**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 | Data input from the user must be validated to ensure that it is within the bounds of expected input. Input that escapes the bounds can overflow buffers and lead to vulnerabilities. |
| 1. Heed Compiler Warnings | Compiler warnings are useful for letting a developer know that their code might do something unexpected. Warnings usually appear when the developer doesn’t initialize a variable, return a value from a method, argument mismatching, etc. |
| 1. Architect and Design for Security Policies | Security policies that are established should be followed. Policies are put in place to defend against malicious users looking to exploit vulnerable code. Code developed using the security policies from this document will be better protected against exploits. |
| 1. Keep It Simple | Ensuring code written meets only the requirements and nothing more is important. Additional features added to code means more chances of developing vulnerable code, and it might not meet user needs. |
| 1. Default Deny | Never trust the user by default and assume that the user might act maliciously. If a user that shouldn’t be trusted is trusted, they could potentially access important company/user data that they shouldn’t have access to. |
| 1. Adhere to the Principle of Least Privilege | It is important to give users access to only what they need and nothing more. If a user is granted access or privileges that they shouldn’t have, then they might be given access to data they shouldn’t have access to; can unleash an attack; exploit a vulnerability; etc. |
| 1. Sanitize Data Sent to Other Systems | For an entire system to run smoothly, data sent to other parts should be checked/sanitized for validity. If something unexpected happened or in other extreme situations, data could be sent to another system that exceeds the bounds of the data expected to be sent. |
| 1. Practice Defense in Depth | The concept behind defense in depth is that one security measure might fail or be exploited, so it is important to have another layer of security. Having multiple layers of defense when passing data through a system is critical. |
| 1. Use Effective Quality Assurance Techniques | Confirming that code is well-written, documented, and error-free is a vital start to quality code. Testing code and the entire system can help ensure that the system is secure and less vulnerable. |
| 1. Adopt a Secure Coding Standard | Following a secure coding standard will guarantee that the written code adheres to laid out security practices, consistent coding practices, naming conventions, etc. This will guarantee that code written by one developer will line up with other developers’ code and prevent complications and confusion later. |

### 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** | EXP39-C | Do not access a variable through a pointer of an incompatible type |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, an object of type float is incremented through an int \*. The programmer can use the unit in the last place to get the next representable value for a floating-point type. However, accessing an object through a pointer of an incompatible type is undefined behavior.   * *SEI CERT C Coding Standard* |
| #include <stdio.h>  void f(void) {  if (sizeof(int) == sizeof(float)) {  float f = 0.0f;  int \*ip = (int \*)&f;  (\*ip)++;  printf("float is %f\n", f);  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the standard C function nextafterf() is used to round toward the highest representable floating-point value.   * *SEI CERT C Coding Standard* |
| #include <float.h>  #include <math.h>  #include <stdio.h>  void f(void) {  float f = 0.0f;  f = nextafterf(f, FLT\_MAX);  printf("float is %f\n", f);  } |

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

| **Principles(s):** This standard aligns with Principle 3, Architect and Design for Security Policies, because the standard provides additional program security as recommended by the *SEI CERT C Coding Standard*. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **94 S, 554 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | **CERT\_C-EXP39-a**  **CERT\_C-EXP39-b**  **CERT\_C-EXP39-c**  **CERT\_C-EXP39-d**  **CERT\_C-EXP39-e**  **CERT\_C-EXP39-f** | There shall be no implicit conversions from integral to floating type A cast should not be performed between a pointer to object type and a different pointer to object type Avoid accessing arrays and pointers out of bounds Avoid buffer overflow from tainted data due to defining incorrect format limits Avoid buffer read overflow from tainted data Avoid buffer write overflow from tainted data |

* <https://wiki.sei.cmu.edu/confluence/display/c/EXP39-C.+Do+not+access+a+variable+through+a+pointer+of+an+incompatible+type>

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT50-CPP | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. On a two’s complement system, the valid range of values that can be represented by EnumType are [0..3], so if a value outside of that range were passed to f(), the cast to EnumType would result in an unspecified value, and using that value within the if statement results in unspecified behavior.   * *SEI CERT C++ Coding Standard* |
| 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 solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does this by restricting the converted value to one for which there is a specific enumerator value.   * *SEI CERT C++ Coding Standard* |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  if (intVar < First || intVar > Third) {  // Handle error  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** This standard aligns with Principle 3, Architect and Design for Security Policies, because the standard provides additional program security as recommended by the *SEI CERT C Coding Standard*. Also, it adheres to Principles 1 and 7 because data/input needs to be sanitized before using it. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **cast-integer-to-enum** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **LANG.CAST.COERCE**  **LANG.CAST.VALUE** | Coercion Alters Value  Cast Alters Value |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-INT50-a** | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |

#### <https://wiki.sei.cmu.edu/confluence/display/cplusplus/INT50-CPP.+Do+not+cast+to+an+out-of-range+enumeration+value>

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR50-CPP | Guarantee that storage for strings has sufficient space for character data and the null terminator |

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

| **Compliant Code** |
| --- |
| The best solution for ensuring that data is not truncated and for guarding against buffer overflows is to use std::string instead of a bounded array, as in this compliant solution.   * *SEI CERT C++ Coding Standard* |
| #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):** This standard aligns with Principle 1, Validate Input Data, because it ensures that data will be within the required range. This standard also aligns with Principle 4, Keep it Simple, because it is a simple solution to a big problem. |
| --- |

**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) | 22.10 | **stream-input-char-array** | Partially checked + soundly supported |
| [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 |
| [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' |

* <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>

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | CWE-89 | Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') |

| **Noncompliant Code** |
| --- |
| An SQL injection uses the fact that 1 = 1 to return all elements in a SQL database. A malicious user can input an arbitrary value with a string such as “OR 1=1” or “OR “”=””” concatenated to the end to get access to all data of the column in the SQL database. |
| uName = getRequestString("username");  uPass = getRequestString("userpassword");  sql = 'SELECT \* FROM Users WHERE Name ="' + uName + '" AND Pass ="' + uPass + '"'   * <https://www.w3schools.com/sql/sql_injection.asp> |

| **Compliant Code** |
| --- |
| SQL parameters are used in many languages to protect against SQL injection. Parameters check the literal expression passed into them rather than treating them as executable SQL. |
| txtUserId = getRequestString("UserId"); txtSQL = "SELECT \* FROM Users WHERE UserId = @0"; db.Execute(txtSQL,txtUserId);   * <https://www.w3schools.com/sql/sql_injection.asp> |

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

| **Principles(s):** This standard aligns with Principle 1, 3, 7, 8, and 10 because it ensures SQL commands are correctly validated, is secure, and it is standard to use parameters to command a SQL database. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MSC24** | Fully implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **BADFUNC.\*  (customization)** | A number of CodeSonar's "Use of \*" checks are for deprecated/obsolescent functions CodeSonar also provides a mechanism for users to create custom checks for uses of specified functions |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **586** | Fully supported |

* <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152260>

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM51-CPP | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| In the following noncompliant code example, an array is allocated with array new[] but is deallocated with a scalar delete call instead of an array delete[] call, resulting in undefined behavior.   * *SEI CERT C++ Coding Standard* |
| void f() {  int \*array = new int[10];  // ...  delete array;  } |

| **Compliant Code** |
| --- |
| In the compliant solution, the code is fixed by replacing the call to delete with a call to delete [] to adhere to the correct pairing of memory allocation and deallocation functions.   * *SEI CERT C++ Coding Standard* |
| void f() {  int \*array = new int[10];  // ...  delete[] array;  } |

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

| **Principles(s):** This standard aligns with Principle 2, Heed Compiler Warnings, because an IDE such as Visual Studios 2022 will warn of improperly deallocated resources such as deleting an array with a scalar delete. It also aligns with Principles 3 and 4 because it is a simple solution to a severe and likely threat and is compliant to the *SEI CERT C++ Coding Standard*. |
| --- |

**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 | **ALLOC.FNH ALLOC.DF ALLOC.TM ALLOC.LEAK** | Free non-heap variable Double free Type mismatch Leak |
| [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 |

* <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM51-CPP.+Properly+deallocate+dynamically+allocated+resources>

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | PRE31-C | Avoid side effects in arguments to unsafe macros |

| **Noncompliant Code** |
| --- |
| This noncompliant code example includes an assert() macro containing an expression (index++) that has a side effect.   * *CEI CERT C Coding Standard*   The statement index++ inside of the assert statement changes the value of index within the assertion but doesn’t change it in the final release version. |
| #include <assert.h>  #include <stddef.h>  void process(size\_t index) {  assert(index++ > 0); /\* Side effect \*/  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution avoids the possibility of side effects in assertions by moving the expression containing the side effect outside of the assert() macro.   * *CEI CERT C Coding Standard* |
| #include <assert.h>  #include <stddef.h>  void process(size\_t index) {  assert(index > 0); /\* No side effect \*/  ++index;  /\* ... \*/  } |

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

| **Principles(s):** This standard aligns with Principles 4, Keep It Simple; and 9, Use Effective Quality Assurance Techniques. It can be a simple issue to avoid, and it can apply to quality assurance techniques such as assert() statements. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-PRE31** | Fully implemented |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.EXP31 CC2.PRE31** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **9 S, 562 S, 572 S, 35 D, 1 Q** | Fully implemented |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **666, 2666** | Fully supported |

* <https://wiki.sei.cmu.edu/confluence/display/c/PRE31-C.+Avoid+side+effects+in+arguments+to+unsafe+macros>

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR62-CPP | Detect errors when converting a string to a number |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, multiple numeric values are converted from the standard input stream. However, if the text received from the standard input stream cannot be converted into a numeric value that can be represented by an int, the resulting value stored into the variables i and j may be unexpected.   * *SEI CERT C++ Coding Standard* |
| #include <iostream>  void f() {  int i, j;  std::cin >> i >> j;  // ...  } |

| **Compliant Code** |
| --- |
| In this compliant solution, each converted value read from the standard input stream is tested for validity before reading the next value in the sequence, allowing error recovery on a per-value basis. It checks std::istream::fail() to see if the failure bit was set due to a conversion failure or whether the bad bit was set due to a loss of integrity with the stream object. If a failure condition is encountered, it is cleared on the input stream and then characters are read and discarded until a ' ' (space) character occurs. The error handling in this case only works if a space character is what delimits the two numeric values to be converted.   * *SEI CERT C++ Coding Standard* |
| #include <iostream>  #include <limits>  void f() {  int i;  std::cin >> i;  if (std::cin.fail()) {  // Handle failure to convert the value.  std::cin.clear();  std::cin.ignore(  std::numeric\_limits<std::streamsize>::max(), ' ');  }  int j;  std::cin >> j;  if (std::cin.fail()) {  std::cin.clear();  std::cin.ignore(  std::numeric\_limits<std::streamsize>::max(), ' ');  }  //…  } |

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

| **Principles(s):** This standard aligns with Principles 1, 3, and 7 because it is a defense against exploits in the form of validating and sanitizing input before working with it or passing it to another part of the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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 atoll * Use of atoll |

* <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR62-CPP.+Detect+errors+when+converting+a+string+to+a+number>

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declarations | DCL51-CPP | Do not declare or define a reserved identifier |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, because the C++ standard template library header <cinttypes> is specified to include <cstdint>, as per [c.files] paragraph 4 [ISO/IEC 14882-2014], the name MAX\_SIZE conflicts with the name of the <cstdint> header macro used to denote the upper limit of std::size\_t.   * *SEI CERT C++ Coding Standard*   Reserved identifiers are reserved for a reason, and they should not be redefined. |
| #include <cinttypes>  // for int\_fast16\_t  void f(std::int\_fast16\_t val) {  enum { MAX\_SIZE = 80 };  // ...  } |

| **Compliant Code** |
| --- |
| This compliant solution avoids redefining reserved names.   * *SEI CERT C++ Coding Standard* |
| #include <cinttypes>// for std::int\_fast16\_t  void f(std::int\_fast16\_t val) {  enum { BufferSize = 80 };  // ...  } |

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

| **Principles(s):** This standard best aligns with Principles 3, Architect and Design for Security Policies, and 4, Keep It Simple, because it is a simple solution to a problem defined by the *SEI CERT C++ Coding Standard*. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **86 S, 218 S, 219 S, 580 S** | Fully implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **LANG.ID.NU.MK**  **LANG.STRUCT.DECL.RESERVED** | Macro name is C keyword  Declaration of reserved name |

* <https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL51-CPP.+Do+not+declare+or+define+a+reserved+identifier>

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | EXP53-CPP | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an uninitialized local variable is evaluated as part of an expression to print its value, resulting in undefined behavior.   * *CEI CERT C++ Coding Standard*   An object will have an indeterminate value when they are initialized without a value until they are assigned one. Attempting to print i will result in undefined behavior. |
| #include <iostream>  void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the object is initialized prior to printing its value.   * *CEI CERT C++ Coding Standard*   Always initialize an object before working with it. |
| #include <iostream>  void f() {  int i = 0;  std::cout << i;  } |

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

| **Principles(s):** This standard aligns best to Principles 2, Heed Compiler Warnings, and 10, Adopt a Secure Coding Standard. An instance of utilizing uninitialized memory would be caught in a compiler such as Visual Studio 2022. Also, it pertains to writing secure, consistent code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **LANG.STRUCT.RPL LANG.MEM.UVAR** | Return pointer to local Uninitialized variable |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **53 D, 69 D, 631 S, 652 S** | Partially implemented |

* <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory>

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous | MSC51-CPP | Ensure your random number generator is properly seeded |

| **Noncompliant Code** |
| --- |
| This noncompliant code example generates a sequence of 10 pseudorandom numbers using the Mersenne Twister engine. No matter how many times this code is executed, it always produces the same sequence because the default seed is used for the engine.   * *CEI CERT C++ Coding Standard*   The Mersenne Twister (MT) pseudorandom number generator engine is created but uses a default seed that will produce the same numbers over and over. Seeding it with the time is also not good enough because someone can use the engine at a specific time to produce results they are looking for. |
| #include <iostream>  void f() {  std::mt19937 engine;  for (int i = 0; i < 10; ++i) {  std::cout << engine() << ", ";  }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses std::random\_device to generate a random value for seeding the Mersenne Twister engine object. The values generated by std::random\_device are nondeterministic random numbers when possible, relying on random number generation devices, such as /dev/random. When such a device is not available, std::random\_device may employ a random number engine; however, the initial value generated should have sufficient randomness to serve as a seed value.   * *SEI CERT C++ Coding Standard*   First, a random\_device should be created to generate a random value to seed the MT engine with before using it. |
| #include <random>  #include <iostream>  void f() {  std::random\_device dev;  std::mt19937 engine(dev());  for (int i = 0; i < 10; ++i) {  std::cout << engine() << ", ";  }  } |

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

| **Principles(s):** This standard aligns with Principles 9 and 10 because it ensures that code is well-written, error-free, and sticks to predefined methods of generating pseudorandom numbers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **default-construction** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **HARDCODED.SEED** **MISC.CRYPTO.TIMESEED** | Hardcoded Seed in PRNG Predictable Seed in PRNG |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: MSC51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc51cpp.html) | Checks for:   * Deterministic random output from constant seed * Predictable random output from predictable seed   Rule partially covered. |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-MSC51-a** | Properly seed pseudorandom number generators |

* <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MSC51-CPP.+Ensure+your+random+number+generator+is+properly+seeded>

### 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.

There are multiple changes to be made for the transition of DevOps to DevSecOps at Green Pace. During the Assess and plan phase above, it’s important to start thinking about where security vulnerabilities might lie within the system. The Design phase is where best practices and coding standards are referenced to ensure that code produced is more secure. After that, the system is started to be created, keeping the secure coding standards in mind. When there is a finished product, it can be tested for usability and security purposes, for example, with automation tools.

Before releasing the product, it can be fully configured and tested appropriately. Penetration testing is good for ensuring all defense-in-depth code is properly securing the system. Proper logs should be kept for events such as users logging in and especially alerting when a user accesses an unintended feature (for example, accessing database features to view, update, delete data). Unintended feature access should be caught and blocked. After a product is released, it needs to be continuously maintained based on customer needs and ever-evolving known security vulnerabilities.

### 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 |
| --- | --- | --- | --- | --- | --- |
| PRE31-C | Low | Unlikely | Low | P3 | L3 |
| EXP39-C | Medium | Unlikely | High | P2 | L3 |
| INT50-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STR50-CPP | High | Likely | Medium | P18 | L1 |
| MEM51-CPP | High | Likely | Medium | P18 | L1 |
| DCL51-CPP | Low | Unlikely | Low | P3 | L3 |
| MSC51-CPP | Medium | Likely | Low | P18 | L1 |
| EXP53-CPP | High | Probable | Medium | P12 | L1 |
| ERR62-CPP | Medium | Unlikely | Medium | P4 | L3 |
| CWE-89 | High | Likely | Medium | P18 | L1 |

### 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 | Data at rest should be properly and securely encrypted as the coding language specifies. Data accessed on a database could be obtained by a malicious user and adding an additional layer of protection to company and users’ data supports Principle 8, Practice Defense in Depth. |
| Encryption at flight | Data at flight can be intercepted, read, or manipulated while in transit. This is called a man-in-the-middle attack, and it is why data at flight should be encrypted. Encryption should always be securely implemented as the coding language specifies to protect user data and practice defense in depth. |
| Encryption in use | Data in use can be intercepted, read, or manipulated by a malicious user. While data is being used or processed, it should always remain securely encrypted as the coding language specifies. Encrypting data supports Principle 8, Practice Defense in Depth because it protects user data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The process of authentication determines which user is logging into a system. This policy is implemented first to verify users of an application before they access its features. This is usually accomplished in the form of a login screen. |
| Authorization | Authorization determines which features the user has access to. Users are often given user levels to limit actions a user can make to only what they need. Usually, features that are inaccessible to a user are hidden from them. For example, an administrator account might be able to make changes to a database. Depending on an application’s requirement, either the admin or any user should be able to add users. This policy is implemented after the user has verified themselves. |
| Accounting | Accounting is the process of logging user requests to track where issues occur. Logging data such as user information, timestamps, data accessed, etc. For example, if a user accessed a file, important\_data.txt, at 2:30 A.M., then it should be logged as such in a log file. This policy is implemented after authorization and as the user interacts with the application. |

**\***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 |  |
| 2.0 | 11/18/2022 | The Ten Core Security Principles were added and Coding Standards were started to be defined. | Trevor Leon |  |
| 3.0 | 12/11/2022 | The Security Policy for Green Pace is completed. Risk Assessments; Automated Detection; Automation; and Encryption and Triple A Policies have all been defined. | Trevor Leon |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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