**Green Pace Developer: Jonathan Boeglin**



# 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 untrusted sources and external data such as command line, network, environmental, and user-controlled sources. This can eliminate many vulnerabilities. |
| 1. Heed Compiler Warnings | Always use the highest warning level possible when compiling and fix all errors that arise. Analysis tools can help with additional flaw detection. |
| 1. Architect and Design for Security Policies | Create software that implements security policies through its inherent architecture and design. If a system requires different amounts of security in different areas, separating those areas appropriately is key to good design. |
| 1. Keep It Simple | Smaller, less complex designs require less to secure properly, and errors are more likely to be caught and handled accurately. |
| 1. Default Deny | Grant privileges based on necessity. The normal state of the system should be that all usage is denied, and exceptions are granted based on what is needed to be accessed. |
| 1. Adhere to the Principle of Least Privilege | Processes should also only be granted access based on need. The access should only last as long as necessary and be terminated immediately after to prevent unwanted opportunities for those granted permissions. |
| 1. Sanitize Data Sent to Other Systems | Injection attacks can be used through complex subsystems such as command shells, relational databases, and commercial off-the-shelf components. All calling processes therefore need to sanitize any data being passed to these systems beforehand. |
| 1. Practice Defense in Depth | Using multiple security strategies, any security flaws, exploits, or areas not covered by a single system can be defended from attacks. Using different levels of security such as encryption and antivirus software can combine to form a stronger barrier. |
| 1. Use Effective Quality Assurance Techniques | Independent testing of security can often find vulnerabilities missed beforehand as assumptions are often made by those creating the system, often without realization. Usage of testing and code audits by third parties can help with dismissing these assumptions. |
| 1. Adopt a Secure Coding Standard | Use a secure coding standard that applies to the platform being developed for and the coding language being used. If there is none available, take time to develop one. |

### 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** | DCL53-CPP. Do not write syntactically ambiguous declarations |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Confusion between an expression statement or declaration can cause incorrect compilation. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an attempt is made to declare a local variable, w, of type Widget while executing the default constructor. However, this declaration is syntactically ambiguous where the code could be either a declaration of a function pointer accepting no arguments and returning a Widget or a declaration of a local variable of type Widget. The syntax used in this example defines the former instead of the latter. |
| #include <iostream>    struct Widget {    Widget() { std::cout << "Constructed" << std::endl; }  };    void f() {    Widget w();  } |

| **Compliant Code** |
| --- |
| This compliant solution shows two equally compliant ways to write the declaration. The first way is to elide the parentheses after the variable declaration, which ensures the syntax is that of a variable declaration instead of a function declaration. The second way is to use a braced-init-list to direct-initialize the local variable. |
| #include <iostream>    struct Widget {    Widget() { std::cout << "Constructed" << std::endl; }  };    void f() {    Widget w1; // Elide the parentheses    Widget w2{}; // Use direct initialization  } |

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

| **Principles(s):**  9: By reviewing code and having third-party input, mistakes can be easily caught.  10: This standard will help flesh out the whole security policy that should be followed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | **LANG.STRUCT.DECL.FNEST** | Nested Function Declaration |
| Polyspace Bug Finder | R2023a | [CERT C++: DCL53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl53cpp.html) | Checks for declarations that can be confused between:   * Function and object declaration * Unnamed object or function parameter declaration   Rule fully covered. |
| Clang | 3.9 | -Wvexing-parse |  |
| SonarQube C/C++ Plugin | 4.10 | [**S3468**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3468) |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **DCL52-CPP. Never qualify a reference type with const or volatile** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Can cause undefined behavior as C++ effectively treats all references as already qualified. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example correctly declares p to be a reference to a const-qualified char. The subsequent modification of p makes the program ill-formed. |
| #include <iostream>    void f(char c) {    const char &p = c;    p = 'p'; // Error: read-only variable is not assignable    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| This compliant solution removes the const qualifier. |
| #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):**  4: Avoidance of the addition of unnecessary qualifiers will keep the code simpler.  9: By reviewing code, using automation tools, and having third-party input mistakes can be easily caught.  10: This standard will help flesh out the whole security policy that should be followed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |
| Polyspace Bug Finder | R2023a | [CERT C++: DCL52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl52cpp.html) | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| Clang | 3.9 |  | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options. |
| SonarQube C/C++ Plugin | 4.10 | [**S3708**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3708) |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **STR51-CPP. Do not attempt to create a std::string from a null pointer** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Undefined behavior can result as it will dereference a null pointer. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist (or some other error occurs). |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>    void f() {    const char \*tmpPtrVal = std::getenv("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");    if (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):**  1: Making sure the pointer is not null is a type of validation.  5: Preventing use if the pointer is null stops undefined behavior.  9: By reviewing code, using automation tools, and having third-party input mistakes can be easily caught.  10: This standard will help flesh out the whole security policy that should be followed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **assert\_failure** |  |
| CodeSonar | 7.4p0 | **LANG.MEM.NPD** | Null Pointer Dereference |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-STR51-a** | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2023a | [CERT C++: STR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr51cpp.html) | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **STR50-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Insufficient space can lead to buffer-overflow issues. |

| **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):**  3: Having a simple way to prevent buffer issues can help with security design.  4: Using strings instead of chars and bufs reduces code and is simpler to understand.  9: By reviewing code, using automation tools, and having third-party input mistakes can be easily caught.  10: This standard will help flesh out the whole security policy that should be followed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **stream-input-char-array** | Partially checked + soundly supported |
| CodeSonar | 7.4p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-STR50-b**  **CERT\_CPP-STR50-c**  **CERT\_CPP-STR50-e**  **CERT\_CPP-STR50-f**  **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| Polyspace Bug Finder | R2023a | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation * Insufficient destination buffer size   Rule partially covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **MEM50-CPP. Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Using memory that has been freed creates dangling pointers and can cause undefined behavior. |

| **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):**  6: Termination should take place after use. Although this principle usually applies to not terminating privilege in a timely manner, doing so too soon can also cause security issues.  9: By reviewing code, using automation tools, and having third-party input mistakes can be easily caught.  10: This standard will help flesh out the whole security policy that should be followed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **dangling\_pointer\_use** |  |
| CodeSonar | 7.4p0 | **ALLOC.UAF** | Use after free |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |
| Polyspace Bug Finder | R2023a | [CERT C++: MEM50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem50cpp.html) | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **EXP50-CPP. Do not depend on the order of evaluation for side effects** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Evaluating a variable more than once at a time can cause undefined results. Assertions need to follow this pattern so the correct comparison is made. |

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

| **Compliant Code** |
| --- |
| These examples are independent of the order of evaluation of the operands and can each be interpreted in only one way. |
| void f(int i, const int \*b) {    ++i;    int a = i + b[i];    // ...  } |

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

| **Principles(s):**  4: Trying to combine too much code into a single line can become complex and hard for anyone reviewing it to understand. In this case, it also results in undefined behavior.  9: By reviewing code, using automation tools, and having third-party input mistakes can be easily caught.  10: This standard will help flesh out the whole security policy that should be followed. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | **P8** | **L2** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wunsequenced | Can detect simple violations of this rule where path-sensitive analysis is not required |
| CodeSonar | 7.4p0 | **LANG.STRUCT.SE.DEC**  **LANG.STRUCT.SE.INC** | Side Effects in Expression with Decrement Side Effects in Expression with Increment |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-EXP50-a**  **CERT\_CPP-EXP50-b**  **CERT\_CPP-EXP50-c**  **CERT\_CPP-EXP50-d**  **CERT\_CPP-EXP50-e**  **CERT\_CPP-EXP50-f** | The value of an expression shall be the same under any order of evaluation that the standard permits  Don't write code that depends on the order of evaluation of function arguments  Don't write code that depends on the order of evaluation of function designator and function arguments  Don't write code that depends on the order of evaluation of expression that involves a function call  Between sequence points an object shall have its stored value modified at most once by the evaluation of an expression  Don't write code that depends on the order of evaluation of function calls |
| Polyspace Bug Finder | R2023a | [CERT C++: EXP50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp50cpp.html) | Checks for situations where expression value depends on order of evaluation (rule fully covered). |

#### Coding Standard 7

| **Coding Standard** | **Label** | **ERR56-CPP. Guarantee exception safety** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | If an exception has a direct effect on the program when it is raised it can cause incorrect operation of the software. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows a flawed copy assignment operator. The implicit invariants of the class are that the array member is a valid (possibly null) pointer and that the nElems member stores the number of elements in the array pointed to by array. The function deallocates array and assigns the element counter, nElems, before allocating a new block of memory for the copy. As a result, if the new expression throws an exception, the function will have modified the state of both member variables in a way that violates the implicit invariants of the class. Consequently, such an object is in an indeterminate state and any operation on it, including its destruction, results in undefined behavior. |
| #include <cstring>    class IntArray {    int \*array;    std::size\_t nElems;  public:    // ...      ~IntArray() {      delete[] array;    }        IntArray(const IntArray& that); // nontrivial copy constructor    IntArray& operator=(const IntArray &rhs) {      if (this != &rhs) {        delete[] array;        array = nullptr;        nElems = rhs.nElems;        if (nElems) {          array = new int[nElems];          std::memcpy(array, rhs.array, nElems \* sizeof(\*array));        }      }      return \*this;    }      // ...  }; |

| **Compliant Code** |
| --- |
| In this compliant solution, the copy assignment operator provides the strong exception safety guarantee. The function allocates new storage for the copy before changing the state of the object. Only after the allocation succeeds does the function proceed to change the state of the object. In addition, by copying the array to the newly allocated storage before deallocating the existing array, the function avoids the test for self-assignment, which improves the performance of the code in the common case [Sutter 2004]. |
| #include <cstring>    class IntArray {    int \*array;    std::size\_t nElems;  public:    // ...      ~IntArray() {      delete[] array;    }      IntArray(const IntArray& that); // nontrivial copy constructor      IntArray& operator=(const IntArray &rhs) {      int \*tmp = nullptr;      if (rhs.nElems) {        tmp = new int[rhs.nElems];        std::memcpy(tmp, rhs.array, rhs.nElems \* sizeof(\*array));      }      delete[] array;      array = tmp;      nElems = rhs.nElems;      return \*this;    }      // ...  }; |

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

| **Principles(s):**  9: By reviewing code, using automation tools, and having third-party input mistakes can be easily caught. In this case, the proper coding also improves quality as it increases performance.  10: This standard will help flesh out the whole security policy that should be followed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | **ALLOC.LEAK** | Leak |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-ERR56-a CERT\_CPP-ERR56-b** | Always catch exceptions Do not leave 'catch' blocks empty |
| Polyspace Bug Finder | R2023a | [CERT C++: ERR56-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr56cpp.html) | Checks for exceptions violating class invariant (rule fully covered). |
| PVS-Studio | 7.26 | [**V565**](https://pvs-studio.com/en/docs/warnings/v565/)**,**[**V1023**](https://pvs-studio.com/en/docs/warnings/v1023/)**,** [**V5002**](https://pvs-studio.com/en/docs/warnings/v5002/) |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **ERR55-CPP. Honor exception specifications** |
| --- | --- | --- |
| **Exceptions** | STD-008-CPP | Not following exception specifications can lead to abnormal program termination. |

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

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

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

| **Principles(s):**  4: Use of unnecessary specifications can cause undefined behavior.  9: By reviewing code, using automation tools, and having third-party input mistakes can be easily caught.  10: This standard will help flesh out the whole security policy that should be followed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **unhandled-throw-noexcept** | Partially checked |
| CodeSonar | 7.4p0 | **LANG.STRUCT.EXCP.THROW** | Use of throw |
| Parasoft C/C++Test | 2023.1 | **CERT\_CPP-ERR55-a** | Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) |
| Polyspace Bug Finder | R2023a | [CERT C++: ERR55-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr55cpp.html) | Checks for noexcept functions exiting with exception (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **DCL58-CPP. Do not modify the standard namespaces** |
| --- | --- | --- |
| **Data Type** | STD-009-CPP | Causes undefined behavior as declarations are already defined. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, 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):**  4: Trying to modify an already established namespace causes undefined behavior, and creating a new namespace with your own parameters is easier.  9: By reviewing code, using automation tools, and having third-party input mistakes can be easily caught.  10: This standard will help flesh out the whole security policy that should be followed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | **LANG.STRUCT.DECL.SNM** | Modification of Standard Namespaces |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-DCL58-a** | Do not modify the standard namespaces 'std' and 'posix' |
| Polyspace Bug Finder | R2023a | [CERT C++: DCL58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl58cpp.html) | Checks for modification of standard namespaces (rule fully covered) |
| SonarQube C/C++ Plugin | 4.10 | [**S3470**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3470) |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **MEM52-CPP. Detect and handle memory allocation errors** |
| --- | --- | --- |
| **Memory Protection** | STD-010-CPP | Can cause abnormal program termination if the pointer is not checked properly before referencing. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an array of int is created using ::operator new[](std::size\_t) and the results of the allocation are not checked. The function is marked as noexcept, so the caller assumes this function does not throw any exceptions. Because ::operator new[](std::size\_t) can throw an exception if the allocation fails, it could lead to abnormal termination of the program. |
| #include <cstring>    void f(const int \*array, std::size\_t size) noexcept {    int \*copy = new int[size];    std::memcpy(copy, array, size \* sizeof(\*copy));    // ...    delete [] copy;  } |

| **Compliant Code** |
| --- |
| When using std::nothrow, the new operator returns either a null pointer or a pointer to the allocated space. Always test the returned pointer to ensure it is not nullptr before referencing the pointer. This compliant solution handles the error condition appropriately when the returned pointer is nullptr. |
| #include <cstring>  #include <new>    void f(const int \*array, std::size\_t size) noexcept {    int \*copy = new (std::nothrow) int[size];    if (!copy) {      // Handle error      return;    }    std::memcpy(copy, array, size \* sizeof(\*copy));    // ...    delete [] copy;  } |

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

| **Principles(s):**  1: Checking for null can prevent undefined behaviors and abnormal terminations.  3: Preventing abnormal terminations averts corruption issues.  9: By reviewing code, using automation tools, and having third-party input mistakes can be easily caught.  10: This standard will help flesh out the whole security policy that should be followed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | **CHECKED\_RETURN** | Finds inconsistencies in how function call return values are handled |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-MEM52-a**  **CERT\_CPP-MEM52-b** | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| Polyspace Bug Finder | R2023a | [CERT C++: MEM52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem52cpp.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |
| PVS-Studio | 7.26 | [**V522**](https://pvs-studio.com/en/docs/warnings/v522/)**,** [**V668**](https://pvs-studio.com/en/docs/warnings/v668/) |  |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

Automation of security practices can be inserted into the DevOps process during pre-production and production. Security needs to be planned for, built properly, tested, and maintained, so adding secure procedures to the planning, design, building, testing, and maintenance stages of the process are the best places for modification. This changes the production process from just maintenance and release of updates to also monitoring for threats and responding to any attacks. Maintenance includes stabilization after attacks, and rollout of updates include live testing of security measures. On the pre-production side the steps stay mostly the same, with the addition of security planning, design, and integration during the appropriate times. Testing and verification will require the creation of new tests for security coding.

Regarding how automation tools will help enforce the standards within this policy, various tools listed within each section can be used to check the code for breaches of each of the specific standards. This will take place during the building and verification and testing phases of pre-production. In addition, any updates to the tools that catch new issues can be run during production on the code during maintenance and any necessary updates can be added to the cycle.

### 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 | Low | Unlikely | Medium | P2 | L3 |
| STD-002-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-007-CPP | High | Likely | High | P9 | L2 |
| STD-008-CPP | Low | Likely | Low | P9 | L2 |
| STD-009-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-010-CPP | High | Likely | Medium | P18 | L1 |

### 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 | Referring to the protection of data that is being stored or is not actively being used, this applies as sensitive data needs safe storage. Varying levels of security need to be implemented to protect data based on its volatility. |
| Encryption at flight | This level of protection is for data that is currently being sent between destinations, such as emails, various app messages, or other communication done through public means. This applies to security for possible mishandling of data, such as data breaches or leaks. |
| Encryption in use | Encryption in use is the most important, as the data it protects is the most likely to be tampered with. It is used for protection while data is actively being accessed and manipulated, such as reading, writing to, or creating a file. An application example of this policy would be protection from attempts to steal user information, like passwords. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is the process of a user giving details on themselves to be applied in the future for verification. Authentication is used to confirm people accessing a program are genuine with user logins. The addition of new users is also handled by authentication of details provided, such as email addresses by way of verification emails. |
| Authorization | Authorization protects data from the wrong people by user access levels. This allows users to access different parts of a network or program depending on the privileges given to them and prevents them from seeing the rest. An example of the application of this policy would be the restriction of accessing and making changes to a website’s database. Only those with the proper credentials would be allowed to do so, preventing unwanted tampering by those that have access to the site, but not the database itself. |
| Accounting | This part of the strategy helps prevent future issues by keeping a record of past events. This is done through recording of users through their data usage-how they used it, what files they accessed, and the length of time they were connected. This policy can be applied when a security breach occurs, as all users should be accounted for, and the offender can be determined from the access data gathered during the attack. |

**\***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 | 09/16/2023 | Filled Security Principles/Standards | Jonathan Boeglin |  |
| 1.2 | 10/08/2023 | Filled in rest of document | Jonathan Boeglin |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

### Citations

\*All standard description, compliant and non-compliant, threat level, and automation sections taken directly from the SEI CERT C++ Coding Standard [website](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682.*).\*

Admin et al. (2023, September 5). *Sei cert C++ coding standard*. SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence. https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682

Dinic, M. (2023, August 16). Encryption: Data at rest, data in motion and data in use. Jatheon Technologies Inc. https://jatheon.com/blog/data-at-rest-data-in-motion-data-in-use/

Fortinet, Inc. (2023). What is AAA Security?. Fortinet. https://www.fortinet.com/resources/cyberglossary/aaa-security