**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 | Always consider external data sources as untrusted sources. Meaning, doors by which security threats may enter the system. As such, all inputs from those “untrusted” sources must be validated in order to prevent software vulnerabilities. |
| 1. Heed Compiler Warnings | When compiling your code, always use the highest warning level available. Then proceed to eliminate warnings in the code. Other static and/or dynamic analysis tools are a great help in helping detecting security flaws. |
| 1. Architect and Design for Security Policies | Always make sure that proper security policies are at the forefront of the architecture design process. |
| 1. Keep It Simple | The simpler, the better. Always make sure functionalities are written with the less amount of code possible. Over-complicating the code has a greater change of introducing security flaws. |
| 1. Default Deny | The protection scheme should be deny access by default, and only allow access when certain criteria are met. This method is more restrictive, thus more secure than allowing access by default and only deny access when certain criteria are met. |
| 1. Adhere to the Principle of Least Privilege | Processes should execute with the least amount of privileges needed to complete the task. This way, if it is used to cause harm, the attacker will be limited and less likely to cause greater damage. |
| 1. Sanitize Data Sent to Other Systems | Unlike input validation, sanitizing data goes a step further. It allows the calling process to sanitize the data such as SQL queries and commands before invoking the called sub-system. The subsystem being called generally does not understand the context the call is made, but the calling process does. Thus, it is the responsibility of the calling process to sanitize the data before calling the subsystem to make sure it’s not passing harmful data to it |
| 1. Practice Defense in Depth | Use multiple layers of defense in your defense strategy. This ensures that if one layer fails or is circumvented, another lay will prevent the attacker from going any further, or at the very least, slow him down. |
| 1. Use Effective Quality Assurance Techniques | Quality Assurance can be effective at catching vulnerabilities early and fixing them before they are released and eventually exploited. It is good practice to have an external team review and assess the environment. Their lack of use to your environment/code will bring a fresh perspective that will be able to detect anomalies that an internal team would probably not be able to uncover. |
| 1. Adopt a Secure Coding Standard | Design and enforce a secure coding standard for your environment. This ensures that everyone follows the same principles and also have a standard to judge against when peer-reviewing codes. As a result, the code base is more uniform and more secure. |

### 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 define a C-style variadic function](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL50-CPP.+Do+not+define+a+C-style+variadic+function). They can introduce vulnerabilities because they don’t check the type safety of arguments being passed. |

| **Noncompliant Code** |
| --- |
| This function reads arguments until a 0-value is found. Calling this function without passing the value 0 as a argument, or passing any type other than an int will result in an 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** |
| --- |
| This complying code uses a function parameter pack to impletement an add() function. As a result, no undefined behavior will be produced if the list of parameters does not terminate with a 0. |
| #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): Incorrectly using a variadic function can result in [abnormal program termination](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-abnormaltermination), unintended information disclosure, or execution of arbitrary code. |
| --- |

**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** |  |
| Helix QAC | 2022.3 | **C++2012, C++2625** |  |
| LDRA tool suite | 9.7.1 | **41 S** | Fully Implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not declare or define a reserved identifier |

| **Noncompliant Code** |
| --- |
| The use of a macro in preprocessor conditional may cause a name clash. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| Avoiding using leading or trailing underscores renders the code compliant, thus prevents this issue; |
| #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):** Using reserved identifiers can lead to incorrect program operation. |
| --- |

**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 | 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. |
| LDRA tool suite | 9.7.1 | **86 S, 218 S, 219 S, 580 S** | Fully implemented |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Never qualify a reference type with const or volatile. C++ does not allow a programmer to change the value of a reference type. Thus reference type variables are already treated as constants. An additional constant/volatile qualifier will cause an undefined behavior |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a const-qualified reference to a char is formed instead of a reference to a const-qualified char. This results in undefined behavior. |
| #include <iostream>    **void** f(**char** c) {  **char** &**const** p = c;    p = 'p';    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| The const qualifier is removed to make the code compliant |
| #include <iostream>    **void** f(**char** c) {  **char** &p = c;    p = 'p';    std::cout << c << std::endl;  } |

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

| **Principles(s):** A const or volatile reference type may result in undefined behavior instead of a fatal diagnostic, causing unexpected values to be stored and leading to possible data integrity violations. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++ - DCL52** |  |
| Helix QAC | 2022.3 | **C++ 0014** |  |
| Klocwork | 2022.3 | **Cert.DCL.REF\_TYPE.CONST\_OR\_VOLATILE** |  |
| Parasoft C/C++ test | 2022.1 | **CERT\_CPP-DCL52-a** | Checks for:   * const-qualified reference types * modification of const-qualified reference types   Rule fully covered. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-003-CPP] | Do not write syntactically ambiguous declarations |

| **Noncompliant Code** |
| --- |
| This argument can be taken to declare an anonymous object and calling its single-argument converting constructor or interpreted as declaring an object named and default constructing it |
| #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 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):** Syntactically ambiguous declarations can lead to unexpected program execution. However, it is likely that rudimentary testing would uncover violations of this rule. |
| --- |

**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++1108, C++2510** |  |
| Klocwork | 2022.3 | **CERT.DCL.AMBIGUOUS\_DECL** |  |
| LDRA tool suite | 9.7.1 | **296 S** | Partially implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Overload allocation and deallocation functions as a pair in the same scope |

| **Noncompliant Code** |
| --- |
| An allocation function is overloaded at global scope. However, no deallocation function is 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** |
| --- |
| 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):** Mismatched usage of new and delete could lead to a denial-of-service attack. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.1 | **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** | [STD-006-CPP] | Avoid information leakage when passing a class object across a trust boundary The data passed must be verified to prevent information leakage. |

| **Noncompliant Code** |
| --- |
| Regardless of how the data is copied, padding bits used within the object may contain sensitive information. |
| #include <cstddef>    **struct** test {  **int** a;  **char** b;  **int** c;  };    // Safely copy bytes to user space  **extern** **int** copy\_to\_user(**void** \*dest, **void** \*src, std::**size\_t** size);    **void** do\_stuff(**void** \*usr\_buf) {    test arg{1, 2, 3};    copy\_to\_user(usr\_buf, &arg, **sizeof**(arg));  } |

| **Compliant Code** |
| --- |
| The structure data is serialized before being copied 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):** Padding bits might inadvertently contain sensitive data such as pointers to kernel data structures or passwords. A pointer to such a structure could be passed to other functions, causing information leakage. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++ - DCL55** |  |
| CodeSonar | 7.1p0 | **MISC.PADDING.POTB** | Padding Passed Across a Trust Boundary |
| 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 |
| Polyspace Bug Finder | R2022b | **CERT C++: DCL55-CPP** | Checks for information leakage due to structure padding (rule partially covered) |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Avoid cycles during initialization of static objects. If a function is reentered during the constant initialization of a static object inside that function, the behavior of the program is undefined. |

| **Noncompliant Code** |
| --- |
| This code block attempts to implement an efficient factorial function using caching |
| #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 function avoids initializing the static local array cache so it can determine whether or not each member of the array has already been assigned a value. |
| #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):** Recursively reentering a function during the initialization of one of its static objects can result in an attacker being able to cause a crash or denial of service. Indeterminately ordered dynamic initialization can lead to undefined behavior due to accessing an uninitialized object. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unliekly | 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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++1552, C++1554, C++1704** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **6 D** | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **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] | [STD-008-CPP] | Do not let exceptions escape from destructors or deallocation functions |

| **Noncompliant Code** |
| --- |
| Destructor may still throw an exception. Thus, it can still trigger an 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** |
| --- |
| In the event an exception is thrown, this function will catch and handle it. Meaning the destructor will behave in the same way whether there was an exception or not. |
| **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):** Attempting to throw exceptions from destructors or deallocation functions can result in undefined behavior, leading to resource leaks or denial-of-service attacks. |
| --- |

**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 |
| Klocwork | 2022.3 | **MISRA.DOTOR.THROW** |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | Do not modify the standard namespaces. Introducing new declarations in namespace std, except under exceptional circumstances, will result in an undefined behavior. |

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

| **Compliant Code** |
| --- |
| In this example, a specialization of std::plus is added to the std namespace, but the specialization depends on a user-defined type and meets the Standard Template Library requirements for the original template |
| #include <functional>  #include <iostream>  #include <string>    **class** MyString {    std::string data;    **public**:    MyString(**const** std::string &data) : data(data) {}    **const** std::string &get\_data() **const** { **return** data; }  };    **namespace** std {  **template** <>  **struct** plus<MyString> {    MyString operator()(**const** MyString &lhs, **const** MyString &rhs) **const** {  **return** lhs.get\_data() + rhs.get\_data();    }  };  }    **void** f() {    std::string s1("My String");    MyString s2(" + Your String");    std::plus<MyString> p;      std::cout << p(s1, s2).get\_data() << std::endl;  } |

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

| **Principles(s):** Altering the standard namespace can cause undefined behavior in the C++ standard library. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL58** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **LANG.STRUCT.DECL.SNM** | Modification of Standard Namespaces |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [**CERT C++: DCL58-CPP**](https://www.mathworks.com/help/bugfinder/ref/certcdcl58cpp.html) | Checks for modification of standard namespaces (rule fully covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.21 | [**V1061**](https://pvs-studio.com/en/docs/warnings/v1061/) |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Do not define an unnamed namespace in a header file. It can lead to surprising and unexpected results. |

| **Noncompliant Code** |
| --- |
| Variable v is defined in an unnamed namespace, causing each translation unit where it’s used to run a separate instance. Modifying it from multiple instances/places can cause the value of v to be unexpected. |
| // 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** |
| --- |
| Variable v is defined in only one translation unit but is externally visible to all translation units. |
| // 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):** Defining an unnamed namespace within a header file can cause data integrity violations and performance problems but is unlikely to go unnoticed with sufficient testing. One-definition rule violations result in undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **unnamed-namespace-header** | Fully checked |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **286 S, 512 S** | Fully implemented |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **unnamed-namespace-header** | Fully checked |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL59-a** | There shall be no unnamed namespaces in header files |

### 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 | Low | Unlikely | Low | P3 | L3 |
| STD-004-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-005-CPP | Low | Probable | Low | P6 | L2 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-008-CPP | Low | Likely | Medium | P6 | L2 |
| STD-009-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-010-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 | Encryption at rest is when the data is store on a medium in its encrypted form. It’s designed to prevent the attacker who gets access to the medium from reading and/or get any use of the data without also obtaining the encryption key. |
| Encryption at flight | Encryption at flight is when the data is encrypted during its transmission from one location to another. It may or may not be encrypted at rest or from the hard drive it is stored on. An example of encryption at flight is when data gets transferred from a web server to a client computer via https. |
| Encryption in use | Encryption in use is when the data actively being used by a user or program used is encrypted. Compromised data in use may occur when an attacker gets direct access to read memory locations. They may get the encryption key in the process, which will enable them to access the unencrypted version of the encrypted data at rest. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying the user is who they say they are. This is usually done by asking the user to supply a username and password, which is compared against credentials saved in a database. Once the user is confirmed the be who they say they are, access is granted. |
| Authorization | Authorization happens right after the user has been authenticated. It is the process of determining what level of access the user has within the system. Depending on the level of access, a user may have unrestricted access to the system (example: an administrator) or has access to only certain part of the system. For instance, an accountant may have access to the financial part of the system while being unable to run reports, which is typically reserved for an Operations Manager. |
| Accounting | Accounting is the process by which a user is being monitored while they navigate the system. Their actions are recorded and saved in a log. That way, it will be possible to determine what they were doing, which database were accessed, and what actions were taken among many other things. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 11/12/2022 | Security Principles and Coding Standards | Jean Petit |  |
| 1.2 | 12/04/2022 | Threat Levels and Tools | Jean Petit |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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