**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](#_Toc52464059) 4

[Coding Standard 2 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 12](#_Toc52464063)

[Coding Standard 6 14](#_Toc52464064)

[Coding Standard 7 16](#_Toc52464065)

[Coding Standard 8 18](#_Toc52464066)

[Coding Standard 9 20](#_Toc52464067)

[Coding Standard 10 22](#_Toc52464068)

[Defense-in-Depth Illustration 24](#_Toc52464069)

[Project One 24](#_Toc52464070)

[1. Revise the C/C++ Standards 24](#_Toc52464071)

[2. Risk Assessment](#_Toc52464072) 24

[3. Automated Detection](#_Toc52464073) 24

[4. Automation 24](#_Toc52464074)

[5. Summary of Risk Assessments 25](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 25](#_Toc52464076)

[7. Map the Principles 26](#_Toc52464077)

[Audit Controls and Management 27](#_Toc52464078)

[Enforcement 27](#_Toc52464079)

[Exceptions Process 27](#_Toc52464080)

[Distribution 28](#_Toc52464081)

[Policy Change Control 28](#_Toc52464082)

[Policy Version History 28](#_Toc52464083)

[Appendix A Lookups 28](#_Toc52464084)

[Approved C/C++ Language Acronyms 28](#_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 | Validated input from all untrusted data sources to prevent and eliminate most software vulnerabilities. All external data sources shall be considered unsafe including command line arguments, network interfaces, environmental variables, and user-controlled files. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available for your compiler 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 | Design and build software that enforces and heeds security policies. An example might be using private methods or classes. |
| 1. Keep It Simple | Code design shall be kept as small and simple as possible; simpler the code, the less likelihood for errors and oversights. |
| 1. Default Deny | Access decisions shall be based on permission rather than exclusion. Users shall not obtain access to a system without proper user credentials. |
| 1. Adhere to the Principle of Least Privilege | Each process shall execute with the least amount of privilege needed to perform that function. Least privilege pertains to users, programs, processes, networks, databases, or entire systems. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Data shall be sanitized prior to passing it to another system/subsystem. An example is removing SQL injection from input prior to passing it as an argument to another method to prevent attackers from invoking unused functionality in these components through the use SQL or other injection attacks. |
| 1. Practice Defense in Depth | Systems shall be protected in layers. Risk shall be managed with multiple strategies to prevent exploits or vulnerabilities where if one layer turns out to be inadequate, another layer can prevent a security flaw from being exploitable. |
| 1. Use Effective Quality Assurance Techniques | Vulnerabilities can be identified and eliminated using good quality assurance techniques such as third-party security reviews. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for your target development language and platform. |

### 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] | Obey the one-definition rule. |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule>

| **Noncompliant Code** |
| --- |
| Two different translation units define a class of the same name with differing definitions. |
| // a.cpp  struct S {  **int** a;  };    // b.cpp  class S {  public:  **int** a;  }; |

| **Compliant Code** |
| --- |
| Use of a header file to introduce the object into both translation units. |
| // S.h  struct S {  **int** a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):** 1: validate input data, 4: keep it simple. Validating data input and keeping the integers and buffers used in a simple and clear design reduces data type errors occurring |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL60** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | |  |  | | --- | --- | |  | **LANG.STRUCT.DEF.FDH LANG.STRUCT.DEF.ODH** | | Function defined in header file Object defined in header file |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | R2022b | [CERT C++: DCL60-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl60cpp.html) | Checks for inline constraints not respected (rule partially covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not read uninitialized variables. |

Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory

| **Noncompliant Code** |
| --- |
| Uninitialized local variable is evaluated as part of an expression to print its value, resulting in undefined behavior. |
| #include <iostream>    void f() {  **int** i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| Object is initialized prior to printing its value. |
| #include <iostream>    void f() {  **int** i = 0;    std::cout << i;  } |

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

| **Principles(s):** 1: validate input data, 4: keep it simple; validating the data has been initialized can be easily seen by keeping the code simple and clear. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **uninitialized-read** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | |  |  | | --- | --- | |  | **LANG.STRUCT.RPL LANG.MEM.UVAR** | | Return pointer to local Uninitialized variable |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++2726, C++2727, C++2728, C++2961, C++2962, C++2963, C++2966, C++2967, C++2968, C++2971, C++2972, C++2973, C++2976, C++2977, C++2978** |  |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: EXP53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp53cpp.html) | Checks for:   * Non-initialized variable * Non-initialized pointer   Rule partially covered. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not attempt to create a std::string from a null pointer. |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR51-CPP.+Do+not+attempt+to+create+a+std%3A%3Astring+from+a+null+pointer>

| **Noncompliant Code** |
| --- |
| 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()) {      // ...    }  }x |

| **Compliant Code** |
| --- |
| 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: Validate input data with appropriate string function; 2: Heed compiler warning of overflow; 7: sanitize string data to prevent string attacks |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **assert\_failure** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **LANG.MEM.NPD** | Null Pointer Dereference |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-STR51-a** | Avoid null pointer dereferencing |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022b | [CERT C++: STR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr51cpp.html) | Checks for string operations on null pointer (rule fully covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL injection. |

Java version source (for reference): <https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection>

| **Noncompliant Code** |
| --- |
| Without precautions, the untrusted data may maliciously alter the query. |
| uName = getRequestString("username");  uPass = getRequestString("userpassword");  sql = “SELECT \* FROM Users WHERE Name = " + uName + " AND Pass = " +  uPass + ” |

| **Compliant Code** |
| --- |
| The primary means of preventing SQL injection are sanitization and validation, which are typically implemented as parameterized queries and stored procedures. |
| PreparedStatement pStmt = PreparedStatement();  std::cin >> username;  std::cin >> userpassword;  sql = “SELECT \* FROM Users WHERE Name = %s AND Pass = %s;”, username,  userpassword}; |

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

| **Principles(s):** 1: validate input to ensure query parameterization; 3: Design code to prevent SQL injections; 7: Sanitize data prior to passing it to another system by removing SQL injection from input prior to passing it as an argument to another method |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [The Checker Framework](https://wiki.sei.cmu.edu/confluence/display/java/The+Checker+Framework) | 2.1.3 | **Tainting Checker** | Trust and security errors (see Chapter 8) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **JAVA.IO.INJ.SQL** | SQL Injection (Java) |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2022.1 | **CERT.IDS00.TDSQL** | |  |  | | --- | --- | |  | Protect against SQL injection | |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 6.7 | |  |  | | --- | --- | |  | [**S2077**](https://rules.sonarsource.com/java/RSPEC-2077)  [**S3649**](https://rules.sonarsource.com/java/RSPEC-3649) | | [Executing SQL queries is security-sensitive](https://rules.sonarsource.com/java/RSPEC-2077)  [SQL queries should not be vulnerable to injection attacks](https://rules.sonarsource.com/java/RSPEC-3649) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory>

| **Noncompliant Code** |
| --- |
| s is dereferenced after it has been deallocated. If this access results in a write-after-free, this can be exploited to run arbitrary code with the permissions of the vulnerable process. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| 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):** 2: heed compiler warnings to detect if data is free or not; 3: remember to release from memory when data is done with task; 10: develop a coding standard with a focus on deallocating dynamically allocated memory after it is no long required. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | |  |  | | --- | --- | |  | 3.9 | | |  |  | | --- | --- | |  | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | | Checked by clang-tidy but does not catch all violations of this rule. |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | v7.5.0 | **USE\_AFTER\_FREE** | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [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. | |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | Use a static assertion to test the value of a constant expression |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression>

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

| **Compliant Code** |
| --- |
| For constant expressions, a preprocessor conditional statement may be used. |
| struct timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** 1: validate input data by use of assertions; 3: design code that involves writing secure memory allocation routines; 9: Use third-party review for QA testing of assertions in unit testing; 10: adopt a standard coding scheme with a focus on white box testing through the use of unit testing. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not abruptly terminate the program |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR50-CPP.+Do+not+abruptly+terminate+the+program>

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

| **Compliant Code** |
| --- |
| f() handles all exceptions thrown by throwing\_func() and does not rethrow. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.    try {      throwing\_func();    } catch (...) {      // Handle error    }  } |

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

| **Principles(s):** 2: heed compiler warning about abruptly terminating a program; 10: adopt a secure coding standard that focuses on preventing abruptly terminating programs. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stdlib-use** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | |  |  | | --- | --- | | 0 | **BADFUNC.ABORT BADFUNC.EXIT** | | |  |  | | --- | --- | |  | Use of abort Use of exit | |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++5014** |  |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: ERR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr50cpp.html) | Checks for implicit call to terminate() function (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-008-CPP] | Write constructor member initializers in the canonical order |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP53-CPP.+Write+constructor+member+initializers+in+the+canonical+order>

| **Noncompliant Code** |
| --- |
| The member initializer list for C::C() attempts to initialize someVal first and then to initialize dependsOnSomeVal to a value dependent on someVal. Because the declaration order of the member variables does not match the member initializer order, attempting to read the value of someVal results in an unspecified value being stored into dependsOnSomeVal. |
| class C {  **int** dependsOnSomeVal;  **int** someVal;    public:    C(**int** val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| Change the declaration order of the class member variables so that the dependency can be ordered properly in the constructor's member initializer list. |
| class C {  **int** someVal;  **int** dependsOnSomeVal;    public:    C(**int** val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

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

| **Principles(s):** 3: build software with a focus on OOP using private methods and classes; 4: using OOP keeps code compartmentalized and easy to read; 10: adopting a secure coding standard is that is consistent between development is one key aspect of OOP. |
| --- |

**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 | **initializer-list-order** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **LANG.STRUCT.INIT.OOMI** | Out of Order Member Initializers |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-OOP53-a** | List members in an initialization list in the order in which they are declared |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: OOP53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop53cpp.html) | Checks for members not initialized in canonical order (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-009-CPP] | Use valid iterator ranges |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR53-CPP.+Use+valid+iterator+ranges>

| **Noncompliant Code** |
| --- |
| On each iteration of its internal loop, std::for\_each() compares the first iterator (after incrementing it) with the second for equality; as long as they are not equal, it will continue to increment the first iterator. Incrementing the iterator representing the past-the-end element of the range results in undefined behavior. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<**int**> &c) {    std::for\_each(c.end(), c.begin(), [](**int** i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| The iterator values passed to std::for\_each() are passed in the proper order. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<**int**> &c) {    std::for\_each(c.begin(), c.end(), [](**int** i) { std::cout << i; });  } |

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

| **Principles(s):** 1: validating input data to pass data in the proper order to prevent errors; 2: heed compiler warnings to detect flaws in iterator ranges; 4: simple code makes it easy to detect flaws; 9: using third-party review tools for QA helps detect flaws in code; 10: a standard coding scheme across development makes correct errors easy. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **LANG.MEM.BO** | Buffer Overrun |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-CTR53-a**  **CERT\_CPP-CTR53-b** | Do not use an iterator range that isn't really a range Do not compare iterators from different containers |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022b | [CERT C++: CTR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcctr53cpp.html) | Checks for invalid iterator range (rule partially covered). |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.21 | [**V539**](https://pvs-studio.com/en/docs/warnings/v539/), [**V662**](https://pvs-studio.com/en/docs/warnings/v662/), [**V789**](https://pvs-studio.com/en/docs/warnings/v789/) |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | [STD-010-CPP] | Do not access an object outside of its lifetime |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP54-CPP.+Do+not+access+an+object+outside+of+its+lifetime>

| **Noncompliant Code** |
| --- |
| A pointer to an object is used to call a non-static member function of the object prior to the beginning of the pointer's lifetime, resulting in undefined behavior. |
| struct S {    void mem\_fn();  };    void f() {    S \*s;    s->mem\_fn();  } |

| **Compliant Code** |
| --- |
| Storage is obtained for the pointer prior to calling S::mem\_fn(). |
| struct S {    void mem\_fn();  };    void f() {    S \*s = new S;    s->mem\_fn();    delete s;  } |

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

| **Principles(s):** 1: validating input data prevents code from resulting in undefined behavior due to pointer errors; 2: heed compiler warnings help detect errors in code; 4: simple code makes detected errors easy to fix; 9: using third-party review tools makes identifying and eliminating vulnerabilities easy; 10: a secure standard of coding helps prevent expression errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **return-reference-local dangling\_pointer\_use** | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | **-Wdangling-initializer-list** | Catches some lifetime issues related to incorrect use of std::initializer\_list<> |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **42 D, 53 D, 77 D, 1 J, 71 S, 565 S** | Partially implemented |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **return-reference-local** | Partially checked |

### 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 starts in Build, by automating manual process. Then in the Verify and Test phase of pre-production, automation needs to be implemented to enforce policies by automating virtualized container deployment. Implement automated security tests and regression tests in QA to automatically check the dependencies used by an application with a predefined and updated list of dependency vulnerabilities. This can greatly reduce the number of potential risks when moving forward in the production phase.

In the Monitor and Detect phase of production, the logging systems should also be automated. This is where notifications and logs should be automatically generated should the product not adhere to the guidelines outlined in this document with detailed logs labeled for future action. Embed dynamic application security testing into SDLC to look for vulnerabilities in real time. Use OWASP Dependency-Check to check code dependency vulnerabilities. The system should also be automated to mitigate or terminate attempts at breaching the guidelines.

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

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest refers to encryption of data that is at rest (not used) using full-desk encryption at the server level. Also, use database encryption using MySQL Server, and provide a backup strategy. For example, if there is a database full of usernames, passwords, and other sensitive information that isn’t in constant use by the system, it is important to ensure that data is encrypted and secure. |
| Encryption at flight | Encryption in flight is the process of encrypting data that is incoming or outgoing (in transit) from the system. An example would be sending and receiving emails. To do encryption at flight, it is important to keep software up to date, secure libraries, use Public Key infrastructure for end-to-end protection on message bodies or attachments, use Managed File Transfer or SSH with expiration date on the link, password access, utilize data leak prevention mechanisms built into cloud services. |
| Encryption in use | Encryption in use is the practice of encryption and security of data currently being accessed by a user. This means the data needs to have a secure, encrypted environment. Two examples of this are password verification and two-step verification. The use of hashing functions to encrypt the and decrypt data to read, translate, and verify the information is an example of encryption in use. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying the identity of the user trying to access the system. There needs to be a username and password for each user logging into the system that can be verified through the system. This should be done through an encrypted database of usernames and passwords. There should also be another form of authentication, such as one-time passwords sent to trusted devices or emails, to verify point of contact. |
| Authorization | Authorization is the process of checking and granting permissions to a user after they have successfully been authenticated into the system. For example, administrators have permissions to make changes to the database, add new users, run queries, and perform system-level commands. General users should not be able to access these permissions and only have access to specific files and programs with the absolute minimum privileges to view and do things. |
| Accounting | Accounting is the process of logging and accounting for the digital well being of the system and its data. A log file is an example of this; a log file is a record of the activity on the system performed by which users, such as what data is accessed and by who. This can be used to track unauthorized access or attacks to the system by allowing developers to understand who the attacker was and where the attack happened. This can then be used to lead to finding a mitigation strategy and create a roadmap to best protect against future attacks. |

**\***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 | 12/04/2022 | Milestone Three Coding Standards | Anthony Lee |  |
| 1.2 | 12/04/2022 | Project One | Anthony Lee |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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