**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)-3

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

[Coding Standard 1 4](#_Toc52464059)-5

[Coding Standard 2 6-7](#_Toc52464060)

[Coding Standard 3 8-9](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 11-12](#_Toc52464063)

[Coding Standard 6 13-14](#_Toc52464064)

[Coding Standard 7 15-17](#_Toc52464065)

[Coding Standard 8 18-19](#_Toc52464066)

[Coding Standard 9 20-21](#_Toc52464067)

[Coding Standard 10 22-23](#_Toc52464068)

[Defense-in-Depth Illustration 24](#_Toc52464069)

[Project One 24](#_Toc52464070)

[1. Revise the C/C++ Standards 24](#_Toc52464071)

[2. Risk Assessment 24](#_Toc52464072)

[3. Automated Detection 24](#_Toc52464073)

[4. Automation 24-25](#_Toc52464074)

[5. Summary of Risk Assessments 25](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 25-26](#_Toc52464076)

[7. Map the Principles 27](#_Toc52464077)

[Audit Controls and Management 28](#_Toc52464078)

[Enforcement 28](#_Toc52464079)

[Exceptions Process 28](#_Toc52464080)

[Distribution 29](#_Toc52464081)

[Policy Change Control 29](#_Toc52464082)

[Policy Version History 29](#_Toc52464083)

[Appendix A Lookups 29](#_Toc52464084)

[Approved C/C++ Language Acronyms 29](#_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 | Certify input from all untrustworthy data sources. Properly check input validation to immediately eliminate most software exploits. Be wary of external data sources to include command line arguments, network interfaces, environmental variables, and user-controlled files. [Seacord 05]. |
| 1. Heed Compiler Warnings | Compile code using the highest maximum warning level possible for your compiler by changing the code [C MSC00-A, C++ MSC00-a]. Utilize static and dynamic analysis tools to detect and eradicate extra security defects. |
| 1. Architect and Design for Security Policies | Build software architecture and layout your software to execute and administer security procedures. An example of such instance is to build subsystems into the architecture for the possibility of different privileges at different times to support each appropriate privilege set. |
| 1. Keep It Simple | Keep the design as plain and small-scale as possible [Saltzer 74, Saltzer 75]. Complicated designs enhance the probability that mistakes will be made in their operation, construction, and purpose. The energy required to attain a suitable level of guarantee improves considerably as security structures become more intricate. |
| 1. Default Deny | Base access rules on permission instead of exclusion. By default, access is refused, and the safety program characterizes circumstances under which access is authorized [Saltzer 74, Saltzer 75]. |
| 1. Adhere to the Principle of Least Privilege | All processes should perform with the minimum set of privileges essential to carry out the job. Every elevated authorization should only be accessed for the minimum amount of time needed to complete the privileged task. |
| 1. Sanitize Data Sent to Other Systems | Clean all data passed to intricate subsystems [C STR02-A] such as command shells, relational databases, and commercial off-the-shell (COTS) components. Invaders may be able to utilize unused functionality in these components using SQL, command, or other injection attacks. |
| 1. Practice Defense in Depth | Control risk with numerous defensive tactics that can protect if one layer of defense turns out to be insufficient, an additional layer of defense can stop a security flaw from becoming an exploitable liability and limit the effects of a successful exploit. [Seacord 05] |
| 1. Use Effective Quality Assurance Techniques | Great quality assurance techniques can be valuable in detecting and removing vulnerabilities. Fuzz testing, penetration testing, and source code audits ought to be included as part of an effective quality assurance program. Exterior reviews bring an independent viewpoint for identifying and correcting invalid assumptions [Seacord 05]. |
| 1. Adopt a Secure Coding Standard | Create and employ a secure coding guideline for your target development language and platform. |

### 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** | **Ensure that operations on signed integers do not result in overflow** |
| --- | --- | --- |
| **Data Value** | [STD-001-CPP] | An application that characterizes signed integer types as being modulo need not identify integer overflow. Applications may also trap on signed arithmetic overflows, or simply suppose that overflows will not happen and produce object code appropriately. It is also probable for the same compliant application to release code that displays different behavior in different contexts. |

| **Noncompliant Code** |
| --- |
| This noncompliant code model can cause a signed integer overflow during the addition of the signed operands si\_a and si\_b: |
| void func(signed int si\_a, signed int si\_b) {  signed int sum = si\_a + si\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution makes sure that the addition operation cannot overflow, irrespective of depiction: |
| #include <limits.h>    void f(signed int si\_a, signed int si\_b) {  signed int sum;  if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||  ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {  /\* Handle error \*/  } else {  sum = si\_a + si\_b;  }  /\* ... \*/  } |

**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** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | Integer-overflow | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  MISC.MEM.SIZE.ADDOFLOW  MISC.MEM.SIZE.BAD  MISC.MEM.SIZE.MULOFLOW  MISC.MEM.SIZE.SUBUFLOW | Addition overflow of allocation size  Integer overflow of allocation size  Multiplication overflow of allocation size  Subtraction underflow of allocation size  Addition overflow of size  Unreasonable size argument  Multiplication overflow of size  Subtraction underflow of size |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2022.3 | C2800, C2801, C2802, C2803, C2860,  C2861, C2862, C2863  C++2800, C++2801, C++2802,  C++2803, C++2860, C++2861,  C++2862, C++2863 | N/A |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that integer conversions do not result in lost or misinterpreted data** |
| --- | --- | --- |
| **Data Type** | [STD-002-CPP] | Integer conversions, mutually implicit and explicit (using a cast), must be guaranteed not to end in lost or misinterpreted data. This rule is especially true for integer values that come from untrusted sources. |

| **Noncompliant Code** |
| --- |
| Type range errors, including loss of data (truncation) and loss of sign (sign errors), can happen when switching from a value of an unsigned integer type to a value of a signed integer type. |
| #include <limits.h>    void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Authenticate ranges when switching from an unsigned type to a signed type. |
| #include <limits.h>    void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  if (u\_a <= SCHAR\_MAX) {  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  } else {  /\* Handle error \*/  }  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | N/A | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | LANG.CAST.PC.AV  LANG.CAST.PC.CONST2PTR  LANG.CAST.PC.INT  LANG.CAST.COERCE  LANG.CAST.VALUE  ALLOC.SIZE.TRUNC  MISC.MEM.SIZE.TRUNC  LANG.MEM.TBA | Cast: arithmetic type/void pointer  Conversion: integer constant to pointer  Conversion: pointer/integer  Coercion alters value  Cast alters value  Truncation of allocation size  Truncation of size  Tainted buffer access |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) | N/A | N/A | Can detect violations of this rule. However, false warnings may be raised if limits.h is included |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | NEGATIVE\_RETURNS  REVERSE\_NEGATIVE  MISRA\_CAST | Can find array accesses, loop bounds, and other expressions that may contain dangerous implied integer conversions that would result in unexpected behavior  Can find instances where a negativity check occurs after the negative value has been used for something else  Can find instances where an integer expression is implicitly converted to a narrower integer type, where the signedness of an integer value is implicitly converted, or where the type of a complex expression is implicitly converted |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Duplicating data to a buffer that is not big enough to store that data results in a buffer overflow. Buffer overflows happen often when manipulating strings. To avoid such errors, either restrict copies through truncation or, if possible, ensure that the destination is of adequate size to hold the character data to be duplicated and the null-termination character. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows an off-by-one error because the loop duplicates data from src to dest. The loop example does not account for the null-termination character and can be inaccurately transcribed 1 byte past the end of dest. |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {  size\_t i;    for (i = 0; src[i] && (i < n); ++i) {  dest[i] = src[i];  }  dest[i] = '\0';  } |

| **Compliant Code** |
| --- |
| The loop termination requirement is adjusted to account for the null-termination character that is attached to dest. |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {  size\_t i;    for (i = 0; src[i] && (i < n - 1); ++i) {  dest[i] = src[i];  }  dest[i] = '\0';  } |

**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** |
| --- | --- | --- | --- | --- |
| 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 |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | C++2835, C++2836, C++2839, C++5216 | N/A |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2022.3 | C++2835, C++2836, C++2839, C++5216 | N/A |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Declare identifiers before using them** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Verifying input and having parametrized queries produces a security data access layer to the structure instead of passing along user input queries that may disrupt the database by SQL injections. |

| **Noncompliant Code** |
| --- |
| This example does not include the specifier type. |
| extern foo; |

| **Compliant Code** |
| --- |
| This example includes the specifier type. |
| Extern int foo; |

**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 | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| use warnings; | N/A | N/A | Name .\* used only once. possible typo |
| use strict | N/A | N/A | Global symbol .\* requires explicit package name |
| Perl::Critic | N/A | N/A | Policy::TestingAndDebugging::RequireUseWarnings  Policy::TestingAndDebugging::RequireUseStrict |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Free dynamic allocated memory when no longer needed** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Prior to the lifetime of the most recent pointer that saves the return value of a call to a standard memory allocation function has ended, it must be paired by a call to free() with that pointer value. |

| **Noncompliant Code** |
| --- |
| The item given by the call to malloc() is not free before the end of the lifetime of the last pointer text\_buffer referring to the object. |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }  return 0;  } |

| **Compliant Code** |
| --- |
| The pointer is deallocated with a call to free(). |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }    free(text\_buffer);  return 0;  } |

**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 | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | N/A | Supported, but no explicit checker |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-MEM31 | Can detect dynamically allocated resources that are not freed |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | ALLOC.LEAK | Leak |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) | N/A | N/A | N/A |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Assertions** | [STD-006-CCP] | Assertions are a useful diagnostic tool for discovering and removing software deficiencies that could result in vulnerabilities. The runtime assert() macro has some shortcomings, however, in that it encounters a runtime overhead and because it calls abort(). Subsequently, the runtime assert() macro is effective only for recognizing false assumptions and not for runtime error checking. |

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

| **Compliant Code** |
| --- |
| For assertions consist of only constant expressions, a preprocessor conditional statement may be applied. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-DCL03 | N/A |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) | N/A | N/A | 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 |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Guarantee exception safety** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Appropriate handling of errors and exceptional situations is vital for the continued correct operation of software. The ideal mechanism for reporting errors in a C++ program is exceptions instead of error codes. A number of basic language facilities, including dynamic\_cast, operator new(), and typeid, state failures by throwing exceptions and most C++ programs must be prepared for exceptions to happen and must handle each properly. |

| **Noncompliant Code** |
| --- |
| The implicit invariants of the class are that the array member is a valid (possibly null) pointer and that the nElems member stores the number of elements in the array pointed to by array. The function deallocates array and appoints the element counter, nElems, before allocating a new space of memory for the copy. When this happens, if the new expression throws an exception, the function will have altered the state of both member variables in a way that damages the implicit invariants of the class. |
| #include <cstring>    class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }    IntArray(const IntArray& that); // nontrivial copy constructor  IntArray& operator=(const IntArray &rhs) {  if (this != &rhs) {  delete[] array;  array = nullptr;  nElems = rhs.nElems;  if (nElems) {  array = new int[nElems];  std::memcpy(array, rhs.array, nElems \* sizeof(\*array));  }  }  return \*this;  }    // ...  }; |

| **Compliant Code** |
| --- |
| The copy assignment operator offers a strong exception safety assurance. The function allocates new space for the copy before altering the state of the object. Only when the allocation is a success, does the function continue to alter the state of the object. |
| #include <cstring>    class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }    IntArray(const IntArray& that); // nontrivial copy constructor    IntArray& operator=(const IntArray &rhs) {  int \*tmp = nullptr;  if (rhs.nElems) {  tmp = new int[rhs.nElems];  std::memcpy(tmp, rhs.array, rhs.nElems \* sizeof(\*array));  }  delete[] array;  array = tmp;  nElems = rhs.nElems;  return \*this;  }    // ...  }; |

**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** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | ALLOC.LEAK | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | C++4075, C++4076 | N/A |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 527 S, 56 D, 71 D | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | CERT\_CPP-ERR56-a  CERT\_CPP-ERR56-b | Always catch exceptions  Do not leave 'catch' blocks empty |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Use valid iterator ranges** |
| --- | --- | --- |
| Containers | [STD-008-CPP] | While iterating over elements of a container, the iterators applied should iterate over a valid range. An iterator range is a set of iterators that belong to the first and past-the-end elements of the range in that order. |

| **Noncompliant Code** |
| --- |
| Two iterators that delimit the range point into the same container, but the first iterator does not go before the second. On each iteration of its internal loop, std::for\_each() assesses the first iterator (after incrementing it) with the second. If the two iterators are not equal, it will continue to increment the first iterator the past-the-end element of the range results in indeterminate behavior. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.end(), c.begin(), [](int i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| The iterator values passed to std::for\_each() are passed in the correct sequence. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.begin(), c.end(), [](int i) { std::cout << i; });  } |

**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** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | overflow\_upon\_dereference | N/A |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | LANG.MEM.BO | Buffer Overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | C++3802 | N/A |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | CERT\_CPP-CTR53-a  CERT\_CPP-CTR53-b | Do not use an iterator range that isn't really a range  Do not compare iterators from different containers |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do no rely on side effects in unevaluated operands** |
| --- | --- | --- |
| Exceptions | [STD-009-CPP] | The subsequent expressions do not evaluate their operands: sizeof(), typeid(), noexcept(), decltype(), and declval(). Since an unevaluated operand in an expression is not evaluated, no side effects from that operand are prompted. Dependence on those side effects will end in unforeseen behavior. Do not rely on side effects in unevaluated operands. |

| **Noncompliant Code** |
| --- |
| The expression a++ is not evaluated. |
| #include <iostream>  void f() {  int a = 14;  int b = sizeof(a++);  std::cout << a << ", " << b << std::endl;  } |

| **Compliant Code** |
| --- |
| The variable a is incremented outside of the sizeof operator. |
| #include <iostream>  void f() {  int a = 14;  int b = sizeof(a);  ++a;  std::cout << a << ", " << b << std::endl;  } |

**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 | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | sizeof | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-EXP52 | N/A |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wunevaluated-expression | N/A |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | LANG.STRUCT.SE.SIZEOF | Side Effects in sizeof |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Never hard code sensitive information** |
| --- | --- | --- |
| Miscellaneous | [STD-010-CPP] | Hard coding sensitive information, such as passwords or encryption keys could leak the information to attackers. An individual who can gain access to the executable or dynamic library files could inspect them for strings or other critical data, uncovering the sensitive information. Leaking data protected by International Traffic in Arms Regulations (ITAR) or the Health Insurance Portability and Accountability Act (HIPAA) can also have legal consequences. Therefore, programs must not hard code sensitive information. |

| **Noncompliant Code** |
| --- |
| The code needs to authenticate to a remote service using a code and using the authenticate() function in the example. The example code transfers the authentication code to the function as a string literal. |
| /\* Returns nonzero if authenticated \*/  int authenticate(const char\* code);    int main() {  if (!authenticate("correct code")) {  printf("Authentication error\n");  return -1;  }    printf("Authentication successful\n");  // ...Work with system...  return 0;  } |

| **Compliant Code** |
| --- |
| The example enforces that the user needs to provide the authentication code, and after using the code it will get deleted using memset\_s() which is an alternate function supplied in C11’s Annex K. |
| /\* Returns nonzero if authenticated \*/  int authenticate(const char\* code);    int main() {  #define CODE\_LEN 50  char code[CODE\_LEN];  printf("Please enter your authentication code:\n");  fgets(code, sizeof(code), stdin);  int flag = authenticate(code);  memset\_s(code, sizeof(code), 0, sizeof(code));  if (!flag) {  printf("Access denied\n");  return -1;  }  printf("Access granted\n");  // ...Work with system...  return 0;  } |

**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** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | HARDCODED.AUTH  HARDCODED.DNS  HARDCODED.KEY  HARDCODED.SALT  HARDCODED.SEED | Hardcoded Authentication  Hardcoded DNS Name  Hardcoded Crypto Key  Hardcoded Crypto Salt  Hardcoded Seed in PRNG |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2022.3 | C3122  C++3842 | N/A |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2022.3 | HCC  HCC.PWD  HCC.USER | N/A |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | CERT\_C-MSC41-a | Do not hard code string literals |

### 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.

[Insert your written explanations here.]

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

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Sensitive data that is stored in a restful state needs to have encryption applied for defense from any unauthorized access. The process of encryption obscures the plaintext data into ciphertext using an algorithm. The information can only be deciphered with a decryption key. The intention is to have an extra safeguard to protect against a security or data breach by adding to the defense in depth process. |
| Encryption at flight | Encrypted data that moves across the network is considered data in flight. The encryption plays an important role of concealment of data between the sending party and receiving party. Both parties have their own unique encryption and decryption keys to be able to read the converted ciphertext into plaintext. The best practice for encryption in flight is to use a virtual private network. This encrypts information that is moving through cloud applications and data services from any potential threats out on the internet. |
| Encryption in use | Data that is being accessed by a user or application is data in use. This form of data is the most vulnerable because the data is in a clear text form in the memory for the time of use by user or the application. Encrypting data end-to-end and only decrypting the information on the client side is a safer way to store sensitive data in use. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication offers a means of distinguishing an individual. A common practice is by having the user submit a username and password that has been authenticated for validity before gaining entry to the network. Authentication is established on every single user having their own set of exclusive login credentials for access to the network. Authentication is the first part of the Triple-A framework that provides a level of security to keep unauthorized users off the network. |
| Authorization | The main goal of authorization is to provide certain level of access control to the user. Each user is granted a certain amount of authorization on what can be performed on the network such as installing applications or access to sensitive directory files. The importance of authorization is that not every user will have full access to the network but enough to fulfill their job requirements. Authorization provides the second level of security within the Triple-A framework. |
| Accounting | Accounting is the collecting and logging of information of a device on a network. The information is then sent back to a server for the purpose of auditing, billing, and reporting. The benefit of accounting is being able to examine the logs of all management sessions used to assess the network. This is useful to produce reports for either troubleshooting and/or inspecting. |

**\***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 | 11/12/2022 | Milestone One | Alex Sandoval | Aaron Demory |
| 3.0 | 12/01/2022 | Project One | Alex Sandoval | Aaron Demory |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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