**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 | Input data is the first line of defense for secure coding. Data must be identified, and validated against minimum and maximum value, length, and content specifications. Any data that can be manipulated by an attacker should be validated before anything is done with the data. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies by looking at how information controls and securities are implemented in systems. |
| 1. Keep It Simple | It is beneficial to keep the design as simple as possible. Complex designs increase the likelihood of errors and makes configuration and troubleshooting more challenging. |
| 1. Default Deny | By default, access should be denied, and access decisions should be based on permissions rather than exclusion by creating code that denies access to resources unless a subject can demonstrate they are authorized to have access. |
| 1. Adhere to the Principle of Least Privilege | A concept that limits user’s rights to only what they are required to perform their job reducing the opportunity for an attacker to execute arbitrary code with higher privileges. |
| 1. Sanitize Data Sent to Other Systems | A cybersecurity measure of checking, cleaning, and filtering data inputs from users to avoid attackers to utilize unused functionality in relational databases, command shells, and commercial-off-the-shelf components. |
| 1. Practice Defense in Depth | Practice multiple defensive strategies, so if one layer of defense is not adequate, another layer of defense will prevent a security flaw from being exploited. |
| 1. Use Effective Quality Assurance Techniques | Effective techniques like penetration testing, fuzz testing, source code audits that identify and eliminate vulnerabilities. Independent perspectives can be brought in by external reviews. |
| 1. Adopt a Secure Coding Standard | Develop secure coding by applying a set of rules and guidelines for a target development platform to reduce security vulnerabilities and errors. |

### 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** | [INT-50-CPP] | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| Checks if a value is within range of acceptable enumeration value. Once the type is casted it may not be able to represent the given integer value. |
| Enum EnumType {  Frist,  Second,  Third  };  Void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  If (enumVar < First || enumVar > Third) {  //Handle errors  }  } |

| **Compliant Code** |
| --- |
| The complaint solution checks the value represented by the enumeration type before the conversion to guarantee the conversion doesn’t result in an unspecific value. Instead, it restricts the converted value to ine specific enumerator type. |
| Enum EnumType {  First,  Second,  Third  };  Void f(int intVar) {  If (intVar < First || intVar > Third) {  //Handles error  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** A value of integral or enumeration type can be explicitly converted to a complete enumeration type. If the enumeration type has a fixed underlying type, the value is first converted to that type by integral conversion, if necessary, and then to the enumeration type. If the enumeration type does not have a fixed underlying type, the value is unchanged if the original value is within the range of the enumeration values (9.7.1), and otherwise, the behavior is undefined. A value of floating-point type can also be explicitly converted to an enumeration type. The resulting value is the same as converting the original value to the underlying type of the enumeration (7.3.10), and subsequently to the enumeration type. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | **cast-integer-to-enum** | Partially Checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-INT50** |  |
| CodeSonar | 7.1p0 | **LANG.CAST.COERCE**  **LANG.CAST.VALUE** | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2022.3 | **C++3013** |  |
| Parasoft C/C++test | 2022.1 | **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 | 7.21 | **V1016** |  |
| RuleChecker | 22.10 | **Cast-integer-to-enum** | Partially Checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [DCL-50-CPP] | Do not define a C-style variadic function |

| **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** |
| --- |
| 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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-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):** Validating Input Data - ensuring proper inputs  Architect and Design for Security Policies - building code to prevent vulnerabilities  Keep it simple - always applies as keeping code as lightweight as possible is best practice |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | **function-ellipsis** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL50** | [Insert text.] |
| Clang | 3.9 | **Cert-dclt50-cpp** | Checked by clangptidy |
| CodeSonar | 7.1p0 | **LANG.STRUCT.ELLIPSIS** | Ellipsis |
| Helix QAC | 2022.3 | **C++2012, C++2625** |  |
| Klocwork | 2022.3 | **MISRA.FUNC.VARARG** |  |
| LDRA tool suite | 9.7.1 | **41 S** | Fully Implemented |
| Parasoft C/C++test | 2022.1 | **CERT\_CPP-DCL50-a** | Functions shall not be defined with a variable number of arguments |
| Polyspace Bug Finder | R2022b | **Cert C++: DCL50-CPP** | Checks for function definition with ellipsis notation (rule fully covered) |
| PRQA QA C++ | 4.4 | **2012, 2625** |  |
| RuleChecker | 22.10 | **Function-ellipsis** | Fully Checked |
| SonarQube C/C++ Plugin | 4.10 | **FunctionEllipsis** |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [DCL-51-CPP] | Do not declare or define a reserved identifier |

| **Noncompliant Code** |
| --- |
| A common practice is to use a macro in a preprocessor conditional that guards against multiple inclusions of a header file. While this is a recommended practice, many programs use reserved names as the header guards. Such a name may clash with reserved names defined by the implementation of the C++ standard template library in its headers or with reserved names implicitly predefined by the compiler even when no C++ standard library header is included. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| This compliant solution avoids using leading or trailing underscores in the name of the header guard. |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    // Contents of <my\_header.h>    #endif // MY\_HEADER\_H |

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

| **Principles(s):** Heed compiler warnings - always pay attention to warnings as they are there for a reason  Architect and Design for Security Policies - building code to prevent vulnerabilities  Keep it simple - always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques - making tests that are as effective as possible  Adopt a secure coding standard - making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | **Reserved-identifier** | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL51** |  |
| 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 | 7.1p0 | **LANG.ID.NU.MK**  **LANG.STRUCT.DECL.RESERVED** | Macro name is C keyword  Declaration of reserved name |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [DCL-53-CPP] | Do not write syntactically ambiguous declarations |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an anonymous local variable of type std::unique\_lock is expected to lock and unlock the mutex m by virtue of RAII. However, the declaration is syntactically ambiguous as it can be interpreted as declaring an anonymous object and calling its single-argument converting constructor or interpreted as declaring an object named m and default constructing it. The syntax used in this example defines the latter instead of the former, and so the mutex object is never locked. |
| #include <mutex>    **static** std::mutex m;  **static** **int** shared\_resource;    **void** increment\_by\_42() {    std::unique\_lock<std::mutex>(m);    shared\_resource += 42;  } |

| **Compliant Code** |
| --- |
| The lock object is given an identifier (other than m) and the proper converting constructor is called. |
| #include <mutex>    **static** std::mutex m;  **static** **int** shared\_resource;    **void** increment\_by\_42() {    std::unique\_lock<std::mutex> lock(m);    shared\_resource += 42;  } |

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

| **Principles(s):** Validating Input Data - ensuring proper inputs  Architect and Design for Security Policies - building code to prevent vulnerabilities  Keep it simple - always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques - making tests that are as effective as possible  Adopt a secure coding standard - making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | **LANG.STRUCT.DECL.FNEST** | Nested Function Declaration |
| Helix QAC | 2022.3 | **C++1109, C++2510** |  |
| KlocWork | 2022.3 | **CERT.DCL.AMBIGUOUS\_DECL** |  |
| LDRA tool suite | 9.7.1 | **296 S** |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [DCL-54-CPP] | Overload allocation and deallocation functions as a pair in the same scope |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an allocation function is overloaded at global scope. However, the corresponding deallocation function is not declared. |
| #include <Windows.h>  #include <new>    **void** \*operator **new**(std::**size\_t** size) noexcept(**false**) {  **static** **HANDLE** h = ::HeapCreate(0, 0, 0); // Private, expandable heap.  **if** (h) {  **return** ::HeapAlloc(h, 0, size);    }  **throw** std::bad\_alloc();  }    // No corresponding global delete operator defined. |

| **Compliant Code** |
| --- |
| the corresponding deallocation function is also defined at global scope. |
| #include <Windows.h>  #include <new>    **class** HeapAllocator {  **static** **HANDLE** h;  **static** **bool** init;    **public**:  **static** **void** \*alloc(std::**size\_t** size) noexcept(**false**) {  **if** (!init) {        h = ::HeapCreate(0, 0, 0); // Private, expandable heap.        init = **true**;      }    **if** (h) {  **return** ::HeapAlloc(h, 0, size);      }  **throw** std::bad\_alloc();    }    **static** **void** dealloc(**void** \*ptr) noexcept {  **if** (h) {        (**void**)::HeapFree(h, 0, ptr);      }    }  };    **HANDLE** HeapAllocator::h = nullptr;  **bool** HeapAllocator::init = **false**;    **void** \*operator **new**(std::**size\_t** size) noexcept(**false**) {  **return** HeapAllocator::alloc(size);  }    **void** operator **delete**(**void** \*ptr) noexcept {  **return** HeapAllocator::dealloc(ptr);  } |

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

| **Principles(s):** Validating Input Data - ensuring proper inputs  Architect and Design for Security Policies - building code to prevent vulnerabilities  Keep it simple - always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques - making tests that are as effective as possible  Adopt a secure coding standard - making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | **New-delete-pairwise** | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL54** |  |
| Clang | 3.9 | **Misc-new-delete-overloads** | Checked with clang-tidy |
| Helix QAC | 2022.3 | **C++2160** |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [DCL-55-CPP] | Avoid information leakage when passing a class object across a trust boundary |

| **Noncompliant Code** |
| --- |
| 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):** Validating Input Data - ensuring proper inputs  Architect and Design for Security Policies - building code to prevent vulnerabilities  Sanitize data sent to other systems - keeping data sent to other systems as only required/authorized  Keep it simple - always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques - making tests that are as effective as possible  Adopt a secure coding standard - making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL55** |  |
| CodeSonar | 7.1p0 | **MISC.PADDING.POTB** | Padding Passed Across a Trust Boundary |
| Helix QAC | 2022.3 | **C++4941, C++4942, C++4943** |  |
| PARASOFT C/C++TEST | 2022.1 | **Cert\_cpp-dcl55-a** | A pointer to a structure should not be passed to a function that can copy data to the user space |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [DCL-56-CPP] | Avoid cycles during initialization of static objects |

| **Noncompliant Code** |
| --- |
| The initialization of the static local array cache involves recursion, the behavior of the function is undefined, even though the recursion is not infinite. |
| #include <stdexcept>    **int** fact(**int** i) noexcept(**false**) {  **if** (i < 0) {      // Negative factorials are undefined.  **throw** std::domain\_error("i must be >= 0");    }    **static** **const** **int** cache[] = {      fact(0), fact(1), fact(2), fact(3), fact(4), fact(5),      fact(6), fact(7), fact(8), fact(9), fact(10), fact(11),      fact(12), fact(13), fact(14), fact(15), fact(16)    };    **if** (i < (**sizeof**(cache) / **sizeof**(**int**))) {  **return** cache[i];    }    **return** i > 0 ? i \* fact(i - 1) : 1;  } |

| **Compliant Code** |
| --- |
| This compliant solution avoids initializing the static local array cache and instead relies on zero-initialization to determine whether each member of the array has been assigned a value yet and, if not, recursively computes its value. It then returns the cached value when possible or computes the value as needed. |
| #include <stdexcept>    **int** fact(**int** i) noexcept(**false**) {  **if** (i < 0) {      // Negative factorials are undefined.  **throw** std::domain\_error("i must be >= 0");    }      // Use the lazy-initialized cache.  **static** **int** cache[17];  **if** (i < (**sizeof**(cache) / **sizeof**(**int**))) {  **if** (0 == cache[i]) {        cache[i] = i > 0 ? i \* fact(i - 1) : 1;      }  **return** cache[i];    }    **return** i > 0 ? i \* fact(i - 1) : 1;  } |

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

| **Principles(s):** Architect and Design for Security Policies - building code to prevent vulnerabilities  Keep it simple - always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques - making tests that are as effective as possible  Adopt a secure coding standard - making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | **LANG.STRUCT.INIT.CYCLE**  **LANG.STRUCT.INIT.UNORDERED** | Initialization Cycle  Unordered Initialization |
| Helix QAC | 2022.3 | **C++1552, C++1554, C++1704** |  |
| LDRA tool suite | 9.7.1 | **6 D** | Enhanced Enforcement |
| Parsoft C/C++test | 2002.1 | **CERT\_CPP-DCL56-a** | Avoid initialization order problems across translation units by replacing non-local static objects with local static objects |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [DCL-57-CPP] | Do not let exceptions escape from destructors or deallocation functions |

| **Noncompliant Code** |
| --- |
| the class destructor does not meet the implicit noexcept guarantee because it may throw an exception even if it was called as the result of an exception being thrown. Consequently, it is declared as noexcept(false) but still can trigger undefined behavior. |
| #include <stdexcept>    **class** S {  **bool** has\_error() **const**;    **public**:    ~S() noexcept(**false**) {      // Normal processing  **if** (has\_error()) {  **throw** std::logic\_error("Something bad");      }    }  }; |

| **Compliant Code** |
| --- |
| A destructor should perform the same way whether or not there is an active exception. Typically, this means that it should invoke only operations that do not throw exceptions, or it should handle all exceptions and not rethrow them (even implicitly). This compliant solution differs from the previous noncompliant code example by having an explicit return statement in the SomeClass destructor. This statement prevents control from reaching the end of the exception handler. Consequently, this handler will catch the exception thrown by Bad::~Bad() when bad\_member is destroyed. It will also catch any exceptions thrown within the compound statement of the *function-try-block*, but the SomeClass destructor will not terminate by throwing an exception. |
| **class** SomeClass {    Bad bad\_member;  **public**:    ~SomeClass()  **try** {      // ...    } **catch**(...) {      // Catch exceptions thrown from noncompliant destructors of      // member objects or base class subobjects.        // NOTE: Flowing off the end of a destructor function-try-block causes      // the caught exception to be implicitly rethrown, but an explicit      // return statement will prevent that from happening.  **return**;    }  }; |

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

| **Principles(s):** Architect and Design for Security Policies - building code to prevent vulnerabilities  Keep it simple - always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques - making tests that are as effective as possible  Adopt a secure coding standard - making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | **destructor-without-noexcept delete-without-noexcept** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++DCL57** |  |
| CodeSonar | 7.1p0 | **LANG.STRUCT.EXCP.CATCH**  **LANG.STRUCT.EXCP.THROW** | Use of catch  Use of throw |
| Helix QAC | 2022.3 | **C++2045, C++2047, C++4032, C++4631** |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [DCL-58-CPP] | Do not modify the standard namespaces |

| **Noncompliant Code** |
| --- |
| The declaration of x is added to the namespace std, resulting in undefined behavior. |
| **namespace** std {  **int** x;  } |

| **Compliant Code** |
| --- |
| This compliant solution assumes the intention of the programmer was to place the declaration of x into a namespace to prevent collisions with other global identifiers. Instead of placing the declaration into the namespace std, the declaration is placed into a namespace without a reserved name. |
| **namespace** nonstd {  **int** x;  } |

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

| **Principles(s):** Architect and Design for Security Policies - building code to prevent vulnerabilities  Keep it simple - always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques - making tests that are as effective as possible  Adopt a secure coding standard - making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL58** |  |
| CodeSonar | 7.2.0 | **LANG.STRUCT.DECL.SNM** | Modification of Standard Namespaces |
| Helix QAC | 2022.3 | **C++3180, C++3181, C++3182** |  |
| Klocwork | 2022.3 | **CERT.DCL.STD\_NS\_MODIFIED** |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [DCL-59-CPP] | Do not define an unnamed namespace in a header file |

| **Noncompliant Code** |
| --- |
| The variable v is defined in an unnamed namespace within a header file and is accessed from two separate translation units. Each translation unit prints the current value of v and then assigns a new value into it. However, because v is defined within an unnamed namespace, each translation unit operates on its own instance of v, resulting in unexpected output. |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE    **namespace** {  **int** v;  }    #endif // A\_HEADER\_FILE    // a.cpp  #include "a.h"  #include <iostream>    **void** f() {    std::cout << "f(): " << v << std::endl;    v = 42;    // ...  }    // b.cpp  #include "a.h"  #include <iostream>    **void** g() {    std::cout << "g(): " << v << std::endl;    v = 100;  }    **int** main() {  **extern** **void** f();    f(); // Prints v, sets it to 42    g(); // Prints v, sets it to 100    f();    g();  } |

| **Compliant Code** |
| --- |
| v is defined in only one translation unit but is externally visible to all translation units, resulting in the expected behavior. |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE    **extern** **int** v;    #endif // A\_HEADER\_FILE    // a.cpp  #include "a.h"  #include <iostream>    **int** v; // Definition of global variable v    **void** f() {    std::cout << "f(): " << v << std::endl;    v = 42;    // ...  }    // b.cpp  #include "a.h"  #include <iostream>    **void** g() {    std::cout << "g(): " << v << std::endl;    v = 100;  }    **int** main() {  **extern** **void** f();    f(); // Prints v, sets it to 42    g(); // Prints v, sets it to 100    f(); // Prints v, sets it back to 42    g(); // Prints v, sets it back to 100  } |

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

| **Principles(s):** Architect and Design for Security Policies - building code to prevent vulnerabilities  Keep it simple - always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques - making tests that are as effective as possible  Adopt a secure coding standard - making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | **unnamed-namespace-header** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL59** |  |
| Clang | 3.9 | cert-dcl59-cpp | Checked by clang-tidy |
| CodeSonar | 7.1p0 | **LANG.STRUCT.DECL.ANH** | Anonymous Namespace in Header File |

### 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 |
| --- | --- | --- | --- | --- | --- |
| INT-50-CPP | Medium | Unlikely | Medium | P4 | L3 |
| DCL-50-CPP | High | Probable | Medium | P12 | L1 |
| DCL-51-CPP | Low | Unlikely | Low | P3 | L3 |
| DCL-53-CPP | Low | Unlikely | Medium | P2 | L3 |
| DCL-54-CPP | Low | Probable | Low | P6 | L2 |
| DCL-55-CPP | Low | Unlikely | High | P1 | L3 |
| DCL-56-CPP | Low | Unlikely | Medium | P2 | L3 |
| DCL-57-CPP | Low | Likely | Medium | P6 | L2 |
| DCL-58-CPP | High | Unlikely | Medium | P6 | L2 |
| DCL-59-CPP | Medium | Unlikely | Medium | P4 | 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 | Data at Rest is data collected in a single file server, workstation, database, or cloud, and it tends to have a logical structure. Following the encryption guidelines set by the National Institute for Standards and Technology- Federal Information Processing Standards (NIST – FIPS) is the current best practice to protect data. Guidelines includes encryption needing to be reliable and robust, being strong enough to protect information from being disclosed, endpoints involved in encrypted communications are secure. Disk and filesystems should use a symmetric encryption key when the data is stored to protect it from unauthorized users. |
| Encryption at flight | Data at flight is data that is being moved from one endpoint to another, the fashion the data is being transmitted will determine how encryption will be applied. Avoiding the use of FTP servers for file transfer, instead using rsync over SSH or transferring files over Windows Remote Desktop. |
| Encryption in use | Data in use is the most vulnerable form of data, it happens when a user access or consumes an application, it is typically stored in clear text in the memory for the duration of its usage. Encryption of such data includes battling against cyber-attacks like ransomware. There are tools like Vaultree that allows data to be encrypted at nearly the same speed as processing plaint text data utilizing homomorphic encryption where text is never decrypted, even while the data is being worked with. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Part of the framework that can prove the user is who they say they are by providing a username, password, and additional authentication factors. Authentication methods can include biometrics, certificate-based, password-based, multifactor, or token-based. |
| Authorization | The authorization determines what type of access the user will have to the resources available in the network, enforcing policies determining the types of activities, resources, or services a user is permitted. |
| Accounting | Measures the resources a user consumes during access, including the amount of system time or data a user sent and received during a given session. Accounting is carried out by logging session statistics and usage information and the information is used for authorization control, billing, trend analysis, resource utilization and capacity planning. |

**\***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
  + - 4 – Keeping system designs simple
    - 5 – Deciding access based on permissions rather than exclusions
    - 8 – Practicing multiple defensive strategies
* Firewall logs
  + - 1 – Validating input data
    - 4 – Keeping system designs simple
    - 5 – Deciding access based on permissions rather than exclusions
    - 6 – Limiting user’s rights to only they require to perform their job
    - 7 – Sanitizing data being sent
    - 8 – Practicing multiple defensive strategies
* Anti-malware logs
  + - 4 – Keeping system designs simple
    - 5 – Deciding access based on permissions rather than exclusions
    - 8 – Practicing multiple defensive strategies
    - 9 – Using effective quality assurance techniques

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 | 11/12/2022 | 3-2 Milestone | Priscilla Gomes-DaCosta | Priscilla Gomes-DaCosta |
| 1.2 | 12/4/2022 | 6-2 Project One | Priscilla Gomes-DaCosta | Priscilla Gomes-DaCosta |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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