**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 the inputs of untrusted data helps prevent any invulnerabilities of software. By validating the inputs, we can confirm the sources of the data as well as any other external data. |
| 1. Heed Compiler Warnings | When developing code, a couple warnings are shown to help identify any potential invulnerabilities. It is important to stay up to date and modify the code to address the warnings. The warnings are shown to help keep coding secure. |
| 1. Architect and Design for Security Policies | When designing software architecture, it is important to follow the security policies to keep the code consistent and up to date. |
| 1. Keep It Simple | When designing code, it is best to keep the code as simple as you can to allow for any modifications in the future to be easily managed. Adding cluttered code will result in multiple errors or potential invulnerabilities. |
| 1. Default Deny | Keep a standard that you follow; By standardizing a specific way, it is easy to track any imperfections in the code and maintain it. |
| 1. Adhere to the Principle of Least Privilege | Keep the access for the code in processes to a bare minimum. Allow for only the amount of time needed to address any process. This will help keep the invulnerabilities to a minimum as it will not be accessed by many. |
| 1. Sanitize Data Sent to Other Systems | Sanitation will address injection attacks. Sanitizing data before it is transferred or sent will help filter information. |
| 1. Practice Defense in Depth | Always implement multiple layers of security as it allows for an increase in level of safety and protection. |
| 1. Use Effective Quality Assurance Techniques | Implement a process where it confirms the quality of the code to ensure the code is secure. The process should include testing the code, reviewing any errors, and checking any functions. |
| 1. Adopt a Secure Coding Standard | Keep a secure coding standard by adjusting to the language that is being used. |

### 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** | **Don’t cast to an enumeration value that is out of range** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Two forms of Enumerations: fixed and unscoped. The form of enumeration may include enumerator values that are not specified. The range is defined by the C++ standard. |

| **Noncompliant Code** |
| --- |
| Checks the value to see if it is within the enum value range. It is checking by casting the enumeration type but it may not be able to represent an integer value. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  EnumType enumVar = static\_cast <EnumType>(intVar);  If (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| Checks the value of enum prior to the conversion, to establish an output without an error. The result should not be an unspecific value. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  If (enumVar < First || enumVar > 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):** Values are unspecified which may cause a buffer overflow and allow for an attacker to use code to breach. This will lead to data integrity issues due to enumerators rarely being used for indexing arrays. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Cast-integer-to-enum | Partially checked |
| CodeSonar | 7.3p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| RuleChecker | 22.10 | Cast-integer-to-enum | Partially checked |
| Parasoft C/C+  +test | 2022.2 | CERT\_CPP-INT50-a | Expression with enum underlying type shall only have values corresponding to enumerators |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Use valid pointers, references, and iterators to reference elements of a basic\_string** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | As std::basic\_string is a container of characters, the CTR51-CPP rule must be followed to help support any iterations in other containers. |

| **Noncompliant Code** |
| --- |
| The code example copies the input into a std::string, which replaces the semicolon (;). It is noncompliant due to the iterator loc being invalidated after the first insert(). When calling to the insert(), it is undefined. |
| #include <string>  void f(const std::string &input) {  std::string email;  // Copy input into email  std::string::iterator loc = email.begin();  for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {  email.insert(loc, \*i != ';' ? \*i : ' ');  }  } |

| **Compliant Code** |
| --- |
| In the code example, the value of the iterator loc is updated as the result of each call insert() which allows for the invalidated iterator to not be accessed. |
| #include <string>  void f(const std::string &input) {  std::string email;  // Copy input into email  std::string::iterator loc = email.begin();  for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {  loc = email.insert(loc, \*i != ';' ? \*i : ' ');  }  } |

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

| **Principles(s):** Using invalid pointers, references, iterators to reference elements of the basic\_string which result in undefined values. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2023.1 | DF4746, DF4747, DF4748, DF4749 |  |
| CodeSonar | 7.3p0 | ALLOC.UAF | Use After Free |
| Parasoft C/C+  + test | 2022.2 | CERT\_CPP-STR52-a | User valid pointers, references, and iterators for basic\_string |
| Polyspace Bug Finder | R2023a | CERT C++:STR52-CPP | Checks for use of invalid string iterator |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Check the range of element access** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | The index operators such as the std::string and the const\_reference operator[](size type) const will return the characters stored in the location specified, pos. The index operators are not checked for ranges so when it is changed to anything out of its range, it will result in undefined errors. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code, the value returned by the call to get\_index() could possibly be greater than the number of elements which are stored in the string. This will result in undefined behavior from the value. |
| #include <string>  extern std::size\_t get\_index();  void f() {  std::string s("01234567");  s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| The compliant code shows that the function std::basic\_string::at(), where it will act similar to the index operator[] and will throw a std::out\_of\_range |
| #include <stdexcept>  #include <string>  extern std::size\_t get\_index();  void f() {  std::string s("01234567");  try {  s.at(get\_index()) = '1';  } catch (std::out\_of\_range &) {  // Handle error  }  } |

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

| **Principles(s):** Unchecked element access can lead to the out-of-bound read of values and can exploit data. This leaves the code vulnerable. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA | Buffer overrun  Buffer underrun  Tainted buffer access |
| Parasoft C/C+  + test | 2022.2 | CERT\_CPP-STR53-A | Guarantee container indices are in range |
| Astree | 22.10 | Assert\_failure |  |
| Polyspace Big Finder | R2023a | CERT C++:STR53-CPP | Checks for:  Array access with tainted index  Array access out of bound  Pointer dereference |

#### Coding Standard 4

| **Coding Standard** | **Label** | **The smart pointer value should not be stored in the already-owned pointer.** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Smart pointers such as std::shared\_ptr and std::unique\_ptr will wrap a pointer value by using the operators ->() and \*(). It can be managed to control where it is constructed. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code, it shows two unrelated smart pointers that are constructed from the same pointer value. As the automatic variable p2 is destroyed, it will delete the pointer value that is being managed. As the automatic variable p1 is destroyed, it will also be deleting the same value. |
| #include <memory>  void f() {  int \*i = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i);  } |

| **Compliant Code** |
| --- |
| In the compliant code, the std::shared\_ptr objects are similar and related due to the copy construction. Here, when the automatic variable p2 is destroyed, the value for the use count is decremented but remains nonzero. Also, when automatic variable p1 is destroyed, the use count for the shared value is decremented to zero and the pointer that was managed by it is destroyed. |
| #include <memory>  void f() {  std::shared\_ptr<int> p1 = std::make\_shared<int>();  std::shared\_ptr<int> p2(p1);  } |

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

| **Principles(s):** Passing the pointer value to a deallocated function that does not match will result in undefined behavior and lead to vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C+  + test | 2022.2 | CERT\_CPP-MEM56-A | Do not store an already-owned pointer value in an unrelated smart pointer |
| Axivion | 7.2.0 | CertC++.MEM56 |  |
| PVS-Studio | 7.24 | V1006 |  |
| Polyspace-Studio | R2023a | CERT C++:MEM56-CPP | Checks for use of already-owned pointer |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Ensure all allocated resources are deallocated correctly.** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Allocating memory in the programming language of C can be done in many ways such as std::calloc() or std::realloc(). There is only one way to free the allocated memory and it is by using std::free(). The rules are seen in MEM34-C and the requirements are shown as well. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, the local variable space is passed to the new operator while the resulting pointer of that call is passed to ::operator delete(). This will create undefined behavior and attempt to free memory that was not returned. |
| #include <iostream>  struct test {  test() { std::cout << "test::test()" << std::endl; }  ~S() { std::cout << "test::~test()" << std::endl; }  };  void func() {  alignas(struct test) char space[sizeof(struct test)];  test \*test1 = new (&space) test;  // ...  delete test1;  } |

| **Compliant Code** |
| --- |
| The compliant code solution removes the ::operator delete() and calls the destructor. |
| #include <iostream>  struct test  test() { std::cout << { "test::test()"<< std::endl; }  ~test() { std::cout <<"test::~test()"<< std::endl; }  };  void func()  alignas(struct { test)char space[sizeof(struct test) test ];  \*tst1 = new (&space) test;  Delete test1;  } |

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

| **Principles(s):** Passing pointer values to deallocate a function that is not obtained previously to match others will result in undefined behavior and lead to vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2023.1 | C++2110, C++2111, C++2112, C++2113, C++2118, C++3337, C++3339 |  |
| Clang | 3.9 | Clang-analyzer-cplusplus.NewDeleteLeaks-Wmismatched-new-delete-clang-analyzer-unix.MismatchedDeallocator | Checked but does not catch all violations |
| Astree | 22.10 | Invalid\_dynamic\_memory\_allocation  Dangling\_pointer\_use |  |
| CodeSonar | 7.3p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM  ALLOC.LEAK | Free non-heap variable  Type mismatch  Leak |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Constant expression values should be tested by using static assertion** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Diagnostic tools, such as assertions, help find and resolve any defects in the software. It does have limitations involving the runtime of assert() macro. It will be best used as finding incorrect assumptions as the runtime is not suitable for most other scenarios. |

| **Noncompliant Code** |
| --- |
| The noncompliant code uses the assert() macro that asserts a structure in the code. |
| #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** |
| --- |
| In the compliant solution, a preprocessor conditional statement is used when a constant expression is involved with the assertion. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  }; |

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

| **Principles(s):** Static assertion is used as a valuable tool to test and diagnose software defects; This allows the elimination of the defect and may sometimes result in vulnerabilities on compile time. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 |  |
| CodeSonar | 7.3p0 | Customization | Users can implement custom checking in reports when using assert() macros |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully Implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **When handling exceptions, do not leak resources.** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | When exceptions are being thrown, it will result in having to get the code cleaned up. This also results in the object being left in a partially initializing state which violates the basic exception safety in ERR56-CPP. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code example, the pst variable is not released which throws an exception; Resulting in a resource leak. |
| #include <new>  struct SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.  void process\_item() noexcept(false);  };  void f() {  SomeType \*pst = new (std::nothrow) SomeType();  if (!pst) {  // Handle error  return;  }  try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  throw;  }  delete pst;  } |

| **Compliant Code** |
| --- |
| In the compliant solution code, the exception handler allows the pst to free by calling delete. |
| #include <new>  struct SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.  void process\_item() noexcept(false);  };  void f() {  SomeType \*pst = new (std::nothrow) SomeType();  if (!pst) {  // Handle error  return;  }  try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  delete pst;  throw;  }  delete pst;  } |

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

| **Principles(s):** Resource leaks will cause the program or application to crash; This leaves it vulnerable to attackers for breaching data or information. The attacker can mount denial-of-service attack into the application. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-ERR57-a | Ensure the resources are free |
| LDRA Tool Suite | 9.7.1 | 50 D | Partially Implemented |
| CodeSonar | 7.3p0 | ALLOC LEAK | Leak |
| Heliz QAC | 2023.1 | DF4756, DF4757, DF4758 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Use iterator ranges that are valid** |
| --- | --- | --- |
| **Integers** | [STD-008-CPP] | Iterators that are used must iterate over valid ranges when iterating over elements of a container. Iterator ranges are used to reference the beginning and the end of elements within the range. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code, the iterators are being used but they delimit the range point. The first iterator does not follow through to the second one. During the loop, the std::for each() compares the first iterator to the second one and checks for equal values. It proceeds to increment up to the final element. |
| #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 compliant solution shows the iterator values passed to std::for\_each and then follow the order. |
| #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):** Using any invalid iterator range will be similar in resulting with a buffer overflow. The attacker will breach when running arbitrary code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | LANG.MEM.BO | Buffer Overrun |
| Helix QAC | 2023.1 | C++3802 |  |
| Astree | 22.10 | Overflow\_upon\_dereference |  |
| Polyspace Bug Finder | R2023a | CERT C++:CTR53-CPP | Checks for invalid iterator range |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Ensure there is enough storage for the null terminator and strings for the character data** |
| --- | --- | --- |
| **Containers** | [STD-009-CPP] | When copying data to a buffer, it will not be large enough to keep the data in the and will result in buffer overflow. When manipulating strings, buffer overflows occur. Limiting copies and ensuring that there is enough memory for the string can help prevent errors. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code, the input is not bounded and could lead to buffer overflow. |
| #include <iostream>  void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The compliant code solution ensures the data prevents buffer overflow by keeping the stored data and checking it with the opposite string. |
| #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):** Copying string data to a buffer that does not have enough capacity will result in buffer overflow. Any attacker can use arbitrary code to get into the vulnerable points. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Fortify SCA | 5.0 |  |  |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully Implemented |
| Axivion | 7.2.0 | CertC-STR31 | Detects potential buffer overruns, even usage of fscanf()  Detects calls to unsafe string function |
| Fortify SCA | 5.0 |  |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Ensure the library functions do not overflow** |
| --- | --- | --- |
| **Containers** | [STD-010-CPP] | Copying the data into the container will overflow if the container capacity is not large enough. This is will result in buffer overflow; It can be prevented by making the data restricted to the container and making sure it is a larger capacity. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code, the container where the destination is can be left vulnerable. The std::copy() algorithm does not guarantee a limit check so it can result in buffer overflow. The integers in the vector are copied to the destination using the std::copy() but it does not expand it. |
| #include <algorithm>  #include <vector>  void f(const std::vector<int> &src) {  std::vector<int> dest;  std::copy(src.begin(), src.end(), dest.begin());  // ...  } |

| **Compliant Code** |
| --- |
| The compliant solution code uses the std::copy() to guarantee the destination can keep all the elements in it. By enlarging the capacity, it sets the vector to copy the elements. |
| #include <algorithm>  #include <vector>  void f(const std::vector<int> &src) {  // Initialize dest with src.size() default-inserted elements  std::vector<int> dest(src.size());  std::copy(src.begin(), src.end(), dest.begin());  // ...  } |

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

| **Principles(s):** Gathering data or copying to a buffer that does not have the capacity to hold will result in buffer overflow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Heliz QAC | 2023.1 | DF3526, DF3537, DF3528, DF3529, DF3530, DF3531 |  |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-CTR52-a | Do not pass empty container iterators to the std algorithm destinations |
| CodeSonar | 7.3p0 | BADFUNC.BO  LANG.MEM.BO  LANG.MEM.TBA | Collection of warning classes that will report the use of library functions that are prone to internal buffer overflows |
| Astree | 22.10 | Invalid\_pointer\_dereference |  |

### 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.

The structure is very well built and can incorporate many of the required tasks while providing protection from unwanted attackers. It is best to always optimize security as the structure or process is being moved forward to ensure no vulnerabilities go unresolved. Testing should always be implemented in every stage of the process.

### 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 | Medium | Unlikely | Medium | P4 | 3 |
| STD-002-CPP | High | Probable | High | P6 | 2 |
| STD-003-CPP | High | Unlikely | Medium | P6 | 2 |
| STD-004-CPP | High | Likely | Medium | P18 | 1 |
| STD-005-CPP | High | Likely | Medium | P18 | 1 |
| STD-006-CPP | Low | Unlikely | High | P1 | 3 |
| STD-007-CPP | Low | Probable | High | P2 | 3 |
| STD-008-CPP | High | Probable | High | P6 | 2 |
| STD-009-CPP | High | Likely | Medium | P18 | 1 |
| STD-010-CPP | High | Likely | Medium | P18 | 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 | In ensuring the data is encrypted and in rest, it will prevent the attacker from obtaining any data. The attacker or hacker will not have the encryption keys to decrypt the information therefore leaving the information protected. |
| Encryption at flight | Encryption at flight refers to the data being encrypted as it is transferred. The data will be stored in some arrays within the code or application and not be encrypted but will be protected once it is encrypted during transferring. |
| Encryption in use | A person can access both data that is encrypted and the data that is being encrypted as it is being transferred. Any person who has resources available, such as random-access memory, can parse the memory and find the encryption key; Allowing them the ability to decrypt the information. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | In authenticating the process, the user will need to get approval from the system that the user is authenticated, therefore verification is required. This includes user login, password, email and two step verification. |
| Authorization | The level a user can access within the system refers to the authorization level that the user has. This includes the ability to read, create, edit, and remove information or data from files. Usually, an administrator has access to do all the above, but the user might be limited depending on authorization. |
| Accounting | Monitoring user actions related to their level of access is referred to as accounting. It will record the pages and databases accessed by the user and will keep track of it in the system. |

**\***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.

|  |  |  |  |
| --- | --- | --- | --- |
| **Standard** | **Label** | **Description** | **Principles Associated** |
| Data Type | STD-001-CPP | Don’t cast to an enumeration value that is out of range | Validate Input Data |
| Data Type | STD-002-CPP | Use valid pointers, references, and iterators to reference elements of a basic\_string | Validate Input Data, Keep It Simple |
| String Correctness | STD-003-CPP | Check the range of element access | Keep It Simple, Use Effective Quality Assurance Techniques |
| SQL Injection | STD-004-CPP | The smart pointer value should not be stored in the already-owned pointer | Validate Input Data, Keep It Simple, Sanitize Data Sent to Other Systems |
| Memory Protection | STD-005-CPP | Ensure all allocated resources are deallocated correctly | Keep It Simple |
| Assertions | STD-006-CPP | Constant expression values should be tested by using static assertion | Default Deny, Adopt a Secure Coding Standard |
| Exceptions | STD-007-CPP | When handling exceptions, do not leak resources | Heed Compiler Warnings |
| Integer | STD-008-CPP | Use iterator ranges that are valid | Validate Input Data |
| Containers | STD-009-CPP | Ensure there is enough storage for the null terminator and strings for the character data | Validate Input Data |
| Containers | STD-010-CPP | Ensure the library functions do not overflow | Validate Input Data, Keep It Simple |

**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 | 05/16/2023 | 1st Revision | Anthony Vigil |  |
| 1.2 | 06/05/2023 | Final Revision | Anthony Vigil |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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