**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 | Proper validation of input data is essential to prevent security vulnerabilities such as SQL injection and buffer overflows. Every piece of data received from external sources should be verified and sanitized to ensure security.  **Example**: Input validation prevents buffer overflow vulnerabilities, such as the OpenSSL Heartbleed vulnerability. |
| 1. Heed Compiler Warnings | Compiler warnings and static analysis tool reports provide valuable insights into potential security risks. Addressing these warnings helps prevent vulnerabilities from reaching production. **Example**: Addressing compiler warnings prevented vulnerabilities such as integer overflows that led to security bypasses. |
| 1. Architect and Design for Security Policies | Security must be an integral part of the software architecture and design. It involves incorporating security controls, defining policies, and ensuring compliance with regulatory standards.  **Example**: The Sony PlayStation Network breach was caused by SQL Injection, which could have been prevented by secure design. |
| 1. Keep It Simple | Complex designs often introduce unintended vulnerabilities. By keeping the design and implementation simple, the likelihood of introducing security flaws is minimized.  **Example**: Overly complex exception handling led to undefined behavior in NASA’s Mars Climate Orbiter. |
| 1. Default Deny | A default deny approach ensures that access to resources is restricted unless explicitly permitted. This principle reduces the attack surface by limiting unnecessary access.  **Example**: Restricting access prevents unauthorized execution of malicious code. |
| 1. Adhere to the Principle of Least Privilege | Granting users and processes only the permissions they need to perform their tasks helps mitigate the risk of unauthorized actions and limit potential damage.  **Example**: Implementing RBAC ensures users only have necessary privileges, reducing the impact of potential breaches. |
| 1. Sanitize Data Sent to Other Systems | Ensuring data sent to external systems is properly sanitized prevents unintended execution of malicious code and maintains data integrity.  **Example**: The Log4Shell vulnerability was caused by improper string handling. |
| 1. Practice Defense in Depth | Implementing multiple layers of security controls provides redundancy and comprehensive protection against potential threats.  **Example**: Combining multiple security controls reduces the likelihood of a successful exploit. |
| 1. Use Effective Quality Assurance Techniques | Comprehensive quality assurance, including security testing and code reviews, helps identify vulnerabilities throughout the development lifecycle.  **Example**: Early detection of vulnerabilities using automated tools like Fortify and Coverity improves security posture. |
| 1. Adopt a Secure Coding Standard | Following established coding standards, such as OWASP and CERT guidelines, helps developers avoid common security pitfalls and write secure code.  **Example**: Following OWASP secure coding principles minimizes common vulnerabilities. |

### 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 Handling** | [STD-001-CPP] | Range Verification  Rationale: Ensuring valid range checking prevents out-of-bounds errors and memory corruption. |

| **Noncompliant Code** |
| --- |
| The following code accesses an array index out of bounds |
| Int main(){  int arr[5] = {1, 2, 3, 4, 5};  // Trying to access an index beyond the array's range  int index = 10;  std::cout << arr[index] << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| The compliant code checks the index range before accessing the array. |
| Int main(){  int arr[5] = {1, 2, 3, 4, 5};  // Trying to access an index beyond the array's range  int index = 10;  if (index >= 0 && index < 5) {  std::cout << arr[index] << std::endl;  } else {  std::cout << "Invalid index!" << std::endl;  }  Return 0;  } |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL60 | - |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| LDRA tool suite | 9.7.1 | 286 S, 287 S | Fully Implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not read uninitialized memory  Source:  https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory |

| **Noncompliant Code** |
| --- |
| An uninitialized local variable is used in an expression to print its value, leading to undefined behavior because its content is indeterminate. Since local variables in C++ are not automatically initialized, accessing them before assigning a value can result in unpredictable program behavior, including crashes or unexpected outputs. |
| #include <iostream>  void f()  {  **int** i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| The object is explicitly initialized before being used in an expression, ensuring it contains a defined value. This prevents undefined behavior and guarantees that the program produces consistent and predictable results when accessing or printing the variable. |
| #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 10: Adopt a Secure Coding Standard** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | uninitialized-read | Partially checked |
| Helix QAC | 2021.2 | 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 | - |
| LDRA tool suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially implemented |
| Polyspace Bug Finder | R2021a | CERT C++: EXP53-CPP | 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 initialized using the return value of std::getenv(). However, if the specified environment variable is not found, std::getenv() returns a null pointer. Attempting to create a std::string from this null pointer results in undefined behavior, potentially causing program crashes or unexpected outcomes. |
| #include <cstdlib>  #include <string>  void f() {  std::string tmp(std::getenv("TMP")); // Potential undefined behavior  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| Before constructing the std::string object, the return value of std::getenv() is validated to ensure it is not a null pointer. If it is null, an appropriate fallback value is used, preventing undefined behavior and ensuring program stability. |
| #include <cstdlib>  #include <string>  void f() {  // Retrieve the environment variable  const char \*tmpPtrVal = std::getenv("TMP");  // Ensure safe conversion to std::string  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  if (!tmp.empty()) {  // Perform operations if TMP is not empty  // ...  }  } |

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

| **Principles(s):** 2: Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Assert\_failure | - |
| Helix QAC | 2021.2 | C++4770, C++4771, C++4772, C++4773, C++4774 | - |
| Klocwork | 2021.1 | NPD.CHECK.CALL.MIGHT NPD.CHECK.CALL.MUST NPD.CHECK.MIGHT NPD.CHECK.MUST NPD.CONST.CALL NPD.CONST.DEREF NPD.FUNC.CALL.MIGHT NPD.FUNC.CALL.MUST NPD.FUNC.MIGHT NPD.FUNC.MUST NPD.GEN.CALL.MIGHT NPD.GEN.CALL.MUST NPD.GEN.MIGHT NPD.GEN.MUST RNPD.CALL RNPD.DEREF | - |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL injection  Source  https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection |

| **Noncompliant Code** |
| --- |
| If proper safeguards are not in place, untrusted data can manipulate the query, potentially leading to SQL injection attacks, unauthorized data access, or database compromise. |
| uName = getRequestString("username");  uPass = getRequestString("userpassword");  sql = “SELECT \* FROM Users WHERE Name = " + uName + " AND Pass = " + uPass + ”; |

| **Compliant Code** |
| --- |
| The most effective way to prevent SQL injection is through input sanitization and validation, primarily achieved by using parameterized queries and stored procedures. These methods ensure that user input is treated as data rather than executable SQL code, preventing malicious alterations to queries. |
| 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 Data 7: Sanitize Data Sent to Other Systems 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATEDFB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| Fortify | 1.0 | HTTP\_Response\_Splitting SQL\_Injection\_\_Persistence SQL\_Injection | Implemented |
| Parasoft Jtest | 2021.1 | CERT.IDS00.TDSQL | Protect against SQL injection |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| After being deallocated, s is still referenced, leading to a use-after-free scenario. If the freed memory is overwritten and accessed improperly, it can result in a write-after-free vulnerability, potentially allowing an attacker to execute arbitrary code with the same privileges as the compromised process. |
| #include <memory>  struct S {  void f();  };  void g() noexcept(false) {  std::unique\_ptr<S> s = std::make\_unique<S>();  // ...  s->f(); // No need for manual deletion  } |

| **Compliant Code** |
| --- |
| The dynamically allocated memory remains allocated until it is no longer needed, ensuring that it is not prematurely deallocated. This prevents use-after-free errors and enhances memory safety by ensuring that the object remains accessible while in use. |
| #include <new>  struct S {  void f();  };  void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f(); // Undefined behavior: Accessing a deleted object  } |

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

| **Principles(s):** 2: Heed Compiler Warnings 5: Default Deny 6: Adhere to the Principle of Least Privilege 9: Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| 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 | 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 |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| Parasoft Insure++ | - | - | Runtime detection |

#### 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 |

| **Noncompliant Code** |
| --- |
| Utilizes the assert() macro to verify a critical property of a memory-mapped structure, ensuring that the code functions as intended and preventing potential issues related to incorrect assumptions about memory layout. |
| #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 directive can be used to validate assumptions at compile time. This ensures that the expected conditions are met before compilation proceeds, preventing potential runtime errors and enforcing memory layout constraints. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  // Ensures the structure has no unexpected padding  #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):** 2: Heed Compiler Warnings 10: Adopt a Secure Coding Standard |
| --- |

**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-DCL03 | - |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| The function f(), registered as an exit handler using std::at\_exit(), may trigger a call to std::terminate() if throwing\_func() throws an exception. Since exit handlers execute during program termination, any uncaught exception in f() can lead to abrupt termination, bypassing normal cleanup procedures. |
| #include <cstdlib>  void throwing\_func() noexcept(false);  void f() {  // Not invoked by the program except as an exit handler.  throwing\_func();  }  int main() {  // Register f() as an exit handler  if (0 != std::atexit(f)) {  // Handle error if registration fails  }    // ...  } |

| **Compliant Code** |
| --- |
| The function f() safely catches all exceptions thrown by throwing\_func() without rethrowing them. This ensures that unexpected exceptions do not propagate, preventing std::terminate() from being invoked and allowing for a controlled program termination. |
| #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 to prevent std::terminate()  }  }  int main() {  if (0 != std::atexit(f)) {  // Handle error if registration fails  }    // ...  } |

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

| **Principles(s):** 9: Use Effective Quality Assurance Techniques 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | BADFUNC.ABORT BADFUNC.EXIT | Use of abort Use of exit |
| Klocwork | 2021.1 | MISRA.CATCH.ALL CERT.ERR.ABRUPT\_TERM | - |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| Polyspace Bug Finder | R2021a | CERT C++: ERR50-CPP | 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 |

| **Noncompliant Code** |
| --- |
| In the constructor C::C(), the member initializer list initializes someVal first, followed by dependsOnSomeVal, which relies on someVal for its value. However, since the declaration order of member variables in the class does not match the order in the initializer list, dependsOnSomeVal may be assigned an unspecified or incorrect value, leading to undefined behavior. |
| class C {  int dependsOnSomeVal;  int someVal;  public:  C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| Rearrange the declaration of the class member variables to ensure they are initialized in the correct sequence within 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):** 4: Keep It Simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | initializer-list-order | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP53 | - |
| LDRA tool suite | 9.7.1 | 206 S | Fully implemented |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-OOP53-a | List members in an initialization list in the order in which they are declared |

#### Coding Standard 9

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

| **Noncompliant Code** |
| --- |
| During each iteration of its internal loop, std::for\_each() increments the first iterator and compares it with the second. The loop continues as long as the two iterators are not equal. However, if the first iterator is incremented beyond the past-the-end element of the range, it leads to undefined behavior, as accessing an invalid memory location is not permitted. |
| #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 iterators provided to std::for\_each() are arranged in the correct order, ensuring that iteration starts from the beginning of the range and progresses sequentially to the end, preventing undefined behavior. |
| #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):** 3: Architect and Design for Security Policies 4: Keep It Simple 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | overflow\_upon\_dereference | - |
| Helix QAC | 2021.2 | C++3802 | - |
| Parasoft C/C++test | 2021.1 | 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 |
| PVS-Studio | 7.14 | V539, V662, V789 | - |

#### Coding Standard 10

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

| **Noncompliant Code** |
| --- |
| A pointer to an object is used to invoke a non-static member function before the object has been properly initialized. Since the pointer does not point to a valid instance at that moment, attempting to access the object leads to undefined behavior, which may cause crashes, memory corruption, or unexpected results. |
| struct S { void mem\_fn(); }; void f() { S \*s; s->mem\_fn(); } |

| **Compliant Code** |
| --- |
| Memory for the pointer is allocated before invoking S::mem\_fn(), ensuring that the object exists and is properly initialized. This prevents undefined behavior caused by accessing an object before its lifetime begins. |
| 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):** 2: Heed Compiler Warnings 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | return-reference-local dangling\_pointer\_use | Partially checked |
| Clang | 3.9 | -Wdangling-initializer-list | Catches some lifetime issues related to incorrect use of std::initializer\_list<> |
| CodeSonar | 6.1p0 | IO.UAC ALLOC.UAF | Use after close Use after free |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-EXP54-a CERT\_CPP-EXP54-b CERT\_CPP-EXP54-c | Do not use resources that have been freed The address of an object with automatic storage shall not be returned from a function The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist |

### 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-002-CPP | High | Probable | Medium | High (12) | 1 |
| STD-003-CPP | High | Likely | Medium | High (18) | 1 |
| STD-004-CPP | High | Probable | Medium | High (12) | 1 |
| STD-005-CPP | High | Likely | Medium | High (18) | 1 |
| STD-010-CPP | High | Probable | High | Medium (6) | 2 |
| STD-006-CLG | Low | Unlikely | High | Low (1) | 3 |
| STD-007-CPP | Low | Probable | Medium | Low (4) | 3 |
| STD-008-CPP | Medium | Unlikely | Medium | Low (4) | 3 |
| STD-001-CPP | High | Unlikely | High | Low (3) | 3 |
| STD-002-CPP | High | Probable | Medium | High (12) | 1 |
| STD-003-CPP | High | Likely | Medium | High (18) | 1 |
| STD-004-CPP | High | Probable | Medium | High (12) | 1 |
| STD-005-CPP | High | Likely | Medium | High (18) | 1 |
| STD-010-CPP | High | Probable | High | Medium (6) | 2 |

### 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 at Rest | Protects stored data across various devices such as hard drives, mobile devices, computers, and cloud storage. Security measures include disk encryption, secure storage solutions, and encryption tools designed to safeguard data from unauthorized access while at rest. |
| Encryption in Transit | Secures data while it is being transmitted between systems, either within a network or across external connections. This protection is achieved through methods like email encryption, Data Loss Prevention (DLP) solutions, network security measures (e.g., firewalls), and strong authentication protocols to ensure secure data pathways. |
| Encryption in Use | Ensures the security of data that is actively being processed, modified, or accessed. Protection strategies include enforcing strict data access controls, implementing encryption techniques during computation, and managing user authentication and identity verification to prevent unauthorized exposure. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The process of verifying a user's identity before granting access to a system. Common authentication methods include static passwords, one-time passwords (OTPs), digital certificates, and biometric credentials. These mechanisms ensure that individuals are who they claim to be before interacting with secured resources. |
| Authorization | Defines and enforces user access rights and privileges. While authentication verifies identity, authorization determines what actions a user can perform and what data they can access. By restricting access to only necessary resources, authorization minimizes security risks and prevents unauthorized interactions with sensitive information. |
| Accounting | Tracks and logs user activities within a system, recording details such as login times, accessed resources, and data transfers. This provides a detailed audit trail for monitoring user behavior, enhancing security oversight, and facilitating forensic investigations when needed. |

**https://www.ccsinet.com/blog/aaa-identity-management/**

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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