**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 all inputs from untrusted data sources. External data sources can be a cause for concern. This can be in command line arguments, network interfaces and environment variables. Validating input data can prevent many major vulnerabilities. |
| 1. Heed Compiler Warnings | When compiling the code, always use the highest warning levels. Eliminate warnings by modifying the code as well. Use a static and dynamic analysis tool to help detect and eliminate any other security flaws that may exist. |
| 1. Architect and Design for Security Policies | Implement and enforce the proper security policies when designing the software architecture. |
| 1. Keep It Simple | Keep the coding design simple and small. This will help avoid having to adjust to more complex systems. These more complex systems can be more susceptible to errors and failed security mechanisms. |
| 1. Default Deny | Rather than exclusion, standardize the access decisions based on permission. The default access should deny, however with specific conditions access should be permitted. |
| 1. Adhere to the Principle of Least Privilege | To complete the job, the least amount of privileges should be used for process executions. If there are elevated permission, then they should only be allotted the time to complete the task. This will help with reducing the chance of an attacker to change code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize the data passed through complex subsystems. This could help prevent attackers from using an injection attack in order to manipulate any components of the subsystems. |
| 1. Practice Defense in Depth | Always practice defense in depth with multiple layers of security. When one layer may fail another will be there to help prevent a security flaw from becoming exploitable. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques can help the chances of finding and eliminating vulnerabilities. Through using multiple testing phases and reviews of independent security and external security, can all lead to a more secure system. |
| 1. Adopt a Secure Coding Standard | It is always important and a must to have a secure coding standard for development in the platform and 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** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not cast to an out-of-range enumeration value. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);    if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| This compliant solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does this by restricting the converted value to one for 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):** It is possible for unspecified values to result in a buffer overflow, leading to the execution of arbitrary code by an attacker. However, because enumerators are rarely used for indexing into arrays or other forms of pointer arithmetic, it is more likely that this scenario will result in data integrity violations rather than arbitrary code execution. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | cast-integer-to-enum | Partially Checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| CodeSonar | 7.2p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2022.4 | C++3013 |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| PRQA QA-C++ | 4.4 | 3013 |  |
| PVS-Studio | 7.23 | V1016 |  |
| RuleChecker | 22.10 | cast-integer-to-enum | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Use valid pointers, references, and iterators to reference elements in a container. |

| **Noncompliant Code** |
| --- |
| This example is noncompliant because the iterator loc is invalidated after the first call to insert(). The behavior of subsequent calls to insert() is undefined. |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  d.insert(pos, items[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, pos is assigned a valid iterator on each insertion, preventing undefined behavior. |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  pos = d.insert(pos, items[i] + 41.0);  }  } |

**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. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | overflow\_upon\_dereference |  |
| CodeSonar | 7.2p0 | ALLOC.UAF | Use After Free |
| Helix QAC | 2022.4 | DF4746, DF4747, DF4748, DF4749 |  |
| Klocwork | 2022.4 | ITER.CONTAINER.MODIFIED |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-CTR51-a | Do not modify container while iterating over it |
| Polyspace Bug Finder | R2022b | CERT C++: CTR51-CPP | Checks for use of invalid iterator (rule partially covered). |
| PVS-Studio | 7.23 | V783 |  |

#### 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):** Dereferencing a null pointer is undefined behavior, typically abnormal program termination. In some situations, however, dereferencing a null pointer can lead to the execution of arbitrary code [Jack 2007, van Sprundel 2006]. The indicated severity is for this more severe case; on platforms where it is not possible to exploit a null pointer dereference to execute arbitrary code, the actual severity is low. |
| --- |

**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 | 7.2p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Helix QAC | 2022.4 | DF4770, DF4771, DF4772, DF4773, DF4774 |  |
| Klocwork | 2022.4 | 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 | 2022.2 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2022b | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule fully covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Honor replacement dynamic storage management requirements |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the global operator new(std::size\_t) function is replaced by a custom implementation. However, the custom implementation fails to honor the behavior required by the function it replaces, as per the C++ Standard. |
| #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 solution implements the required behavior for the replaced global allocator function by properly throwing a std::bad\_alloc exception when the allocation fails. |
| #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):** Failing to meet the stated requirements for a replaceable dynamic storage function leads to undefined behavior. The severity of risk depends heavily on the caller of the allocation functions, but in some situations, dereferencing a null pointer can lead to the execution of arbitrary code [Jack 2007, van Sprundel 2006]. The indicated severity is for this more severe case. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2022.4 | DF4736, DF4737, DF4738, DF4739 |  |
| Klocwork | 2022.4 | CERT.MEM.OVERRIDE.DELETE  CERT.MEM.OVERRIDE.NEW |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-MEM55-a | The user defined 'new' operator should throw the 'std::bad\_alloc' exception when the allocation fails |
| Polyspace Bug Finder | R2022b | CERT C++: MEM55-CPP | Checks for replacement allocation/deallocation functions that do not meet requirements of the Standard (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):** Passing a pointer value to a deallocation function that was not previously obtained by the matching allocation function results in undefined behavior, which can lead to exploitable vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | invalid\_dynamic\_memory\_allocation  dangling\_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 | 7.2p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM  ALLOC.LEAK | Free non-heap variable  Double free  Type mismatch  Leak |
| Helix QAC | 2022.4 | C++2110, C++2111, C++2112, C++2113, C++2118, C++3337, C++3339, C++4262, C++4263, C++4264 |  |
| Klocwork | 2022.4 | 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 | 2022.2 | 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/free  Always provide empty brackets ([]) for delete when deallocating arrays  Both copy constructor and copy assignment operator should be declared for classes with a nontrivial destructor  Properly deallocate dynamically allocated resources |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2022b | CERT C++: MEM51-CPP | 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 | 7.23 | 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-CPP] | Use a static assertion to test the value of a constant expression |

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

| **Compliant Code** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant 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):** Static assertion is a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities at compile time. The absence of static assertions, however, does not mean that code is incorrect. |
| --- |

**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 | 7.2p0 | (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] | Honor exception specifications |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a function is declared as nonthrowing, but it is possible for std::vector::resize() to throw an exception when the requested memory cannot be allocated. |
| #include <cstddef>  #include <vector>    void f(std::vector<int> &v, size\_t s) noexcept(true) {  v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the function's noexcept-specification is removed, signifying that the function allows all exceptions. |
| #include <cstddef>  #include <vector>    void f(std::vector<int> &v, size\_t s) {  v.resize(s); // May throw, but that is okay  } |

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

| **Principles(s):** Throwing unexpected exceptions disrupts control flow and can cause premature termination and denial of service. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | unhandled-throw-noexcept | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR55 |  |
| CodeSonar | 7.2p0 | LANG.STRUCT.EXCP.THROW | Use of throw |
| Helix QAC | 2022.4 | C++4035, C++4036, C++4632 |  |
| LDRA tool suite | 9.7.1 | 56 D | Partially implemented |
| Parasoft C/C++Test | 2022.2 | CERT\_CPP-ERR55-a | Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) |
| Polyspace Bug Finder | R2022b | CERT C++: ERR55-CPP | Checks for noexcept functions exiting with exception (rule fully covered) |
| PRQA QA-C++ | 4.4 | 4035, 4036, 4632 |  |
| RuleChecker | 22.10 | unhandled-throw-noexcept | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Expressions** | [STD-008-CPP] | Do not delete an array through a pointer of the incorrect type |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an array of Derived objects is created and the pointer is stored in a Base \*. Despite Base::~Base() being declared virtual, it still results in undefined behavior. Further, attempting to perform pointer arithmetic on the static type Base \* violates CTR56-CPP. Do not use pointer arithmetic on polymorphic objects. |
| struct Base {  virtual ~Base() = default;  };    struct Derived final : Base {};    void f() {  Base \*b = new Derived[10];  // ...  delete [] b;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the static type of b is Derived \*, which removes the undefined behavior when indexing into the array as well as when deleting the pointer. |
| struct Base {  virtual ~Base() = default;  };    struct Derived final : Base {};    void f() {  Derived \*b = new Derived[10];  // ...  delete [] b;  } |

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

| **Principles(s):** Attempting to destroy an array of polymorphic objects through the incorrect static type is undefined behavior. In practice, potential consequences include abnormal program execution and memory leaks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -analyzer-checker=cplusplus | Checked with clang -cc1 or (preferably) scan-build |
| CodeSonar | 7.2p0 | ALLOC.TM | Type Mismatch |
| Helix QAC | 2022.4 | C++3166 |  |
| Klocwork | 2022.4 | CERT.EXPR.DELETE\_ARR.BASE\_PTR |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-EXP51-a | Do not treat arrays polymorphically |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2022b |  | Checks for delete operator used to destroy downcast object of different type. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Input/Output** | [STD-009-CPP] | Do not alternately input and output from a file stream without an intervening positioning call. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example appends data to the end of a file and then reads from the same file. However, because there is no intervening positioning call between the formatted output and input calls, the behavior is undefined. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }    file << "Output some data";  std::string str;  file >> str;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::basic\_istream<T>::seekg() function is called between the output and input, eliminating the undefined behavior. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }    file << "Output some data";    std::string str;  file.seekg(0, std::ios::beg);  file >> str;  } |

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

| **Principles(s):** Alternately inputting and outputting from a stream without an intervening flush or positioning call is undefined behavior. |
| --- |

**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 | [Insert text.] |
| CodeSonar | 7.2p0 | IO.IOWOP  IO.OIWOP | Input After Output Without Positioning  Output After Input Without Positioning |
| Helix QAC | 2022.4 | DF4711, DF4712, DF4713 |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-FIO50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| Polyspace Bug Finder | R2022b | CERT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Object Oriented Programming (OOP)** | [STD-010-CPP] | Honor replacement handler requirements |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a replacement new\_handler is written to attempt to release salvageable resources when the dynamic memory manager runs out of memory. |
| #include <new>    void custom\_new\_handler() {  // Returns number of bytes freed.  extern std::size\_t reclaim\_resources();  reclaim\_resources();  }    int main() {  std::set\_new\_handler(custom\_new\_handler);    // ...  } |

| **Compliant Code** |
| --- |
| In this compliant solution, custom\_new\_handler() uses the return value from reclaim\_resources(). If it returns 0, then there will be insufficient memory for operator new to succeed. |
| #include <new>    void custom\_new\_handler() noexcept(false) {  // Returns number of bytes freed.  extern std::size\_t reclaim\_resources();  if (0 == reclaim\_resources()) {  throw std::bad\_alloc();  }  }    int main() {  std::set\_new\_handler(custom\_new\_handler);    // ...  } |

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

| **Principles(s):** Failing to meet the required behavior for a replacement handler results in undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2022.4 | DF4776, DF4777, DF4778, DF4779 |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-OOP56-a  CERT\_CPP-OOP56-b  CERT\_CPP-OOP56-c | Properly define terminate handlers  Properly define unexpected handlers  Properly define new handlers |

### 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.

Automation is always good to have throughout the entire process. This will include even after final release of the system. The system should always be updating/improving as more data on how the system is functioning. Also being mindful of Defense in Depth. Making sure that testing is done early and often. This will be used to detect flaws or vulnerabilities to be able to fix early.

### 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 | High | Probable | High | P6 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-009-CPP | Low | Likely | Medium | P6 | L2 |
| STD-010-CPP | Low | Probable | High | P2 | L3 |

### 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 the process of securely encoding data as it is written into storage and decrypting that data as it is pulled from storage for use. This will help prevent an attacker from accessing the unencrypted data. |
| Encryption at flight | Encryption of data in-flight is the process of securely encoding data as it is being transmitted. Depending on the how the data will be transferred will depend on the type and application of the encryption that will be needed. A web browser will always use secure protocols, however, something like email is encrypted prior to sending it. |
| Encryption in use | Encryption of data in-use is the process of protecting data as it is utilized in memory. Someone with access to random access memory could parse the memory and locate the encryption key used for the data at rest. After obtaining that key they could decrypt that data. |

| **Triple-A Framework** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process where the user is being confirmed as someone who has.  access to the system. This can be done with things like user login, passwords, possibly higher-level security such as secure tokens, PIN, 2 step authentication and other hardware credentials. |
| Authorization | Authorization is the level of access that a user has within the system. The user would be able to read, create, modify, or delete file within the database depending on the access that is authorized. |
| Accounting | Accounting is the process of monitoring what a user is doing with their level of access to the system. This keeps track of the databases that are being accessed, who accessed those database, and what the user did to the database. |

**\***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

-Principle 4: A simple solution, as to not make it to complex and susceptible to errors.

-Principle 5: As a default deny access until given access to all accounts.

-Principle 6: Minimize the privileges to complete the job.

-Principle 8: One layer may fail but will be back up by others.

-Principle 10: Maintain a secure standard of coding for the platform and language all the time.

* Firewall logs

-Principle 4: A simple solution, as to not make it to complex and susceptible to errors.

-Principle 5: As a default deny access until given access to all accounts.

-Principle 6: Minimize the privileges to complete the job.

-Principle 7: prevent unnecessary data from being transmitted-Principle 8: One layer may fail but will be back up by others.

-Principle 10: Maintain a secure standard of coding for the platform and language all the time.

* Anti-malware logs

-Principle 4: A simple solution, as to not make it to complex and susceptible to errors.

-Principle 5: As a default deny access until given access to all accounts.

-Principle 6: Minimize the privileges to complete the job.

-Principle 7: prevent unnecessary data from being transmitted -Principle 8: One layer may fail but will be back up by others.

-Principle 10: Maintain a secure standard of coding for the platform and language all the time.

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 | 2/12/2023 | Project One | Hillary Loyd | Hillary Loyd |
| [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 |