**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 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 12](#_Toc52464063)

[Coding Standard 6 14](#_Toc52464064)

[Coding Standard 7 16](#_Toc52464065)

[Coding Standard 8 18](#_Toc52464066)

[Coding Standard 9 20](#_Toc52464067)

[Coding Standard 10 22](#_Toc52464068)

[Defense-in-Depth Illustration 23](#_Toc52464069)

[Project One 23](#_Toc52464070)

[1. Revise the C/C++ Standards 23](#_Toc52464071)

[2. Risk Assessment 23](#_Toc52464072)

[3. Automated Detection 23](#_Toc52464073)

[4. Automation 23](#_Toc52464074)

[5. Summary of Risk Assessments 24](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 24](#_Toc52464076)

[7. Map the Principles 25](#_Toc52464077)

[Audit Controls and Management 26](#_Toc52464078)

[Enforcement 26](#_Toc52464079)

[Exceptions Process 26](#_Toc52464080)

[Distribution 27](#_Toc52464081)

[Policy Change Control 27](#_Toc52464082)

[Policy Version History 27](#_Toc52464083)

[Appendix A Lookups 27](#_Toc52464084)

[Approved C/C++ Language Acronyms 27](#_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 | Validate input from an untrusted data source. This helps to prevent numerous software vulnerabilities where the user inputs command line arguments or SQL code as the input, gaining access to areas of the program which are restricted. |
| 1. Heed Compiler Warnings | It’s important to use the highest warning levels available when compiling your code. This will help to find the most potential security vulnerabilities. |
| 1. Architect and Design for Security Policies | When designing software architecture, it’s important to implement proper security policies in each step. If one check fails, others may not. |
| 1. Keep It Simple | Keeping code simple with good comments helps to make it easier to understand. Simpler code can be understood well by others and checked for bugs more easily. |
| 1. Default Deny | Keep access decisions to deny access by default. Only allow when correct parameters are met. |
| 1. Adhere to the Principle of Least Privilege | Processes should have the least privilege needed to complete their task. If a task requires elevated privileges, the privileges should be granted on use and revoked on completion. If these elevated privileges are left on, that allows possible exploitation of them. |
| 1. Sanitize Data Sent to Other Systems | Use subsystems to clean the data on input from an unknown or untrusted source. This will help to prevent injection attacks. |
| 1. Practice Defense in Depth | Add redundancies for security within systems. When an outer layer fails, subsequent layers can stop it. |
| 1. Use Effective Quality Assurance Techniques | Use Quality Assurance Techniques to find hidden vulnerabilities within your code. Have internal and external reviews and testing to ensure it meets security requirements. |
| 1. Adopt a Secure Coding Standard | Follow the coding standards for your company and/or language. This will help to eliminate errors and confusion. |

### 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] | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| Checks whether a given value is within range of acceptable enumeration values. After casting the type it might not be able to represent the given 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** |
| --- |
| The compliant solution checks the value represented by the enumeration type before performing the conversion to guarantee the conversion doesn’t result in an unspecified value. In turn is restricts the converted value to one specific enumerator type. |
| enum EnumType {  First,  Second,  Third };  void f(int intVar) {  if (intVar < First || intVar > Third) {  // Handle error  }  EnumType enumVar = static\_cast<EnumType>(intVar); } |

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

| **Principles(s):** This could cause a risk for a buffer overflow attack. This is not likely to happen since enumerators are rarely used in practice as for indexing arrays. |
| --- |

**Threat Level**

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

Automation

|  |  |  |  |
| --- | --- | --- | --- |
| Tool | Version | Checker | Description |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| CodeSonar | 6.2p0 | Coercion Alters ValueCast Alters Value |  |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| Helix QAC | 2021.2 | C++3013 |  |
| PRQA QA-C++ | 4.4 | 3013 |  |
| PVS-Studio | 7.17 | V1016 |  |

#### 

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Guarantee that container indices and iterators are within the valid range |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows a function, insert\_in\_table(), that has two int parameters, pos and value, both of which can be influenced by data originating from untrusted sources. The function performs a range check to ensure that pos does not exceed the upper bound of the array, specified by tableSize, but fails to check the lower bound. Because pos is declared as a (signed) int, this parameter can assume a negative value, resulting in a write outside the bounds of the memory referenced by table. |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, int pos, int value) {    if (pos >= tableSize) {      // Handle error      return;    }    table[pos] = value;  } |

| **Compliant Code** |
| --- |
| Non-type templates can be used to define functions accepting an array type where the array bounds are deduced at compile time. This compliant solution is functionally equivalent to the previous bounds-checking one except that it additionally supports calling insert\_in\_table() with an array of known bounds. |
| #include <cstddef>  #include <new>    void insert\_in\_table(int \*table, std::size\_t tableSize, std::size\_t pos, int value) { // #1    if (pos >= tableSize) {      // Handle error      return;    }    table[pos] = value;  }    template <std::size\_t N>  void insert\_in\_table(int (&table)[N], std::size\_t pos, int value) { // #2    insert\_in\_table(table, N, pos, value);  }    void f() {    // Exposition only    int table1[100];    int \*table2 = new int[100];    insert\_in\_table(table1, 0, 0); // Calls #2    insert\_in\_table(table2, 0, 0); // Error, no matching function call    insert\_in\_table(table1, 100, 0, 0); // Calls #1    insert\_in\_table(table2, 100, 0, 0); // Calls #1    delete [] table2;  } |

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

| **Principles(s):** If a iterator variable exceeds range, this will lead to a program failure causing unexpected behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | overflow\_upon\_dereference |  |
| CodeSonar | 6.2p0 | LANG.MEM.BOLANG.MEM.BULANG.MEM.TOLANG.MEM.TULANG.MEM.TBALANG.STRUCT.PBB LANG.STRUCT.PPE | Buffer overrunBuffer underrunType overrunType underrunTainted buffer accessPointer before beginning of objectPointer past end of object |
| Helix QAC | 2021.2 | C++2891, C++3139, C++3140 |  |
| Klocwork | 2021.4 | ABV.ANY\_SIZE\_ARRAYABV.GENERALABV.STACKABV.TAINTEDSV.TAINTED.ALLOC\_SIZESV.TAINTED.CALL.INDEX\_ACCESSSV.TAINTED.CALL.LOOP\_BOUND SV.TAINTED.INDEX\_ACCESS |  |
| LDRA tool suite | 9.7.1 | 45 D, 47 S, 476 S, 489 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X | Partially implemented |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-CTR50-a | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2021b | CERT C++: CTR50-CPP | Checks for:Array access out of boundsArray access with tainted indexPointer dereference with tainted offsetRule partially covered. |
| PRQA QA-C++ | 4.4 | 2891, 3139, 3140 |  |
| PVS-Studio | 7.17 | V781 |  |

#### 

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist (or some other error occurs). |
| #include <cstdlib> #include <string> void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  } } |

| **Compliant Code** |
| --- |
| In this compliant solution, the results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib> #include <string> void f() {  const char \*tmpPtrVal = std::getenv("TMP");  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  if (!tmp.empty()) {  // ...  } } |

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

| **Principles(s):** When applying any literal values to a null pointer, this can cause unexpected behavior or possible attack by using it to execute arbitrary code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | assert\_failure |  |
| Helix QAC | 2021.2 | C++4770, C++4771, C++4772, C++4773, C++4774 |  |
| Klocwork | 2021.4 | NPD.CHECK.CALL.MIGHTNPD.CHECK.CALL.MUSTNPD.CHECK.MIGHTNPD.CHECK.MUSTNPD.CONST.CALLNPD.CONST.DEREFNPD.FUNC.CALL.MIGHTNPD.FUNC.CALL.MUSTNPD.FUNC.MIGHTNPD.FUNC.MUSTNPD.GEN.CALL.MIGHTNPD.GEN.CALL.MUSTNPD.GEN.MIGHTNPD.GEN.MUSTRNPD.CALLR NPD.DEREF |  |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

#### 

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Avoid using default operator new for over-aligned types |

| **Noncompliant Code** |
| --- |
| In the following noncompliant code example, the new expression is used to invoke the default operator new to obtain storage in which to then construct an object of the user-defined type Vector with alignment that exceeds the fundamental alignment of most implementations (typically 16 bytes). Objects of such over-aligned types are typically required by SIMD (single instruction, multiple data) vectorization instructions, which can trap when passed unsuitably aligned arguments. |
| struct alignas(32) Vector {    char elems[32];  };    Vector \*f() {    Vector \*pv = new Vector;    return pv;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, an overloaded operator new function is defined to obtain appropriately aligned storage by calling the C11 function aligned\_alloc(). Programs that make use of the array form of the new expression must define the corresponding member array operator new[] and operator delete[]. The aligned\_alloc() function is not part of the C++ 98, C++ 11, or C++ 14 standards but may be provided by implementations of such standards as an extension. Programs targeting C++ implementations that do not provide the C11 aligned\_alloc() function must define the member operator new to adjust the alignment of the storage obtained by the allocation function of their choice. |
| #include <cstdlib>  #include <new>    struct alignas(32) Vector {    char elems[32];    static void \*operator new(size\_t nbytes) {      if (void \*p = std::aligned\_alloc(alignof(Vector), nbytes)) {        return p;      }      throw std::bad\_alloc();    }    static void operator delete(void \*p) {      free(p);    }  };    Vector \*f() {    Vector \*pv = new Vector;    return pv;  } |

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

| **Principles(s):** when the new keyword is used, the system will automatically allocate the max amount the type can handle. It is not safe to use this object of a type with a stricter alignment requirement. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | C++3129 |  |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-MEM57-a | Avoid using the default operator 'new' for over-aligned types |
| Polyspace Bug Finder | R2021b | CERT C++: MEM57-CPP | Checks for situations where operator new is not overloaded for possibly overaligned types (rule fully covered) |

#### 

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the local variable space is passed as the expression to the placement new operator. The resulting pointer of that call is then passed to ::operator delete(), resulting in undefined behavior due to ::operator delete() attempting to free memory that was not returned by ::operator new(). |
| #include <iostream>  struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; } };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;  // ...  delete s1; } |

| **Compliant Code** |
| --- |
| This compliant solution removes the call to ::operator delete(), instead explicitly calling s1's destructor. This is one of the few times when explicitly invoking a destructor is warranted. |
| #include <iostream>  struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; } };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;  // ...  s1->~S(); } |

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

| **Principles(s):** Not deallocating dynamic memory could lead to memory fragments remaining stored, reducing performance, and causing risk for data to be stolen. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| Astrée | 20.10 | invalid\_dynamic\_memory\_allocationdangling\_pointer\_use |  |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM51 |  |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| CodeSonar | 6.2p0 | **ALLOC.FNH ALLOC.DF ALLOC.TM** | Free non-heap variableDouble freeType mismatch |
| Helix QAC | 2021.2 | C++2110, C++2111, C++2112, C++2113, C++2118, C++3337, C++3339, C++4262, C++4263, C++4264 |  |
| Klocwork | 2021.4 | **CL.FFM.ASSIGN CL.FFM.COPY CL.FMM**  **CL.SHALLOW.ASSIGN CL.SHALLOW.COPY**  **FMM.MIGHT FMM.MUST FNH.MIGHT FNH.MUST FUM.GEN.MIGHT FUM.GEN.MUST**  **UNINIT.CTOR.MIGHT UNINIT.CTOR.MUST UNINIT.HEAP.MIGHT UNINIT.HEAP.MUST** |  |
| LDRA tool suite | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D | Partially implemented |
| Parasoft C/C++test | 2021.2 | **CERT\_CPP-MEM51-a CERT\_CPP-MEM51-b CERT\_CPP-MEM51-c CERT\_CPP-MEM51-d** | Use the same form in corresponding calls to new/malloc and delete/freeAlways provide empty brackets ([]) for delete when deallocating arraysBoth copy constructor and copy assignment operator should be declared for classes with a nontrivial destructorProperly deallocate dynamically allocated resources |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2021b | CERT C++: MEM51-CPP | Checks for:Invalid Checks for:   * Invalid deletion of pointer * Invalid free of pointer * Deallocation of previously deallocated pointer   Rule partially covered. |
| PRQA QA-C++ | 4.4 | 2110, 2111, 2112, 2113, 2118,3337, 3339, 4262, 4263, 4264 |  |
| PVS-Studio | 7.17 | V515, V554, V611, V701, V748, V773, V1066 |  |
| SonarQube C/C++ Plugin | 4.10 | S1232 |  |

#### 

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | Incorporate diagnostic tests using assertions |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses the assert() macro to verify that memory allocation succeeded. Because memory availability depends on the overall state of the system and can become exhausted at any point during a process lifetime, a robust program must be prepared to gracefully handle and recover from its exhaustion. Consequently, using the assert() macro to verify that a memory allocation succeeded would be inappropriate because doing so might lead to an abrupt termination of the process, opening the possibility of a denial-of-service attack. See also MEM11-C. Do not assume infinite heap space and void MEM32-C. Detect and handle memory allocation errors. |
| char \*dupstring(const char \*c\_str) {    size\_t len;    char \*dup;      len = strlen(c\_str);    dup = (char \*)malloc(len + 1);    assert(NULL != dup);      memcpy(dup, c\_str, len + 1);    return dup;  } |

| **Compliant Code** |
| --- |
| This compliant solution demonstrates how to detect and handle possible memory exhaustion: |
| char \*dupstring(const char \*c\_str) {    size\_t len;    char \*dup;      len = strlen(c\_str);    dup = (char\*)malloc(len + 1);    /\* Detect and handle memory allocation error \*/    if (NULL == dup) {        return NULL;    }      memcpy(dup, c\_str, len + 1);    return dup;  } |

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

| **Principles(s):** Using assertions for data strengthens code from security flaws and assists in finding defects |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **CodeSonar** | **6.2p0** | **LANG.FUNCS.ASSERTS** | **Not enough assertions** |
| **Coverity** | **2017.07** | **ASSERT\_SIDE\_EFFECT** | **Can detect the specific instance where assertion contains an operation/function call that may have a side effect** |
| **Parasoft C/C++test** | **2021.2** | **CERT\_C-MSC11-a** | **Assert liberally to document internal assumptions and invariants** |

#### 

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    try {      f();    } catch (...) {      // Handle error    }  } |

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

| **Principles(s):** Unhandled exceptions could lead to undefined behavior, program errors, or risk exploit by an attacker. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | main-function-catch-allearly-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 |  |
| Helix QAC | 2021.2 | C++4035, C++4036, C++4037 |  |
| Klocwork | 2021.4 | MISRA.CATCH.ALL |  |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-ERR51-aCERT\_CPP-ERR51-b | Always catch exceptionsEach exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| Polyspace Bug Finder | R2021b | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| PRQA QA-C++ | 4.4 | 4035, 4036, 4037 |  |
| RuleChecker | 20.10 | main-function-catch-allearly-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Overflow | [STD-008-CPP] | Use valid iterator ranges |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the two iterators that delimit the range point into the same container, but the first iterator does not precede the second. On each iteration of its internal loop, std::for\_each() compares the first iterator (after incrementing it) with the second for equality; as long as they are not equal, it will continue to increment the first iterator. Incrementing the iterator representing the past-the-end element of the range results in undefined 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** |
| --- |
| In this compliant solution, the iterator values passed to std::for\_each() are passed in the proper 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 invalid references, pointers, or iterators to reference elements of a container results in undefined behavior and risk . |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-FIO50 |  |
| CodeSonar | 6.2p0 | IO.IOWOPIO.OIWOP | Input After Output Without PositioningOutput After Input Without Positioning |
| Helix QAC | 2021.2 | C++4711, C++4712, C++4713 |  |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-FIO50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| Polyspace Bug Finder | R2021b | CERT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |

Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory | [STD-009-CPP] | Detect and handle memory allocation errors |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an array of int is created using ::operator new[](std::size\_t) and the results of the allocation are not checked. The function is marked as noexcept, so the caller assumes this function does not throw any exceptions. Because ::operator new[](std::size\_t) can throw an exception if the allocation fails, it could lead to abnormal termination of the program. |
| #include <cstring>    void f(const int \*array, std::size\_t size) noexcept {    int \*copy = new int[size];    std::memcpy(copy, array, size \* sizeof(\*copy));    // ...    delete [] copy;  } |

| **Compliant Code** |
| --- |
| When using std::nothrow, the new operator returns either a null pointer or a pointer to the allocated space. Always test the returned pointer to ensure it is not nullptr before referencing the pointer. This compliant solution handles the error condition appropriately when the returned pointer is nullptr. |
| #include <cstring>  #include <new>    void f(const int \*array, std::size\_t size) noexcept {    int \*copy = new (std::nothrow) int[size];    if (!copy) {      // Handle error      return;    }    std::memcpy(copy, array, size \* sizeof(\*copy));    // ...    delete [] copy;  } |

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

| **Principles(s):** It’s important to properly manage your memory and handle errors due to the risk of an attacker exploiting it with a buffer overflow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE |  |  |  |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| Helix QAC | 2021.2 | C++3225, C++3226, C++3227, C++3228, C++3229, C++4632 |  |
| Klocwork | 2021.4 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF |  |
| LDRA tool suite | 9.7.1 | 45 D | Partially implemented |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-MEM52-aCERT\_CPP-MEM52-b | Check the return value of newDo not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2021b | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |
| PRQA QA-C++ | 4.4 | 3225, 3226, 3227, 3228, 3229, 4632 |  |
| PVS-Studio | 7.17 | V522, V668 |  |

#### 

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data Type | [STD-010-CPP] | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist (or some other error occurs). |
| #include <cstdlib>  #include <string>    void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>    void f() {    const char \*tmpPtrVal = std::getenv("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");    if (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):** Dereferencing a Null pointer is accessing a data that is currently only being referred to by a pointer. This can lead to data issues or possible attack by using it to execute arbitrary code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | assert\_failure |  |
| Helix QAC | 2021.2 | C++4770, C++4771, C++4772, C++4773, C++4774 |  |
| Klocwork | 2021.4 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF |  |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

### 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 | Unlikely | Medium | High | 2 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | Medium | Unlikely | Low | P6 | L2 |
| STD-005-CPP | **High** | **Likely** | **Medium** | **P18** | L1 |
| STD-006-CLG | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | **Low** | **Probable** | **Medium** | **P4** | **L3** |
| STD-008-CPP | Low | Likely | Medium | P6 | L2 |
| STD-009-CPP | High | Likely | Medium | P18 | L1 |
| STD-010-CPP | High | Likely | Medium | P18 | L1 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at Rest is to keep the attacker from accessing unencrypted data. This is done by keeping data encrypted when it is not in use. Then even if the attacker gains access, they must first decrypt the files. |
| Encryption at flight | Encryption in Flight is the practice of encrypting data while it’s being transmitted. So if the data is intercepted, it must be decrypted first. This data is then decrypted by the proper receiver with their keys. |
| Encryption in use | Encrypted in use is data that remains encrypted while it is being used so that the keys never become available to be stolen. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the allowance of access to files that are normally restricted. This is normally granted with a username and password |
| Authorization | Authorization is the use of allowing access to a higher level of files to already Authenticated users. |
| Accounting | Accounting is the process of monitoring users and their activity on their Authorization levels as well as removing or DeAuthorizing users who lose access, so it does not cause a security breach. |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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