**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validating proper input data can eliminate the vast majority of software vulnerabilities. SQL Injection attacks can be prevented by being cautious of third party and external data inputs. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available as much as possible to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies in the initial stages of the software design planning process. This will prevent unexpected surprises and breaches in the software from having a catastrophic effect if it were to be found. |
| 1. Keep It Simple | Complex designs increase the likelihood that errors will be made in their execution. Keeping the design as simple and small as possible will make going back and fixing faulty code much easier. |
| 1. Default Deny | Base access decisions on permission rather than exclusion. This means that, by default, access is denied. For example, new hires will not have access to the hire tiers of security coverage and will gain access overtime as they prove they are trustworthy. |
| 1. Adhere to the Principle of Least Privilege | Following up on Default Deny, Adhering to the Principle of Least Privilege means that every process should execute with the least set of privileges necessary to complete the job. Elevated permission should only be granted for the time deemed required to complete the task. |
| 1. Sanitize Data Sent to Other Systems | Command shells, relational databases, and commercial off-the-shelf (COTS) components are all complex subsystems that data should run through before being sent to other systems. |
| 1. Practice Defense in Depth | Secure coding practices should be executed at all times with top priority. Risk can be managed with multiple defensive strategies and layers. For example, many different preparation scenarios like what to do if an attack does happen or how to prevent different types of attacks. |
| 1. Use Effective Quality Assurance Techniques | Fuzz testing, penetration testing, and source code audits are some of the effective quality assurance techniques that can identify and eliminate vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Secure coding practices should be executed at all times with top priority, while this is part of practicing defense in depth, all of these principles work in conjunction to maintain a secure application. |

### 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] | DCL50-CPP. Do not define a C-style variadic function |

| **Noncompliant Code** |
| --- |
| This function reads the arguments until the 0 value is found or if it isn’t found after two arguments it can causes undefined behavior. |
| #include <cstdarg>    int add(int first, int second, ...) {  int r = first + second;  va\_list va;  va\_start(va, second);  while (int v = va\_arg(va, int)) {  r += v;  }  va\_end(va);  return r;  } |

| **Compliant Code** |
| --- |
| Rather than having undefined behavior, a variadic function using a function parameter pack is used to implement the add() function, allowing identical behavior for call sites and the parameters is not terminated with 0. Furthermore, non-integer values that pass through the function are ill-informed rather than undefined behavior. |
| #include <type\_traits>    template <typename Arg, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  int add(Arg f, Arg s) { return f + s; }    template <typename Arg, typename... Ts, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  int add(Arg f, Ts... rest) {  return f + add(rest...);  } |

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

| **Principles(s):**  Validating Input Data: Non-integer values that pass through the function are ill-informed rather than causing undefined behavior.  Keep It Simple: A variadic function using a function parameter pack is used to implement the add() function, allowing identical behavior for call sites and the parameters is not terminated with 0. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | function-ellipsis | Fully checked |
| Clang | 3.9 | cert-dcl50-cpp | Checked by clang-tidy. |
| CodeSonar | 7.1p0 | LANG.STRUCT.ELLIPSIS | Ellipsis |
| LDRA tool suite | 9.7.1 | 41 S | Fully Implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002- CPP] | DCL37-C. Do not declare or define a reserved identifier |

| **Noncompliant Code** |
| --- |
| Naming standards aren’t met in the code below which may cause undefined behavior. The example below has characters (underbars) that may clash with reserved names defined by the CPP standard library. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    /\* Contents of <my\_header.h> \*/    #endif /\* \_MY\_HEADER\_H\_ \*/ |

| **Compliant Code** |
| --- |
| Undefined behavior is prevented by removing the trailing and leading underbars. |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    /\* Contents of <my\_header.h> \*/    #endif /\* MY\_HEADER\_H \*/ |

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

| **Principles(s):**  Heed Compiler Warnings: The highest warning level available could’ve detected extra characters in the headers that would clash with reserved names defined by the CPP standard library.  Use Effective QA Techniques: Source code audits can prevent clashing of header names reserved by the CPP standard library. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | future-library-use  language-override  language-override-c99  reserved-declaration  reserved-declaration-c99  reserved-identifier | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL37 | Fully implemented. Reserved identifiers, as in DCL37-C-EX3, are configurable. |
| CodeSonar | 7.1p0 | LANG.STRUCT.DECL.RESERVED | Declaration of reserved name |
| Coverity | 2017.07 | MISRA C 2004 Rule 20.1  MISRA C 2004 Rule 20.2  MISRA C 2012 Rule 21.1  MISRA C 2012 Rule 21.2 | Implemented |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003- CPP] | DCL52-CPP. Never qualify a reference type with const or volatile |

| **Noncompliant Code** |
| --- |
| a const-qualified reference to a char is formed instead of a reference to a const-qualified char which results in undefined behavior. |
| #include <iostream>    void f(char c) {  char &const p = c;  p = 'p';  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| The const qualifier is removed to prevent an undefined behavior from occurring. |
| #include <iostream>  void f(char c) {  char &p = c;  p = 'p';  std::cout << c << std::endl;  } |

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

| **Principles(s):**  Heed compiler warnings: Warnings that undefined behavior occurring because of the const qualifier is added.  Keep it simple: Const qualifier is removed to prevent undefine behavior.  Use effective QA techniques: Repeated fuzz testing can help find a solution to this problem since it’ll lead to undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-DCL52-a | Never qualify a reference type with 'const' or 'volatile' |
| Polyspace Bug Finder | R2022b | CERT C++: DCL52-CPP | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| Clang | 3.9 |  | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options. |
| Klocwork | 2022.3 | CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004- CPP] | DCL53-CPP. Do not write syntactically ambiguous declarations |

| **Noncompliant Code** |
| --- |
| The argument can be used to declare an anonymous object since it is syntactically ambiguous. The syntax used in the example defines the latter rather than the former which does not lock the mutex object. |
| #include <mutex>  static std::mutex m;  static int shared\_resource;  void increment\_by\_42() {  std::unique\_lock<std::mutex>(m);  shared\_resource += 42;  } |

| **Compliant Code** |
| --- |
| The mutex object is properly locked and proper converting constructor is called. |
| #include <mutex>  static std::mutex m;  static int shared\_resource;  void increment\_by\_42() {  std::unique\_lock<std::mutex> lock(m);  shared\_resource += 42;  } |

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

| **Principles(s):**  Validate Input Data: The user input is can be defined by the wrong object if the mutex object isn’t properly locked.  Heed Compiler Warning: Implementation of a warning that the mutex object isn’t properly locked. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.STRUCT.DECL.FNEST | Nested Function Declaration |
| LDRA tool suite | 9.7.1 | 296 S | Partially implemented |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-DCL53-a  CERT\_CPP-DCL53-b  CERT\_CPP-DCL53-c | Parameter names in function declarations should not be enclosed in parentheses  Local variable names in variable declarations should not be enclosed in parentheses  Avoid function declarations that are syntactically ambiguous |
| Polyspace Bug Finder | R2022b | CERT C++: DCL53-CPP | Checks for declarations that can be confused between:   * Function and object declaration * Unnamed object or function parameter declaration   Rule fully covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-  CPP] | DCL54-CPP. Overload allocation and deallocation functions as a pair in the same scope |

| **Noncompliant Code** |
| --- |
| The allocation function below is overloaded at global scope but there is not a deallocation function declared. If an object were to be allocated with the overloaded allocation, then attempting to delete the object would cause an undefined behavior. |
| #include <Windows.h>  #include <new>    void \*operator new(std::size\_t size) noexcept(false) {  static HANDLE h = ::HeapCreate(0, 0, 0); // Private, expandable heap.  if (h) {  return ::HeapAlloc(h, 0, size);  }  throw std::bad\_alloc();  }    // No corresponding global delete operator defined. |

| **Compliant Code** |
| --- |
| Undefined behavior and an overloaded condition can be prevented by declaring a deallocation function. |
| #include <Windows.h>  #include <new>    class HeapAllocator {  static HANDLE h;  static bool init;    public:  static void \*alloc(std::size\_t size) noexcept(false) {  if (!init) {  h = ::HeapCreate(0, 0, 0); // Private, expandable heap.  init = true;  }    if (h) {  return ::HeapAlloc(h, 0, size);  }  throw std::bad\_alloc();  }    static void dealloc(void \*ptr) noexcept {  if (h) {  (void)::HeapFree(h, 0, ptr);  }  }  };    HANDLE HeapAllocator::h = nullptr;  bool HeapAllocator::init = false;    void \*operator new(std::size\_t size) noexcept(false) {  return HeapAllocator::alloc(size);  }    void operator delete(void \*ptr) noexcept {  return HeapAllocator::dealloc(ptr);  } |

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

| **Principles(s):**  Heed Compiler Warnings: Warning the user that any attempt to delete the object would result in undefined behavior.  Use Effective QA Techniques: Fuzz testing and source code audits would be able to identify that undefined behavior may occur. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | new-delete-pairwise | Partially checked |
| Clang | 3.9 | misc-new-delete-overloads | Checked with clang-tidy. |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-DCL54-a | Always provide new and delete together |
| Polyspace Bug Finder | R2022b | CERT C++: DCL54-CPP | Checks for mismatch between overloaded operator new and operator delete (rule fully covered) |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006- CPP] | DCL55-CPP. Avoid information leakage when passing a class object across a trust boundary |

| **Noncompliant Code** |
| --- |
| The code below copies data from arg to user space. The padding bits being used to ensure the proper alignment of class data members may leak sensitive information when it is being copied to user space. |
| #include <cstddef>    struct test {  int a;  char b;  int c;  };    // Safely copy bytes to user space  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void do\_stuff(void \*usr\_buf) {  test arg{1, 2, 3};  copy\_to\_user(usr\_buf, &arg, sizeof(arg));  } |

| **Compliant Code** |
| --- |
| Padding Bytes can be declared as fields within the structure. This will also ensure that the sensitive data is serialized before copying it to an untrusted context. |
| #include <cstddef>    struct test {  int a;  char b;  char padding\_1, padding\_2, padding\_3;  int c;    test(int a, char b, int c) : a(a), b(b),  padding\_1(0), padding\_2(0), padding\_3(0),  c(c) {}  };  // Ensure c is the next byte after the last padding byte.  static\_assert(offsetof(test, c) == offsetof(test, padding\_3) + 1,  "Object contains intermediate padding");  // Ensure there is no trailing padding.  static\_assert(sizeof(test) == offsetof(test, c) + sizeof(int),  "Object contains trailing padding");        // Safely copy bytes to user space.  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void do\_stuff(void \*usr\_buf) {  test arg{1, 2, 3};  copy\_to\_user(usr\_buf, &arg, sizeof(arg));  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems: Padding Bytes can be declared as fields within the structure.  Adopt a Secure Coding Standard: Padding bytes will serialize the sensitive data. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL55 |  |
| CodeSonar | 7.1p0 | MISC.PADDING.POTB | Padding Passed Across a Trust Boundary |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-DCL55-a | A pointer to a structure should not be passed to a function that can copy data to the user space |
| Polyspace Bug Finder | R2022b | CERT C++: DCL55-CPP | Checks for information leakage due to structure padding (rule partially covered) |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-  CPP] | ERR56-CPP. Guarantee exception safety |

| **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 assigns the element counter, nElems, before allocating a new block of memory for the copy which results in undefined behavior. |
| #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 provides the strong exception safety guarantee by allocating new storage for the copy before changing the state of the object. The function avoids the test for self-assignment which initially resulted in undefined behavior. |
| #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):**  Validate Input Data: The copy assignment operator allocates new storage for the copy before changing the state of the object.  Use Effective QA Techniques: The function added avoids the test for self-assignment which initially resulted in undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | ALLOC.LEAK | Leak |
| LDRA tool suite | 9.7.1 | 527 S, 56 D, 71 D | Partially implemented |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-ERR56-a  CERT\_CPP-ERR56-b | Always catch exceptions  Do not leave 'catch' blocks empty |
| Polyspace Bug Finder | R2022b | CERT C++: ERR56-CPP | Checks for exceptions violating class invariant (rule fully covered). |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Constructor member initializers | [STD-008-  CPP] | OOP53-CPP. Write constructor member initializers in the canonical order |

| **Noncompliant Code** |
| --- |
| The declaration order of the member variables does not match the member initializer order which will result in an unspecified value being stored in dependsOnSomeVal. |
| class C {  int dependsOnSomeVal;  int someVal;    public:  C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| The declaration order is changed to match the class member variables in order for the dependency to be ordered properly in the member initializer list. |
| class C {  int someVal;  int dependsOnSomeVal;    public:  C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

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

| **Principles(s):**  Validate Input Data: The declaration order of the member variables does not match the member initializer order.  Keep It Simple: Declare the order and member initializer at the same time. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | initializer-list-order | Fully checked |
| CodeSonar | 7.1p0 | LANG.STRUCT.INIT.OOMI | Out of Order Member Initializers |
| LDRA tool suite | 9.7.1 | 206 S | Fully implemented |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-OOP53-a | List members in an initialization list in the order in which they are declared |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Error Detection | [STD-009-  CPP] | ERR62-CPP. Detect errors when converting a string to a number |

| **Noncompliant Code** |
| --- |
| The code example below is intended to convert multiple numeric values from the standard input streak. However, if the user’s input can not be represented by int like a numeric value can, then it may cause undefined behavior. |
| #include <iostream>    void f() {  int i, j;  std::cin >> i >> j;  // ...  } |

| **Compliant Code** |
| --- |
| An exception is enabled so that any conversions that can not be represented by int will result in it being thrown. |
| #include <iostream>    void f() {  int i, j;    std::cin.exceptions(std::istream::failbit | std::istream::badbit);  try {  std::cin >> i >> j;  // ...  } catch (std::istream::failure &E) {  // Handle error  }  } |

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

| **Principles(s):**  Validate Input Data: An exception is enabled so that any conversions that cannot be represented by int will result in it being thrown.  Heed Compiler Warnings: Warning that any input that isn’t an int may cause undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | cert-err34-c | Checked by clang-tidy; only identifies use of unsafe C Standard Library functions corresponding to ERR34-C |
| CodeSonar | 7.1p0 | BADFUNC.ATOF  BADFUNC.ATOI  BADFUNC.ATOL  BADFUNC.ATOLL | Use of atof  Use of atoi  Use of atol  Use of atoll |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-ERR62-a | The library functions atof, atoi and atol from library stdlib.h shall not be used |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR62 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Valid Input and Output format | [STD-010-  CPP] | FIO47-C. Use valid format strings |

| **Noncompliant Code** |
| --- |
| Undefined behavior may occur if there are mismatches between arguments and conversion specifications. the error\_type argument to printf() is incorrectly matched with the s specifier rather than with the s specifier. Likewise, the error\_msg argument is incorrectly matched with the d specifier instead of the s specifier. |
| #include <stdio.h>    void func(void) {  const char \*error\_msg = "Resource not available to user.";  int error\_type = 3;  /\* ... \*/  printf("Error (type %s): %d\n", error\_type, error\_msg);  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The arguments to the printf() function now match their respective conversion specifications. |
| #include <stdio.h>    void func(void) {  const char \*error\_msg = "Resource not available to user.";  int error\_type = 3;  /\* ... \*/  printf("Error (type %d): %s\n", error\_type, error\_msg);    /\* ... \*/  } |

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

| **Principles(s):**  Heed Compiler Warnings: Warning that undefined behavior may occur if there are mismatches between arguments and conversion specifications.  Use Effective QA techniques: The printf() function must match their respective conversion specifications. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO47 | Fully implemented |
| CodeSonar | 7.1p0 | IO.INJ.FMT  MISC.FMT  MISC.FMTTYPE | Format string injection  Format string  Format string type error |
| Coverity | 2017.07 | PW | Reports when the number of arguments differs from the number of required arguments according to the format string |
| GCC | 4.3.5 |  | Can detect violations of this recommendation when the -Wformat flag is used |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

Green Pace’s DevOps process already contains the most important components of anticipating, designing, and responding to a security process. However, encorporating Automation is a crucial process of DevSecOps because security becomes an integral part of the development process instead of an afterthought. Automation can decrease manual errors and creates a more transparent development life cycle. Automation should start in the Assess and plan stage in pre-production since that is generally where development starts. The security process should start as soon as possible and test as much as possible to detect any 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 |
| --- | --- | --- | --- | --- | --- |
| [STD-001-  CPP] | High | Probable | Medium | P12 | L1 |
| [STD-002-  CPP] | Low | Unlikely | Low | P3 | L3 |
| [STD-003-  CPP] | Low | Unlikely | Low | P3 | L3 |
| [STD-004-  CPP] | Low | Unlikely | Medium | P2 | L3 |
| [STD-005-  CPP] | Low | Probable | Low | P6 | L2 |
| [STD-006-  CPP] | Low | Unlikely | High | P1 | L3 |
| [STD-007-  CPP] | High | Likely | High | P9 | L2 |
| [STD-008-  CPP] | Medium | Unlikely | Medium | P4 | L3 |
| [STD-009-  CPP] | Medium | Unlikely | Medium | P4 | L3 |
| [STD-010-  CPP] | High | Unlikely | Medium | P6 | 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 | Encryption at rest is stored data that’s already encoded to prevent unauthorized users from being able to decrypt and access the data. The Encryption at rest process starts by encoding the data while it’s being stored and only an authorized user with an encryption key will be able to decode the data as it’s being pulled. Encryption at rest should be used because it can protect sensitive information and it limits data access to only authorized users that have an encryption key. |
| Encryption in flight | Encryption in flight is the process of encrypting the data as it’s being transmitted. Encryption in flight should be used because it can protect sensitive data while it’s being transmitted. An example of this is sending and receiving data through email and the encryption will prevent the information from being intercepted. |
| Encryption in use | Encryption in use is the process of securing data in protected memory. Although user passwords are confidential and protected, encryption in use allows for the recognition of the user’s input to what is currently encrypted as the password without it being compromised. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of identifying a user by their username and password before network access is granted. If a user enters valid credentials that is stored in a database, then they are granted network access. If the credentials aren’t valid, then network access is denied. |
| Authorization | Authorization is the process of enforcing policies granted to the user by allowing or denying their access to certain services and resources such as drives and folders. |
| Accounting | Accounting is the process of monitoring the resources that is consumed by an authorized user during their network access. |

**\***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 | 12/04/2022 | Project One | Daniel Pham |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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