |

### 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] | Ensure accurate use of data typing to configure project. This ensures computers can read and handle information accordingly. It also helps reinforce consistency and readability. |

| **Noncompliant Code** |
| --- |
| Data type does not match what the variable is for |
| string numberOfApples;  int gravity; |

| **Compliant Code** |
| --- |
| Clearly defined type that matches values that other developers would understand |
| int numberOfApples;  double gravity; |

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

| **Principles(s):** Ensure correct data type declaration. Use rationality to determine the data type that will hold the information needed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2024a | CERT C: Rule INT31-C | Checks for integer overflow, sign change integer conversion overflow, etc |
| Polyspace Bug Finder | R2024a | CERT C++: CTR53-CPP | Checks for invalid iterator range (rule partially covered). |
| Astree | 22.1 | Cast-integer-to-enum | Partially checks |
| CodeSonar | 8.1p0 | ALLOC.UAF  STR52-CPP | Use after free |

Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Choosing the appropriate data value for the data type and the programs purpose ensures reusability and reduces errors |

| **Noncompliant Code** |
| --- |
| Unclear and inconsistent naming devices used |
| int A;  double g; |

| **Compliant Code** |
| --- |
| Complete name and declaration to allow others to clearly understand intentions |
| int numberOfApples = 0;  double gravity = 9.8; |

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

| **Principles(s):** When declaring variables ensure they are properly initialized. This helps set intentions. incorrect initializations are hard to catch |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | high | L3 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  ALLOC.SIZE.TRUNC  IO.TAINT.SIZE  MISC.MEM.SIZE.BAD  LANG.MEM.BO  LANG.MEM.BU  LANG.STRUCT.PARITH  LANG.STRUCT.PBB  LANG.STRUCT.PPE  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Addition overflow of allocation size Addition overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Truncation of allocation size Tainted allocation size Unreasonable size argument Buffer Overrun Buffer Underrun Pointer Arithmetic Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun |
| Polyspace Bug Finder | R2024a | CERT C: Rule MEM31-C | Checks for memory leak (rule fully covered) |
| Axivion Bauhaus Suite | CertC-MEM31 | CertC-MEM31 | Can detect dynamically allocated resources that are not freed |
| Polyspace Bug Finder | R2024a | CERT C++: MEM53-CPP | Checks for objects allocated but not initialized (rule fully covered). |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003  -CPP] | When using strings in a program it is vital to ensure correctness to prevent vulnerabilities and exploitation such as null-termination errors or truncations. |

| **Noncompliant Code** |
| --- |
| Reading from user input using gets() function. Gets() does not limit what can be entered and can therefore be targeted by buffer overflows |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  char userInput[5];  puts(“enter PIN”);  gets(userInput); |

| **Compliant Code** |
| --- |
| [Compliant description] |
| [Compliant code block; code should be indented using 12-point Courier New font.]  char userInput[5];  puts(“enter PIN”);  getline(std::cin, userInput, 5) |

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

| **Principles(s):** When receiving a string input from a user ensure that it is limited, to avoid buffer overflow. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| medium | Likely | high | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  ALLOC.SIZE.TRUNC  IO.TAINT.SIZE  MISC.MEM.SIZE.BAD  LANG.MEM.BO  LANG.MEM.BU  LANG.STRUCT.PARITH  LANG.STRUCT.PBB  LANG.STRUCT.PPE  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Addition overflow of allocation size Addition overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Truncation of allocation size Tainted allocation size Unreasonable size argument Buffer Overrun Buffer Underrun Pointer Arithmetic Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun |
| Polyspace Bug Finder | R2024a | CERT C: Rule FIO30-C | Checks for tainted string format (rule partially covered) |
| Parasoft C/C++test | [2023.1 | CERT\_C-FIO37-a | Avoid accessing arrays out of bounds |
| Polyspace Bug Finde | R2024a | CERT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004  -CPP] | Be aware of potential SQL injection vulnerabilities and seek to prevent them |

| **Noncompliant Code** |
| --- |
| Using user input raw data without validating |
| Query = “SELECT \* FROM animals WHERE name = ‘” + name + “’”; |

| **Compliant Code** |
| --- |
| Check to ensure the inputted data is valid before inserting it into an SQL query |
| Query = “SELECT \* FROM animals WHERE name = ?”;  //check to see if name is valid and within bounds  Query.setString(1, name); |

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

| **Principles(s):** Validate user input before inserting it into an SQL Query. Instead of letting the user directly input a variable into the query have another function set the variable. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  ALLOC.SIZE.TRUNC  IO.TAINT.SIZE  MISC.MEM.SIZE.BAD  LANG.MEM.BO  LANG.MEM.BU  LANG.STRUCT.PARITH  LANG.STRUCT.PBB  LANG.STRUCT.PPE  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Addition overflow of allocation size Addition overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Truncation of allocation size Tainted allocation size Unreasonable size argument Buffer Overrun Buffer Underrun Pointer Arithmetic Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun |
| Polyspace Bug Finder | R2024a | CERT C: Rule FIO30-C | Checks for tainted string format (rule partially covered) |
| Parasoft C/C++test | [2023.1 | CERT\_C-FIO37-a | Avoid accessing arrays out of bounds |
| Polyspace Bug Finde | R2024a | CERT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |
| Parasoft C/C++test | [2023.1 | CERT\_C-MSC41-a | Do not hard code string literals |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Check to ensure code is not accessing memory it should not |

| **Noncompliant Code** |
| --- |
| When a pointer is used after the space has been deallocated by using free() |
| Example from SEI CERT C++ Coding[[1]](#footnote-1)  struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  for (struct node \*p = head; p != NULL; p = p->next) {  free(p);  }  } |

| **Compliant Code** |
| --- |
| In this example the value of P is stored in a new variable in order to preserve its contents before freeing it |
| Example from SEI CERT C++ Coding[[2]](#footnote-2)  Struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  struct node \*q;  for (struct node \*p = head; p != NULL; p = q) {  q = p->next;  free(p);  }  } |

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

| **Principles(s):** Ensure proper allocation and deallocation of memory. This protects against attackers by limiting available information. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.1 | Dangling\_pointer\_use | Directly checks that a dangling pointer is not used |
| Polyspace Bug Finder | [R2024a | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |
| Polyspace Bug Finder | R2024a | CERT C++: MEM51-CPP | Checks for:  Invalid deletion of pointer  Invalid free of pointer  Deallocation of previously deallocated pointer  Rule partially covered. |
| CodeSonar | 8.1p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  ALLOC.SIZE.TRUNC  IO.TAINT.SIZE  MISC.MEM.SIZE.BAD  LANG.MEM.BO  LANG.MEM.BU  LANG.STRUCT.PARITH  LANG.STRUCT.PBB  LANG.STRUCT.PPE  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Addition overflow of allocation size Addition overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Truncation of allocation size Tainted allocation size Unreasonable size argument Buffer Overrun Buffer Underrun Pointer Arithmetic Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Assertions are used to make assumptions when debugging code but should not be used for runtime checking. |

| **Noncompliant Code** |
| --- |
| Using assert() for runtime checks may cause a denial of service attack[[3]](#footnote-3) |
| char input;  cout<< “please enter ‘y’ or ‘n’”;  cin>> input;  assert(name == y || input == n); |

| **Compliant Code** |
| --- |
| Assert() should not be used to validate input |
| char input;  cout<< “please enter ‘y’ or ‘n’”;  cin>> input;  if(input ==y){  //do something}  If(input == n){  //do something} |

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

| **Principles(s):** Assertions can be used to catch developer errors that arise from assumptions. They are useful for identifying or stopping problems but cannot be used for runtime error checking. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ECLAIR | [1.2 | **CC2.DCL03** | Fully implemented |
| CodeSonar | 8.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| [Coverity | 2017.07 | ASSERT\_SIDE\_EFFECT | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| [Parasoft Jtest | [2024.1 | CERT.MSC60.ASSERT | Do not use assertions in production code |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-STD] | To prevent the program from terminating from an exception, error handling can be used. |

| **Noncompliant Code** |
| --- |
| When an error is thrown that is not handled the nearest handler, the default handler, will call the program to terminate. |
| Example from SEI CERT C++ Coding[[4]](#footnote-4)    void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| Using a try catch block handles exceptions thrown without using the default handle |
| Example from SEI CERT C++ Coding[[5]](#footnote-5)  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):** Catching all known exceptions helps prevent abrupt terminations of programs that could otherwise lead to potential data vulnerabilities. The automatic handle of exceptions is to abort the program which leaves open streams of information and resources. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-ERR50-a** **CERT\_CPP-ERR50-b** **CERT\_CPP-ERR50-c** **CERT\_CPP-ERR50-d** **CERT\_CPP-ERR50-e** **CERT\_CPP-ERR50-f** **CERT\_CPP-ERR50-g** **CERT\_CPP-ERR50-h** **CERT\_CPP-ERR50-i** **CERT\_CPP-ERR50-j** **CERT\_CPP-ERR50-k** **CERT\_CPP-ERR50-l** **CERT\_CPP-ERR50-m CERT\_CPP-ERR50-n** | The execution of a function registered with 'std::atexit()' or 'std::at\_quick\_exit()' should not exit via an exception Never allow an exception to be thrown from a destructor, deallocation, and swap Do not throw from within destructor There should be at least one exception handler to catch all otherwise unhandled exceptions An empty throw (throw;) shall only be used in the compound-statement of a catch handler Exceptions shall be raised only after start-up and before termination of the program Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) Function called in global or namespace scope shall not throw unhandled exceptions Always catch exceptions Properly define exit handlers The 'abort()' function from the 'stdlib.h' or 'cstdlib' library shall not be used Avoid throwing exceptions from functions that are declared not to throw The 'quick\_exit()' and '\_Exit()' functions from the 'stdlib.h' or 'cstdlib' library shall not be used |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: ERR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr50cpp.html) | Checks for implicit call to terminate() function (rule partially covered) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: ERR58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr58cpp.html) | Checks for exceptions raised during program startup (rule fully covered) |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Buffer Overflows | [STD-008-CPP] | Prevent buffer overflows by validating the input or truncating input |

| **Noncompliant Code** |
| --- |
| Not limiting user input may cause buffer overflows, overwriting another variable |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  String account = “username”;  Char user\_input[5];  Cin>> user\_input; |

| **Compliant Code** |
| --- |
| Limit the user’s input by checking that they entered within bounds. |
| String account = “username”;  Char user\_input[5];  Cin >> setw(5) >> user\_input;  //a check can then be added to check if last value of the array is 0 or some user input that should not be there |

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

| **Principles(s):** Validating user input before using it will help prevent buffer overflows and unauthorized actions. |
| --- |

**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/c/CodeSonar) | 8.1p0 | **IO.INJ.FMT MISC.FMT** | Format string injection Format string |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **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 |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **conversion\_overflow**  **essential-type-assign** | Soundly supported |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **486 S** **589 S** | Fully implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| References to containers | [STD-009-ccp] | If a pointer or reference is stored in a container it is at risk of being invalidated. |

| **Noncompliant Code** |
| --- |
| The insert function can cause invalid iterators if iterator is not properly defined |
| Example from SEI CERT C++ Coding[[6]](#footnote-6)  void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  d.insert(pos, items[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| Properly defining iterators reduces likelihood of vulnerabilities |
| Example from SEI CERT C++ Coding[[7]](#footnote-7)  void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  pos = d.insert(pos, items[i] + 41.0);  }  } |

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

| **Principles(s):** When referencing an item in a container ensure the use of proper iterator. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-CTR51-a** | Do not modify container while iterating over it |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C++: CTR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcctr51cpp.html) | Checks for use of invalid iterator (rule partially covered). |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **ALLOC.UAF** | Use After Free |
| Astree | 22.10 | Overflow\_upon\_dereference | Checks that an error occurs if accessing a dereferenced pointer |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| precedence | [STD-010-CPP] | Know the precedence of operations to ensure accurate results |

| **Noncompliant Code** |
| --- |
| If the user intends to first combine two values and then use a comparison operator, the lack of parentheses causes the wrong result |
| [Noncompliant code block; code should be indented using 12-point Courier New font.]  X & Y == 1  //it will evaluate as follows: X & (Y == 1) |

| **Compliant Code** |
| --- |
| Use parentheses to indicate correct order. The problem stems from the low precedence levels of operators &, |, ^, <<, and >> |
| (X & Y) == 1 |

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

| **Principles(s):** In expressions ensure correct use of precedence and order. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.EXP30** | Fully implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.STRUCT.SE.DEC** **LANG.STRUCT.SE.INC** | Side Effects in Expression with Decrement Side Effects in Expression with Increment |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-EXP50-a** **CERT\_CPP-EXP50-b** **CERT\_CPP-EXP50-c** **CERT\_CPP-EXP50-d** **CERT\_CPP-EXP50-e** **CERT\_CPP-EXP50-f** | The value of an expression shall be the same under any order of evaluation that the standard permits Don't write code that depends on the order of evaluation of function arguments Don't write code that depends on the order of evaluation of function designator and function arguments Don't write code that depends on the order of evaluation of expression that involves a function call Between sequence points an object shall have its stored value modified at most once by the evaluation of an expression Don't write code that depends on the order of evaluation of function calls |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: EXP50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp50cpp.html) | Checks for situations where expression value depends on order of evaluation (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.)*

Tool automation must be added to address vulnerabilities in the DevOps System. These tools act as a sort of testing to verify that the code does not violate any coding standards. Therefore, the use of these tools should be implemented in the early stages of testing. Before addressing all the vulnerabilities of the system, these tools could help illustrate easily fixed problems. They should also be used while building to ensure that the coding standards are followed. Referring to the model above, tool automation should be placed between the ‘build’ and ‘verify and test’ stages.

### 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 | Unlikely | Medium | High | 2 |
| STD-002-CPP | High | Unlikely | high | L3 | 3 |
| STD-003-CPP | medium | Likely | high | P6 | L2 |
| STD-004-CPP | Medium | Likely | High | P9 | L2 |
| 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 | High | Likely | Medium | P18 | L1 |
| STD-009-CPP | High | Probable | High | P | L2 |
| STD-010-CPP | Medium | Probable | Medium | P8 | L2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | This is the policy of encrypting the data where it is stored. All the information in a database should be encrypted to reduce the likelihood of someone accessing all the information readily. This also prevents people such as employees from accessing sensitive user data such as passwords. |
| Encryption in flight | Encryption in flight is the policy of keeping the data encrypted as it moves. When data moves from the back end to the front end it stays encrypted. Attackers then won’t get useful information from this weak point. Another example is when information travels from a user to a server, such as messaging. |
| Encryption in use | Encryption in use is the policy of leaving the data encrypted when using it in the program. Any actions performed on data should always be encrypted. Functions and methods should never directly use the data. Keeping it encrypted through all processes limits it’s vulnerability. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the way the application, or program, identifies a user. Authentication is the check to determine that a user is accessing their own data. It is the first step in protecting an individual’s data. |
| Authorization | Authorization as a security policy is the concept of only allowing users to access information and functions that they require to perform their unique tasks. For example, an end user should never be able to access the admin console. |
| Accounting | Accounting is a security policy that records access. It is a way for the company to monitor who is accessing data and all relevant information such as when, whom, and what. This may help track down malicious users. |

**\***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 | 11/17/2024 | Update Template | Shayla Vincent | [Insert text.] |
| 1.2 | 12/14/2024 | Finalize Template | Shayla Vincent | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

1. https://wiki.sei.cmu.edu/confluence/display/c/MEM30-C.+Do+not+access+freed+memory [↑](#footnote-ref-1)
2. https://wiki.sei.cmu.edu/confluence/display/c/MEM30-C.+Do+not+access+freed+memory [↑](#footnote-ref-2)
3. https://wiki.sei.cmu.edu/confluence/display/c/MSC11-C.+Incorporate+diagnostic+tests+using+assertions [↑](#footnote-ref-3)
4. https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions [↑](#footnote-ref-4)
5. https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions [↑](#footnote-ref-5)
6. https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR51-CPP.+Use+valid+references%2C+pointers%2C+and+iterators+to+reference+elements+of+a+container [↑](#footnote-ref-6)
7. https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR51-CPP.+Use+valid+references%2C+pointers%2C+and+iterators+to+reference+elements+of+a+container [↑](#footnote-ref-7)