**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 | Validate all data from untrusted sources. Proper input validation can eliminate a vast majority of software vulnerabilities that exist in the world today. Always be suspicious of external data sources. This includes anything that originated from outside of the program itself. |
| 1. Heed Compiler Warnings | Code should be compiled using its highest warning level. Static and dynamic tools both can be used to detect additional security flaws that might exist in the program inherently. |
| 1. Architect and Design for Security Policies | Best practice is to design software that enforces security polices right from the start. If the system that is being developed requires different privileges at different times, it would be best practice to divide the program in such a way that only the subsystems that require those privileges are the ones able to access the data that is required. |
| 1. Keep It Simple | Remember to only make the design as complicated as it needs to be. The more complex the design, the more likelihood there is for error. One thing to keep in mind is that the effort to ensure software security increases linearly with the complexity of the system itself. |
| 1. Default Deny | Access should be denied by default. The protection scheme is what should identify conditions in which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Ensure that each part of the system overall only has the privileges that it needs for the program to function properly. For example, a function that performs calculations should not have the ability to access user account information. |
| 1. Sanitize Data Sent to Other Systems | Ensure that all data that is passed through complex subsystems is sanitized. This is the kind of attack that those with malicious intent can take advantage of through the use or SQL, command, and other injection attacks. This is different than input validation because complex subsystems do not necessarily understand the context in which a cell is formed. The function call is what understands the context of the data that is being formed, therefore it is what needs to perform the sanitation. |
| 1. Practice Defense in Depth | Multiple strategies should always be implemented for preventative measures. This ensures that multiple layers of defense are in place in the even that one fails or to minimize the effectiveness of the vulnerability in the event of a successful attack. |
| 1. Use Effective Quality Assurance Techniques | Quality testing is one of the best ways to verify that you have a secure system. Things such as fuzz and penetration testing should always be incorporated into the development timeline. Independent security reviews are also other strategies that can lead to a more secure system. |
| 1. Adopt a Secure Coding Standard | Apply secure coding practice guidelines based on your target development environment and language of choice. |

### 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** | **Do not define a C-style variadic function** |
| --- | --- | --- |
| **Data Type** | [STD-001-C++] | A variadic function using a C-style ellipsis has no mechanisms to check the type safety of arguments being passed to the function or to check that the number of arguments being passed matches the semantics of the function definition. |
|  |  |  |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses a C-style variadic function to add a series of integers together. The function reads arguments until the value 0 is found. Calling this function without passing the value 0 as an argument (after the first two arguments) results in undefined behavior. Furthermore, passing any type other than an int also results in undefined behavior. |
| #include <cstdarg>    int add(int first, int second, ...) {  int r = first + second;  va\_list va;  va\_start(va, second);  while (int v = va\_arg(va, int)) {  r += v;  }  va\_end(va);  return r;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, a variadic function using a function parameter pack is used to implement the add() function, allowing identical behavior for call sites. Unlike the C-style variadic function used in the noncompliant code example, this compliant solution does not result in undefined behavior if the list of parameters is not terminated with 0. Additionally, if any of the values passed to the function are not integers, the code is ill-formed rather than producing undefined behavior. |
| #include <type\_traits>    template <typename Arg, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  int add(Arg f, Arg s) { return f + s; }    template <typename Arg, typename... Ts, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  int add(Arg f, Ts... rest) {  return f + add(rest...);  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **function-ellipsis** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-dcl50-cpp | Checked by clang-tidy. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.0p0 | **LANG.STRUCT.ELLIPSIS** | Ellipsis |

#### Coding Standard 2

| **Coding Standard** | **Label** | Do not declare or define a reserved identifier |
| --- | --- | --- |
| **Data Value** | [STD-002-C++] | When specifying the type argument for the offsetof() macro, pass only a standard-layout class. The full description of a standard-layout class can be found in paragraph 7 of the [class] clause of the C++ Standard, or the type can be checked with the std::is\_standard\_layout<> type trait. When specifying the member designator argument for the offsetof() macro, do not pass a bit-field, static data member, or function member. Passing an invalid type or member to the offsetof() macro is undefined behavior. |

| **Noncompliant Code** |
| --- |
| When specifying the type argument for the offsetof() macro, pass only a standard-layout class. The full description of a standard-layout class can be found in paragraph 7 of the [class] clause of the C++ Standard, or the type can be checked with the std::is\_standard\_layout<> type trait. When specifying the member designator argument for the offsetof() macro, do not pass a bit-field, static data member, or function member. Passing an invalid type or member to the offsetof() macro is [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <cstddef>    struct D {    virtual void f() {}    int i;  };    void f() {    size\_t off = offsetof(D, i);    // ...  } |

| **Compliant Code** |
| --- |
| It is not possible to determine the offset to i within D because D is not a standard-layout class. However, it is possible to make a standard-layout class within D if this functionality is critical to the application, as demonstrated by this compliant solution. |
| #include <cstddef>    struct D {    virtual void f() {}    struct InnerStandardLayout {      int i;    } inner;  };    void f() {    size\_t off = offsetof(D::InnerStandardLayout, i);    // ...  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **reserved-identifier** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL51** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreserved-id-macro -Wuser-defined-literals | The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverything. This flag does not catch all instances of this rule, such as redefining reserved names. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.0p0 | **LANG.ID.NU.MK**  **LANG.STRUCT.DECL.RESERVED** | Macro name is C keyword  Declaration of reserved name |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | [STD-003-C++] | Copying data to a buffer that is not large enough to hold that data results in a buffer overflow. Buffer overflows occur frequently when manipulating strings [Seacord 2013]. To prevent such errors, either limit copies through truncation or, preferably, ensure that the destination is of sufficient size to hold the data to be copied. C-style strings require a null character to indicate the end of the string, while the C++ std::basic\_string template requires no such character. |
|  |  |  |

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

| **Compliant Code** |
| --- |
| The best solution for ensuring that data is not truncated and for guarding against buffer overflows is to use std::string instead of a bounded array, as in this compliant solution. |
| #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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | **P18** | **L1** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.0p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.1 | **C++2835, C++2836, C++2839, C++5216** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.1 | **NNTS.MIGHT** **NNTS.TAINTED** **NNTS.MUST** **SV.UNBOUND\_STRING\_INPUT.CIN** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Use valid iterator ranges** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C++] | When iterating over elements of a container, the iterators used must iterate over a valid range. An iterator range is a pair of iterators that refer to the first and past-the-end elements of the range respectively.  A valid iterator range has all of the following characteristics:  Both iterators refer into the same container.  The iterator representing the start of the range precedes the iterator representing the end of the range.  The iterators are not invalidated, in conformance with CTR51-CPP. Use valid references, pointers, and iterators to reference elements of a container.  An empty iterator range (where the two iterators are valid and equivalent) is considered to be valid.  Using a range of two iterators that are invalidated or do not refer into the same container results in undefined behavior. |

| **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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **overflow\_upon\_dereference** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.0p0 | **LANG.MEM.BO** | Buffer Overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.1 | **C++3802** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-CTR53-a** **CERT\_CPP-CTR53-b** | Do not use an iterator range that isn't really a range Do not compare iterators from different containers |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-C++] | Evaluating a pointer—including dereferencing the pointer, using it as an operand of an arithmetic operation, type casting it, and using it as the right-hand side of an assignment—into memory that has been deallocated by a memory management function is undefined behavior. Pointers to memory that has been deallocated are called dangling pointers. Accessing a dangling pointer can result in exploitable vulnerabilities.  It is at the memory manager's discretion when to reallocate or recycle the freed memory. When memory is freed, all pointers into it become invalid, and its contents might either be returned to the operating system, making the freed space inaccessible, or remain intact and accessible. As a result, the data at the freed location can appear to be valid but change unexpectedly. Consequently, memory must not be written to or read from once it is freed. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    s->f();    delete s;  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **dangling\_pointer\_use** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MEM50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.0p0 | **ALLOC.UAF** | Use after free |

#### 

#### Coding Standard 6

| **Coding Standard** | **Label** | **Avoid information leakage when passing a class**  **object across a trust boundary** |
| --- | --- | --- |
| **Assertions** | [STD-006-C++] | Non-static data members of a (non-union) class with the same access control are allocated so that later members have higher addresses within a class object. The order of allocation of non-static data members with different access control is unspecified. Implementation alignment requirements might cause two adjacent members not to be allocated immediately after each other; so might requirements for space for managing virtual functions and virtual base classes. |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] This noncompliant code example runs in kernel space and copies data from arg to user space. However, padding bits may be used within the object, for example, to ensure the proper alignment  of class data members. These padding bits may contain sensitive information that may then be  leaked when the data is copied to user space, regardless of how the data is copied. |
| #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** |
| --- |
| This compliant solution serializes the structure data before copying it to an untrusted context |
| #include <cstddef>  #include <cstring>  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};  // May be larger than strictly needed.  unsigned char buf[sizeof(arg)];  std::size\_t offset = 0;  std::memcpy(buf + offset, &arg.a, sizeof(arg.a));  offset += sizeof(arg.a);  std::memcpy(buf + offset, &arg.b, sizeof(arg.b));  offset += sizeof(arg.b);  std::memcpy(buf + offset, &arg.c, sizeof(arg.c));  offset += sizeof(arg.c);  copy\_to\_user(usr\_buf, buf, offset /\* size of info copied \*/); |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL55** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.0p0 | **MISC.PADDING.POTB** | Padding Passed Across a Trust Boundary |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.1 | **C++4941, C++4942, C++4943** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-DCL55-a** | A pointer to a structure should not be passed to a function that can copy data to the user space |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Do not abruptly terminate the program** |
| --- | --- | --- |
| **Exceptions** | [STD-007-C++] | The std::abort(), std::quick\_exit(), and std::\_Exit() functions are used to terminate the program in an immediate fashion. They do so without calling exit handlers registered with std::atexit() and without executing destructors for objects with automatic, thread, or static storage duration. How a system manages open streams when a program ends is implementation-defined [ISO/IEC 9899:1999]. Open streams with unwritten buffered data may or may not be flushed, open streams may or may not be closed, and temporary files may or may not be removed. Because these functions can leave external resources, such as files and network communications, in an indeterminate state, they should be called explicitly only in direct response to a critical error in the application. (See ERR50-CPP-EX1 for more information.) |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the call to f(), which was registered as an exit handler with std::at\_exit(), may result in a call to std::terminate() because throwing\_func() may throw an exception. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    int main() {    if (0 != std::atexit(f)) {      // Handle error    }    // ...  } |

| **Compliant Code** |
| --- |
| In this compliant solution, f() handles all exceptions thrown by throwing\_func() and does not rethrow. |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

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

| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.    try {      throwing\_func();    } catch (...) {      // Handle error    }  }    int main() {    if (0 != std::atexit(f)) {      // Handle error    }    // ...  } |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **stdlib-use** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.0p0 | **BADFUNC.ABORT BADFUNC.EXIT** | Use of abort Use of exit |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.1 | **C++5014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.1 | **MISRA.TERMINATE** **CERT.ERR.ABRUPT\_TERM** |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not invoke virtual functions from constructors or destructors** |
| --- | --- | --- |
| Object Oriented Programming | [STD-008-C++] | Virtual functions allow for the choice of member function calls to be determined at run time based on the dynamic type of the object that the member function is being called on. This convention supports object-oriented programming practices commonly associated with object inheritance and function overriding. When calling a nonvirtual member function or when using a class member access expression to denote a call, the specified function is called. Otherwise, a virtual function call is made to the final overrider in the dynamic type of the object expression. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the base class attempts to seize and release an object's resources through calls to virtual functions from the constructor and destructor. However, the B::B() constructor calls B::seize() rather than D::seize(). Likewise, the B::~B() destructor calls B::release() rather than D::release(). |
| struct B {    B() { seize(); }    virtual ~B() { release(); }    protected:    virtual void seize();    virtual void release();  };    struct D : B {    virtual ~D() = default;    protected:    void seize() override {      B::seize();      // Get derived resources...    }      void release() override {      // Release derived resources...      B::release();    }  }; |

| **Compliant Code** |
| --- |
| [Compliant description] In this compliant solution, the constructors and destructors call a nonvirtual, private member function (suffixed with mine) instead of calling a virtual function. The result is that each class is responsible for seizing and releasing its own resources. |
| class B {    void seize\_mine();    void release\_mine();    public:    B() { seize\_mine(); }    virtual ~B() { release\_mine(); }    protected:    virtual void seize() { seize\_mine(); }    virtual void release() { release\_mine(); }  };    class D : public B {    void seize\_mine();    void release\_mine();    public:    D() { seize\_mine(); }    virtual ~D() { release\_mine(); }    protected:    void seize() override {      B::seize();      seize\_mine();    }      void release() override {      release\_mine();      B::release();    }  }; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **virtual-call-in-constructor invalid\_function\_pointer** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-OOP50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-alpha.cplusplus.VirtualCall | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.0p0 | **LANG.STRUCT.VCALL\_IN\_CTOR**  **LANG.STRUCT.VCALL\_IN\_DTOR** | Virtual Call in Constructor  Virtual Call in Destructor |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do not slice derived objects** |
| --- | --- | --- |
| Slicing | [STD-009-C++] | An object deriving from a base class typically contains additional member variables that extend the base class. When by-value assigning or copying an object of the derived type to an object of the base type, those additional member variables are not copied because the base class contains insufficient space in which to store them. This action is commonly called slicing the object because the additional members are "sliced off" the resulting object.  Do not initialize an object of base class type with an object of derived class type, except through references, pointers, or pointer-like abstractions (such as std::unique\_ptr, or std::shared\_ptr). |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an object of the derived Manager type is passed by value to a function accepting a base Employee type. Consequently, the Manager objects are sliced, resulting in information loss and unexpected behavior when the print() function is called. |
| #include <iostream>  #include <string>    class Employee {    std::string name;    protected:    virtual void print(std::ostream &os) const {      os << "Employee: " << get\_name() << std::endl;    }    public:    Employee(const std::string &name) : name(name) {}    const std::string &get\_name() const { return name; }    friend std::ostream &operator<<(std::ostream &os, const Employee &e) {      e.print(os);      return os;    }  };    class Manager : public Employee {    Employee assistant;    protected:    void print(std::ostream &os) const override {      os << "Manager: " << get\_name() << std::endl;      os << "Assistant: " << std::endl << "\t" << get\_assistant() << std::endl;    }    public:    Manager(const std::string &name, const Employee &assistant) : Employee(name), assistant(assistant) {}    const Employee &get\_assistant() const { return assistant; }  };    void f(Employee e) {    std::cout << e;  }    int main() {    Employee coder("Joe Smith");    Employee typist("Bill Jones");    Manager designer("Jane Doe", typist);      f(coder);    f(typist);    f(designer);  } |

| **Compliant Code** |
| --- |
| [Compliant description] Using the same class definitions as the noncompliant code example, this compliant solution modifies the definition of f() to require raw pointers to the object, removing the slicing problem. |
| // Remainder of code unchanged...    void f(const Employee \*e) {    if (e) {      std::cout << \*e;    }  }    int main() {    Employee coder("Joe Smith");    Employee typist("Bill Jones");    Manager designer("Jane Doe", typist);      f(&coder);    f(&typist);    f(&designer);  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.0p0 | **LANG.CAST.OBJSLICE** | Object Slicing |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.1 | **C++3072** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-OOP51-a** | Avoid slicing function arguments / return value |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022a | [CERT C++: OOP51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop51cpp.html) | Checks for object slicing (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do not delete a polymorphic object without a virtual destructor** |
| --- | --- | --- |
| Polymorphism | [STD-010-C++] | Do not delete a polymorphic object without a virtual destructor |

| **Noncompliant Code** |
| --- |
| [In this noncompliant example, b is a polymorphic pointer type whose static type is Base \* and whose dynamic type is Derived \*. When b is deleted, it results in undefined behavior because Base does not have a virtual destructor. The C++ Standard, [class.dtor], paragraph 4 [ISO/IEC 14882-2014], states the following:  If a class has no user-declared destructor, a destructor is implicitly declared as defaulted. An implicitly declared destructor is an inline public member of its class.  The implicitly declared destructor is not declared as virtual even in the presence of other virtual functions. |
| struct Base {    virtual void f();  };    struct Derived : Base {};    void f() {    Base \*b = new Derived();    // ...    delete b;  } |

| **Compliant Code** |
| --- |
| [Compliant description] In this compliant solution, the destructor for Base has an explicitly declared virtual destructor, ensuring that the polymorphic delete operation results in well-defined behavior. |
| struct Base {    virtual ~Base() = default;    virtual void f();  };    struct Derived : Base {};    void f() {    Base \*b = new Derived();    // ...    delete b;  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **non-virtual-public-destructor-in-non-final-class** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-OOP52** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wdelete-non-virtual-dtor |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.0p0 | **LANG.STRUCT.DNVD** | delete with Non-Virtual Destructor |

### 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 | Probable | Medium | P12 | L1 |
| STD-002-CPP | Low | Unlikely | Low | **P3** | **L3** |
| STD-003-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-004-CPP | High | Probable | High | **P6** | **L2** |
| STD-005-CPP | Low | Unlikely | High | **P1** | **L3** |
| STD-006-CPP | Low | Unlikely | High | **P1** | **L3** |
| STD-007-CPP | Low | Probable | Medium | **P4** | **L3** |
| STD-008-CPP | Low | Unlikely | Medium | **P2** | **L3** |
| STD-009-CPP | Unlikely | Medium | **P2** | **L3** | Low |
| STD- 010-CPP | Low | Likely | Low | **P9** | **L2** |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | **Encryption-at-rest** is a concept in where data that is not being actively utilized (say on a hard disk or database) is encrypted. The goal of this policy is to protect data in the event of a breech and these files are stolen. The attacker would have to brute force the encryption (which would take a long period of time) or gain access to the encryption keys to be able to decrypt the data and make it actionable. This type of process would take several years and limits the effectiveness of the data stolen. |
| Encryption at flight | **Encryption-in-flight** refers to the concept or protecting data as its moving around the network. For example, a web application pulling data from a database. This policy is important as it attempts to protect data while arguably at its most vulnerable point and subject to the most exposure. Protecting data in flight is achieved through the use of SSL/TLS connections between web server and database. Additional measure of protections is the use of VPN where network segments need to be joined. This ensures that normal packet sniffing applications like WireShark and TCPDump are not able to read the packets of the transmission and piece them together. |
| Encryption in use | **Encryption-in-Use** refers to the concept of protecting data when its actively being utilized. For example, when a web server has pulled data from a database and is making computations/delivering data to a consumer. Protecting data-in-use is accomplished through the use of programming techniques like utilizing protected memory (like .NET ProtectedMemory class) and Homomorphic encryption which allows manipulating encrypted strings as it were plaintext. Utilizing these safeguards limits attacking an application server in its usefulness, as you would need to breach these types of safeguards using brute force. This limits the data’s usefulness as this type of decryption can take years. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | **Authentication** is the process in which a server or application proves that you are who you say you are. Authentication is important as it determines who should and who should not have access to the application as a whole. This process is governed through a logon process. Usually, a user has a logon that is tied to their email address or specific user name. A password is provided to verify the login holder’s access. Additional forms of authentication should be included, such as 2 factor OTP or utilizing OAUTH technology for SSO. This policy should apply to all resources to prevent unauthorized access by a third party and to lock data behind a door where you need a key to access it. |
| Authorization | **Authorization** is the process that occurs after **Authentication** and grants the level of access that you have to a system, its files, and its resources. Authorization is used to control a user’s level of access, most often through role-based permissions, that define what a user can do. Tying this with the principal of **Default Deny** new users should not inherently have access to any resources even if authenticated. Only through the granting of a role should access be allowed to a particular resource. This policy is important to upheld as it prevents authenticated users without the proper authorization from accessing data, they should not be able to access. Additionally, it makes it very ease to ‘promote’ someone into a higher tier with better resource access. |
| Accounting | **Accounting** is the process of tracking changes to a particular system or resource. An example of accounting is tracking what files are accessed by users and what changes are made to database files. While systems can be built to provide this functionality, there are many software-based applications that provide aggregation for this data and allow reports to be run off of it. An example of this type of system is Thycotic Secret Server. This is important system to have as its both informative before and after a breach. If a specific user account is accessing a resource before a breech, having this information allows the security team to scrutinize this access and potentially prevent a breech. Immediately after a breech, this information is important to performing an root cause analysis on the breach, and to seal that pathway up as fast as possible. This policy applies to our scenario, as we want to protect unauthorized intrusion on our protected data. |

**\***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 |
| --- | --- | --- | --- | --- |
| Version | Date | Description | Edited By | Approved By |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 04/17/2021 | Initial Milestone Completion | Michael Mihalik | David Buksbaum |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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