**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 input data is checking to ensure that the user-provided data is of the expected type, and it meets any of our constraints prior to processing that data. If the data is of the incorrect type or isn’t fitting within the constraints, we provided the user through the input validation step we are able to catch that and help prevent some vulnerabilities. |
| 1. Heed Compiler Warnings | Compilers can detect some common mistakes like not returning a value from a function. The code provided may be legal, but heeding these warnings help to reduce bugs and unexpected outcomes in your code. It can also help to remove some entry ways for bad actors to exploit your code. |
| 1. Architect and Design for Security Policies | While designing your code it is important that you keep security in mind and meet the standards for what the code should accomplish. Through architecting and designing for security policies we can save time while also delivering secure code. |
| 1. Keep It Simple | When developing the code, we should keep it clean and simple. It ensures that maintenance of the code is easier because those after you can have a clear understanding of what you are attempting to accomplish and fix any problems that arise. It also makes it easier to update and protect the end user. |
| 1. Default Deny | When we are doing any type of access it is a good idea to default to denial. This helps prevent anyone from gaining access to something that was not intended for them. While doing this we can then go forward and grant access to the correct parties when necessary. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege goes hand and hand with default denial. It is a standard where we are giving the user only the access that is necessary. For most users it will be defaulting to no access for others we can provide them with only what level of higher access that they need. This helps ensure that users and developers only have access to what is necessary for there task. |
| 1. Sanitize Data Sent to Other Systems | Whenever we are sending data to any other system it is important you sanitize it for any possible vulnerabilities or sensitive information. When sending code without sanitization we can allow for things like man in the middle attacks or other exploits to gain data or information that wasn’t needed to be sent. |
| 1. Practice Defense in Depth | Most systems need more than just one layer of protection and when designing a system, it is important to combine multiple layers. This can be for things like compliance to government standards and/or to help safeguard the system from attacks. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance is a large part of every process and when you are designing any type of product it is crucial to the result. The quality assurance technique should be in depth to ensure that issues and vulnerabilities are discovered prior to a customer or malicious actor viding and using them. |
| 1. Adopt a Secure Coding Standard | Having a secure coding standard is an important principle with anything that we are designing. No matter what language or platform we are creating for ensure you are implementing a secure coding standard approach. |

### 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** |
| --- |
| The noncompliant code attempts to check if the value is within the range of acceptable enumeration, but it does so after casting the enumeration type which may not be able to represent the given integer value. This could result into an unspecified value and unspecified behavior if the value falls out of that range. |
| 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 complaint code checks that the value can be represented by the enumeration type prior to performing a conversion. This guarantees that the conversion doesn’t result in an unspecified value. It accomplishes this through restricting the converted value to one of which there is a specific enumerator value. |
| 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 code would map to keeping it simple. In order to avoid errors with our calculations we should ensure that we are assigning integers in a straightforward manner. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | cast-integar-to-enum | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| CodeSonar | 7.4p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2023.3 | C++3013 |  |
| Parasoft  C/C++test | 2023.1 | CERT\_CPP\_INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| PVS-Studio | 7.26 | V1016 |  |
| RuleChecker | 22.10 | cast-integer-to-enum | Partially checked |
| Polyspace  Bug Finder | R2023b | CERT C++: INT50-CPP | Checks for casting to out-of-range enumeration value (rule fully covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Gracefully handle self-copy assignment |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the copy assignment operator doesn’t protect against self-copy assignment. If self-copy occurs this ->s1 is deleted and this results in rhs.s1 being deleted. The invalidated memory for rhs.s1 is passed into the copy constructor for s and can result in dereferencing an invalid pointer. |
| struct S { S(const S &) noexcept; /\* ... \*/ };    class T {  int n;  S \*s1;    public:  T(const T &rhs) : n(rhs.n), s1(rhs.s1 ? new S(\*rhs.s1) : nullptr) {}  ~T() { delete s1; }    // ...    T& operator=(const T &rhs) {  n = rhs.n;  delete s1;  s1 = new S(\*rhs.s1);  return \*this;  }  }; |

| **Compliant Code** |
| --- |
| The complaint code solution guards against self-copy assignment by testing if the given parameter is the same as this. If the self-copy assignment occurs the operator= does nothing. |
| #include <new>    struct S { S(const S &) noexcept; /\* ... \*/ };    class T {  int n;  S \*s1;    public:  T(const T &rhs) : n(rhs.n), s1(rhs.s1 ? new S(\*rhs.s1) : nullptr) {}  ~T() { delete s1; }    // ...    T& operator=(const T &rhs) {  if (this != &rhs) {  n = rhs.n;  delete s1;  try {  s1 = new S(\*rhs.s1);  } catch (std::bad\_alloc &) {  s1 = nullptr; // For basic exception guarantees  throw;  }  }  return \*this;  }  } |

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

| **Principles(s):** This would map to using effective quality assurance techniques. We want to ensure that we are employing effective tests to prevent errors or vulnerabilities being exposed to the user. In this example we are implementing a test to verify if the variable is the same prior to copying it. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | dangling\_pointer\_use |  |
| Clang | 9.0 (r361550) | IO.DC  ALLOC.DF  ALLOC.LEAK  LANG.MEM.NPD  LANG.STRUCT.RC  IO.UAC  ALLOC.UAF | Double Close  Double Free  Leak  Null Pointer Dereference  Redundant Condition  Use After Close  Use After Free |
| Helix QAC | 2023.3 | C++4072, C++4073,  C++4075, C++4076 |  |
| Klocwork | 2023.3 | CL.SELF-ASSIGN |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-OOP54-a | Check for assignment to self in operator= |
| Polyspace Bug Finder | R2023b | CERT C++: OOP54-CPP | Checks for copy assignment operators where self-assignment is not tested (rule partially covered) |

#### 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** |
| --- |
| This noncompliant code attempts to create a string object from the results of a call to std::getenv(). However, since std::getenv() returns a null pointer on failure the code can lead to an undefined behavior whenever the environment variable doesn’t exit. |
| #include <cstdlib>  #include <string>    void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| In the compliant solution the results of the call to std::getenv() are check to see if it is null prior to creating the std::string object. |
| #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):** This principal maps to validating input data. By ensuring that the input data isn’t a null pointer prior to assigning it we are able to prevent this issue. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | assert\_failure |  |
| CodeSonar | 7.4p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Helix QAC | 2023.3 | DF4770, DF4771, DF4772,  DF4773, DF4774 |  |
| Klocwork | 2023.3 | 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 | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2023b | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004 -CPP | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| Since the input is unbounded the following code can lead to a buffer overflow. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| This solution is using std::string to store the data since the data is not truncated and helps guard against buffer overflow. |
| #include <iostream>  #include <string>    void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):** This would map to validate input data. Since we are allowing the user to send unbounded input data we want to ensure that we are mapping it in a proper way and the compliant code does so. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | stream-input-char-array | Partially checked + soundly supported |
| CodeSonar | 7.4p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Helix QAC | 2023.3 | C++5216  DF2835, DF2836, DF2839, |  |
| Klocwork | 2023.3 | NNTS.MIGHT  NNTS.TAINTED  NNTS.MUST  SV.UNBOUND\_STRING\_INPUT.CIN |  |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| Parasoft  C/C++test | 2023.1 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the 'char' buffer to store input from 'std::cin' |
| Polyspace  Bug Finder | R2023b | CERT C++: STR50-CPP | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation * Insufficient destination buffer size   Rule partially covered. |
| RuleChecker | 22.10 | stream-input-char-array | Partially checked |
| SonarQube C/C++ Plugin | 4.10 | S3519 |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005 -CPP | Avoid information leakage when passing a class object across a trust boundary |

| **Noncompliant Code** |
| --- |
| The following noncompliant code runs in kernel space, and it copies data from arg to user space. However, the object may be using padding bits, and these padding bits can contain sensitive information that may be leaked when the data is copied to the 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** |
| --- |
| The compliant code is explicitly declaring padding bits within the structure. However, this solution is dependent on the target memory architecture and thus would not be portable. |
| #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):** This maps to sanitizing data that is sent to other systems. When we are sending the data to user space we want to ensure we are mapping it correctly so we aren’t leaking data that was in the padding bits in this example. |
| --- |

**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 | 7.4p0 | MISC.PADDING.POTB | Padding Passed Across a Trust  Boundary |
| Helix QAC | 2023.3 | **DF4941, DF4942, DF4943** |  |
| Parasoft C/C++test | 2023.1 | 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 | R2023b | CERT C++: DCL55-CPP | Checks for information leakage due to structure padding (rule partially covered) |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-C | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| The noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential in order for this code to behave correctly |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| If you are using constant expressions for assertions you can use a preprocessor conditional statement as in this complaint solution. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** This is a great example of using effective quality assurance techniques. We want to ensure that our assertations are being used properly. |
| --- |

**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-DCL03** |  |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 7.4p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| Compass/ROSE |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| ECLAIR | 1.2 | **CC2.DCL03** | Fully implemented |
| LDRA tool suite | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| In the noncompliant code neither f() or main () catch the exceptions that are thrown by throwing\_function(). Since no matching handler can be found for the exception that is thrown std::terminate() is called. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In the complaint code the main entry point handles all the exceptions. This 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):** This would map to adopting a secure coding standard. Instead of just mapping one error the compliant code ensures that the stack is properly unwound in other errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **main-function-catch-all**  **early-catch-all** | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-ERR51** |  |
| CodeSonar | 7.4p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Helix QAC | 2023.3 | **C++4035, C++4036, C++4037** |  |
| Klocwork | 2023.3 | **MISRA.CATCH.ALL** |  |
| LDRA tool suite | 9.7.1 | **527 S** | Partially implemented |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each 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 | R2023b | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| RuleChecker | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory Management | STD-008 -CPP | Honor replacement dynamic storage management requirements |

| **Noncompliant Code** |
| --- |
| In the following noncompliant code example the global operator new(std::size\_t) function is replaced by a custom implementation. However, this custom implementation doesn’t honor the behavior required by the function it replaces. This custom allocator returns a null pointer instead of throwing an exception of type std::bad\_alloc. Since it is returning a null pointer instead of throwing functions may attempt to dereference a null pointer. |
| #include <new>    void \*operator new(std::size\_t size) {  extern void \*alloc\_mem(std::size\_t); // Implemented elsewhere; may return nullptr  return alloc\_mem(size);  }    void operator delete(void \*ptr) noexcept; // Defined elsewhere  void operator delete(void \*ptr, std::size\_t) noexcept; // Defined elsewhere |

| **Compliant Code** |
| --- |
| This compliant code implements the required behavior for the replaced global allocator function by throwing a std::bad\_alloc |
| #include <new>    void \*operator new(std::size\_t size) {  extern void \*alloc\_mem(std::size\_t); // Implemented elsewhere; may return nullptr  if (void \*ret = alloc\_mem(size)) {  return ret;  }  throw std::bad\_alloc();  }    void operator delete(void \*ptr) noexcept; // Defined elsewhere  void operator delete(void \*ptr, std::size\_t) noexcept; // Defined elsewhere |

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

| **Principles(s):** This would map to sanitize data sent to other systems. When a function can send a nullptr or another error to a different part of the code we need to ensure it is handle properly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2023.3 | **DF4736, DF4737, DF4738, DF4739** |  |
| Klocwork | 2023.3 | **CERT.MEM.OVERRIDE.DELETE** **CERT.MEM.OVERRIDE.NEW** |  |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-MEM55-a** | The user defined 'new' operator should throw the 'std::bad\_alloc' exception when the allocation fails |
| Polyspace Bug Finder | R2023b | CERT C++: MEM55-CPP | Checks for replacement allocation/deallocation functions that do not meet requirements of the Standard (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming (OOP) | STD-009 -CPP | Copy operations must not mutate the source object |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the copy operations for A mutate the source operand by resetting its member variable m to 0. When std::fill() is called the first element copied will have the original value of obj.m, 12, which at this point obj.m is set to 0. The other copies will all retain the value of 0. |
| #include <algorithm>  #include <vector>    class A {  mutable int m;    public:  A() : m(0) {}  explicit A(int m) : m(m) {}    A(const A &other) : m(other.m) {  other.m = 0;  }    A& operator=(const A &other) {  if (&other != this) {  m = other.m;  other.m = 0;  }  return \*this;  }    int get\_m() const { return m; }  };    void f() {  std::vector<A> v{10};  A obj(12);  std::fill(v.begin(), v.end(), obj);  } |

| **Compliant Code** |
| --- |
| In this complaint code solution, the copy operations for A no longer mutate the source operand. The ensures that the vector contains the equivalent copies of obj. Instead, A has been given move operations that perform the mutation when it is safe for it to do so. |
| #include <algorithm>  #include <vector>    class A {  int m;    public:  A() : m(0) {}  explicit A(int m) : m(m) {}    A(const A &other) : m(other.m) {}  A(A &&other) : m(other.m) { other.m = 0; }    A& operator=(const A &other) {  if (&other != this) {  m = other.m;  }  return \*this;  }    A& operator=(A &&other) {  m = other.m;  other.m = 0;  return \*this;  }    int get\_m() const { return m; }  };    void f() {  std::vector<A> v{10};  A obj(12);  std::fill(v.begin(), v.end(), obj);  } |

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

| **Principles(s):** This would map to use effective quality assurance techniques. By testing the code we would have found that it was mutating the source when performing the copy operands and work to resolve it. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | **LANG.FUNCS.COPINC** | Copy Operation Parameter Is Not const |
| Helix QAC | 2023.3 | **C++4075** |  |
| Klocwork | 2023.3 | **CERT.OOP.COPY\_MUTATES** |  |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-OOP58-a** | Copy operations must not mutate the source object |
| Polyspace Bug Finder | R2023b | CERT C++: OOP58-CPP | Checks for copy operation modifying source operand (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | STD-010 -CPP | Do not depend on the order of evaluation for side effects |

| **Noncompliant Code** |
| --- |
| In the noncompliant code example, I is evaluated more than once in an unsequenced manner. The behavior will be undefined for this expression. |
| void f(int i, const int \*b) {  int a = i + b[++i];  // ...  } |

| **Compliant Code** |
| --- |
| In the complaint code the odder of evaluation of the operands can only be interpreted in one way. |
| void f(int i, const int \*b) {  ++i;  int a = i + b[i];  // ...  } |

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

| **Principles(s):** This would map to keeping it simple. By ensuring we define how I is iterated upon in the code in a clear manner we can resolve the issue in the non-compliant code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| EXP50-CPP | Medium | Probable | Medium | **P8** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-EXP50** |  |
| Clang | 3.9 | -Wunsequenced | Can detect simple violations of this rule where path-sensitive analysis is not required |
| CodeSonar | 7.4p0 | **LANG.STRUCT.SE.DEC** **LANG.STRUCT.SE.INC** | Side Effects in Expression with Decrement Side Effects in Expression with Increment |
| Compass/ROSE |  |  | Can detect simple violations of this rule. It needs to examine each expression and make sure that no variable is modified twice in the expression. It also must check that no variable is modified once, then read elsewhere, with the single exception that a variable may appear on both the left and right of an assignment operator |
| Coverity | v7.5.0 | **EVALUATION\_ORDER** | Can detect the specific instance where a statement contains multiple side effects on the same value with an undefined evaluation order because, with different compiler flags or different compilers or platforms, the statement may behave differently |
| ECLAIR | 1.2 | **CC2.EXP30** | Fully implemented |
| GCC | 4.9 |  | Can detect violations of this rule when the -Wsequence-point flag is used |
| Helix QAC | 2023.3 | **C++3220, C++3221, C++3222, C++3223, C++3228** |  |
| Klocwork | 2023.3 | **PORTING.VAR.EFFECTS** **CERT.EXPR.PARENS** **MISRA.EXPR.PARENS.INSUFFICIENT** **MISRA.INCR\_DECR.OTHER** |  |
| LDRA tool suite | 9.7.1 | **35 D, 1 Q, 9 S, 134 S, 67 D, 72 D** | Partially implemented |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-EXP50-a** **CERT\_CPP-EXP50-b** **CERT\_CPP-EXP50-c** **CERT\_CPP-EXP50-d** **CERT\_CPP-EXP50-e** **CERT\_CPP-EXP50-f** | The value of an expression shall be the same under any order of evaluation that the standard permits Don't write code that depends on the order of evaluation of function arguments Don't write code that depends on the order of evaluation of function designator and function arguments Don't write code that depends on the order of evaluation of expression that involves a function call Between sequence points an object shall have its stored value modified at most once by the evaluation of an expression Don't write code that depends on the order of evaluation of function calls |
| Polyspace Bug Finder | R2023b | CERT C++: EXP50-CPP | Checks for situations where expression value depends on order of evaluation (rule fully covered). |
| PVS-Studio | 7.26 | **V521**, **V708** |  |
| SonarQube C/C++ Plugin | 4.10 | **IncAndDecMixedWithOtherOperators** | Partially implemented |
| Splint | 5.0 |  |  |

### 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.

To streamline the enforcement of our standards, the establishment of a centralized policy repository is important. This repository is an integral component of our assessment and planning procedures, serving as the designated storage for our organizational policies. This will include adherence to regulations, release requirements, and organizational guidelines. This repository will also play a large part in allowing us to set up and implement policy automation and enforcement.

Automated tools will be employed to assess risks, and issue alerts, and notifications. The automation of compliance not only enhances operational efficiency but also mitigates the need for repetitive tasks. We will also be able to automate the transition and health check phases. Using automated penetration testing aids in minimizing false alarms and streamlining the resolution of identified issues.

We can extend this automation in many ways like through establishing a system for logging and storing data in a dedicated database. This database serves as a critical tool for identifying vulnerabilities and preventing potential attacks. Various security measures, such as signature checks, ensuring data integrity, and employing multiple security layers, will be implemented. We can also do things like automate the creation of save points. These backups will serve as secure backups allowing us to swiftly revert to a previous stable version in the event of an attack or system malfunction.

### 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 | L3 |
| STD-002-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-003-CPP | Low | Probable | High | P2 | L3 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | Low | Unlikely | High | P1 | L3 |
| STD-006-C | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-009-CPP | Low | Likely | Low | **P9** | **L2** |
| STD-010-CPP | EXP50-CPP | Medium | Probable | Medium | **P8** |

### 

### 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 | While data is being stored the encryption at rest policy would want to ensure that it is encrypted. This can be applied by encrypting data in our databases or on the user device with whatever encryption algorithm is best for that occasion from things like SHA and other encryption algorithms. The reason we should implement encryption at rest is because it helps protect the data in the case that a bad actor can get it. If someone can hack our database or mine the data from the user’s device having a second layer of protection from encryption would help protect that data and not allow for someone to use it. |
| Encryption at flight | When we are transmitting data from point A to point B we can send the data in a plaintext format, but that isn’t secure. Instead, the data should be encrypted while it is being sent. In a lot of applications, we will be sending data over the network or to servers and to protect it the data should be encrypted. This will help safeguard the data from unauthorized individuals intercepting it and using it. Applying this practice allows us to mitigate the damage from things like man-in-the-middle attacks that are trying to intercept data we are sending. |
| Encryption in use | Through things like Secure enclaves, we can perform operations on data while it is encrypted. Encryption, while it is in use, would be to have the data secure while we are using it through different approaches like the use of secure enclaves. The reason we should use this is because some attack methods like buffer overflow or exploits allowing us to read some data from RAM if that data isn’t encrypted then bad actors will be able to use it and gather more. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is a more common policy, and it is ensuring that the person that is intending to log into your system for example has the proper login credentials to access the system. Using authentication, we can ensure that the user of the system is supposed to have access. Authentication could go beyond just login credentials like using 2FA for the user, biometrics, NFC cards, and much more. This helps ensure that only allowed users can access the system protecting it from unauthorized access. |
| Authorization | Authorization is the process of limiting what actions certain users can perform. The way this can be implemented is through adding distinct roles and allowing those roles to have specific functions. Then we can assign those roles to new users. This can also tie into the principle of least privilege giving the user only the privileges that are necessary for the role they are doing. This helps protect us from disgruntled ex-employees who may want to attack the system prior to leaving, or in the case of a standard end-user account being hacked them not having unnecessary privileges to the database that is now given to some bad actor. |
| Accounting | Accounting is keeping a log of user actions and interactions in a system. This can be used by making a record of every change that a user in the system does with rolling backups of a set time frame. The reason why it is so that in the case that a user makes a mistake, or someone makes an unauthorized change admin will be able to see when it occurred and roll back the changes to a previous state. It also allows for the discovery of a data breach in the case that the admin sees a user download a lot of data and the user didn’t authorize it. It could lead them to discover that the user's account was compromised and spot the data breach to help remedy the issue and notify end users. |

**\***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 |  |
| 1.5 | 11/12/2023 | Updated coding standards and core principles | Malcolm McGee |  |
| 2.0 | 12/03/2023 | Created polices for for Encryption and Triple A. Completed automation, threat level, and risk and other sections to complete form. | Malcolm McGee |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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