**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 | Ensuring all input data is properly validated helps prevent attacks such as injection attacks and buffer overflows by verifying that the data is accurate, expected, and safe before it's processed. |
| 1. Heed Compiler Warnings | We should pay attention to compiler warnings as they often indicate potential security vulnerabilities. Fixing these warnings helps reduce the risk of common issues like buffer overflows, memory leaks, or undefined behaviors. |
| 1. Architect and Design for Security Policies | Building security into the system from the beginning ensures that the architecture supports robust security controls, including defining how security policies are enforced throughout the software lifecycle. |
| 1. Keep It Simple | Complex designs can introduce more security risks. Following the principle of simplicity reduces the attack surface, making it easier to identify and fix vulnerabilities. |
| 1. Default Deny | Always deny access by default, only allow explicitly. This reduces unauthorized access, ensuring proper authorization. |
| 1. Adhere to the Principle of Least Privilege | Limit users and systems to the minimal level of access necessary for their functions. This minimizes the potential damage if an account is compromised. |
| 1. Sanitize Data Sent to Other Systems | Ensure that data shared with other systems is cleansed of malicious code or unexpected input, preventing cross-site scripting (XSS), SQL injection, and other data-driven attacks. |
| 1. Practice Defense in Depth | Implement multiple layers of security controls to ensure that if one layer fails, additional defenses are in place, safeguarding the entire system against any single vulnerability. |
| 1. Use Effective Quality Assurance Techniques | Employ thorough testing, such as static code analysis, penetration testing, and code reviews, to identify and mitigate security issues early in the development process. |
| 1. Adopt a Secure Coding Standard | Following a well-established secure coding standard (e.g., CERT or OWASP) provides developers with guidelines for writing safer code. This helps them avoid common security pitfalls and reduces the likelihood of introducing 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** | [STD-001-CPP] | Do not cast to an out-of-range enumeration value. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);    if (enumVar < First || enumVar > Third) {  // Handle error  } |

| **Compliant Code** |
| --- |
| This compliant solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does this by restricting the converted value to one for which there is a specific enumerator value. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  if (intVar < First || intVar > Third) {  // Handle error  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** Heed Complier Warnings – complier warnings can detect unsafe casting practices that may lead to undefined behavior or security vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astree](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Cast-integer-to-enum | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | C++3013 | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | Cast-integer-to-enum | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Use valid references, pointers, and iterators to reference elements of a container |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, pos is invalidated after the first call to insert(), and subsequent loop iterations have [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  d.insert(pos, items[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, pos is assigned a valid iterator on each insertion, preventing undefined behavior. |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  pos = d.insert(pos, items[i] + 41.0);  }  } |

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

| **Principles(s):** Keep it simple – maintaining valid references ensures straightforward, predictable, behavior, reducing complexity and minimizing the risk of errors like segmentation faults, memory corruption, or undefined behavior in the code. |
| --- |

**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) | 8.1p0 | ALLOC.UAF | Use after free |
| [Astree](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Overflow\_upon\_deference | N/A |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP\_CTR51-a | Do not modify container while iterating over it |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | CERT C++: CTR51-CPP | Checks for use of invalid iterator (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. |

| **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):** Validate input data – emphasizes the need to verify and sanitize input to ensure it is valid. Attempting to create a std::string from a null pointer can lead to undefined behavior or crashes, so input validation ensures that only valid pointers are used when constructing objects. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule partially covered) |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-STRS1-a | Avoid null pointer dereferencing |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | LANG.MEM.NPD | Null pointer dereferencing |
| [Astree](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Assert\_failure | N/A |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Do not store an already-owned pointer value in an unrelated smart pointer |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, two unrelated smart pointers are constructed from the same underlying pointer value. When the local, automatic variable p2 is destroyed, it deletes the pointer value it manages. Then, when the local, automatic variable p1 is destroyed, it deletes the same pointer value, resulting in a double-free vulnerability. |
| #include <memory>    void f() {  int \*I = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::shared\_ptr objects are related to one another through copy construction. When the local, automatic variable p2 is destroyed, the use count for the shared pointer value is decremented but still nonzero. Then, when the local, automatic variable p1 is destroyed, the use count for the shared pointer value is decremented to zero, and the managed pointer is destroyed. |
| #include <memory>    void f() {  std::shared\_ptr<int> p1 = std::make\_shared<int>();  std::shared\_ptr<int> p2(p1);  } |

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

| **Principles(s):** Adhere to the principle of least privilege – emphasizes limiting access and ownership to resources, minimizing the risk of unintended consequences such as memory leaks or double deletions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astree](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Dangling\_pointer\_use | N/A |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-MEM56 | N/A |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-MEM56-a | Do not store an already owned pointer value in an unrelated smart pointer |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | CERTC++:MEM56-CPP | Checks for use of already-owned pointers (rule fully covered) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the local variable space is passed as the expression to the placement new operator. The resulting pointer of that call is then passed to ::operator delete(), resulting in undefined behavior due to ::operator delete() attempting to free memory that was not returned by ::operator new(). |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...    delete s1;  } |

| **Compliant Code** |
| --- |
| This compliant solution removes the call to ::operator delete(), instead explicitly calling s1's destructor. This is one of the few times when explicitly invoking a destructor is warranted. |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...    s1->~S();  } |

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

| **Principles(s):** Adhere to the principle of least privilege – involves minimizing resource access and ensuring that only the necessary components have control over allocated memory, thus reducing the risk of memory leaks or unauthorized. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | ALLOC.FNH ALLOC.DF ALLOC.TM ALLOC.LEAK | Free non-heap variable. Double free Type mismatch Leak. |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-MEM51-a CERT\_CPP-MEM51-b CERT\_CPP-MEM51-c CERT\_CPP-MEM51-d | Use the same form in corresponding calls to new/malloc and delete/free Always provide empty brackets ([]) for delete when deallocating arrays Both copy constructor and copy assignment operator should be declared for classes with a nontrivial destructor Properly deallocate dynamically allocated resources |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzercplusplus.NewDeleteLeaks -Wmismatched-new-delete clang-analyzerunix.MismatchedDeallocator | Checked by clang-tidy but does not catch all violations of this rule |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| This 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 assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution: |
| 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):** Heed compiler warnings – involves leveraging compile-time checks to validate constant expression, ensuring correctness and preventing runtime errors |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-DCL03 | N/A |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | Misc-static-assert | Checked by clang-tidy |
| [ÉCLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | CC2.DCL03 | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions thrown before main() begins executing |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the constructor for S may throw an exception that is not caught when globalS is constructed during program startup. |
| struct S {  S() noexcept(false);  };    static S globalS; |

| **Compliant Code** |
| --- |
| This compliant solution makes globalS into a local variable with static storage duration, allowing any exceptions thrown during object construction to be caught because the constructor for S will be executed the first time the function globalS() is called rather than at program startup. This solution does require the programmer to modify source code so that previous uses of globalS are replaced by a function call to globalS(). |
| struct S {  S() noexcept(false);  };    S &globalS() {  try {  static S s;  return s;  } catch (...) {  // Handle error, perhaps by logging it and gracefully terminating the application.  }  // Unreachable.  } |

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

| **Principles(s):** Practice defense in depth – emphasizes proactive measures like proper exception handling to defend against unexpected issues that could compromise the application’s stability and security from the outset. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astree](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Potentially-throwing-static-initialization | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | Cert-err58-cpp | Checked by clang tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | LANG.STRUCT.EXCPT.THROW | Use of throw |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP\_ERR58-a | Exceptions shall be raised only after start-up and before termination of the program |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::fstream object file is constructed. The constructor for std::fstream calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. Consequently, the underlying std::basic\_filebuf<T> object maintained by the object is not properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  std::terminate();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  file.close();  if (file.fail()) {  // Handle error  }  std::terminate();  } |

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

| **Principles(s):** Practice defense in depth – underscores the importance of resource management and minimizing potential vulnerabilities by ensuring that files are promptly closed to prevent unauthorized access or resource leaks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | ALLOC.LEAK | Leak |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork?_gl=1*1cs7u70*_gcl_au*MTYyMjYzODI0Ni4xNzI2OTI1NjQy*_ga*MTc4MDkwNDguMTcyODY4NDgwMg..*_ga_87WECW6HCS*MTcyODY4NDgwMi4xLjEuMTcyODY5MDE4Mi4zNy4wLjA.) | 2024.2 | RH.LEAK | N/A |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | CERT\_C-FIO51-a | Ensure resources are freed |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-009-CPP] | Do not delete a polymorphic object without a virtual destructor |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, b is a polymorphic pointer type whose static type is Base \* and whose dynamic type is Derived \*. When b is deleted, it results in undefined behavior because Base does not have a virtual destructor. The C++ Standard, [class.dtor], paragraph 4 [ISO/IEC 14882-2014], states the following:  If a class has no user-declared destructor, a destructor is implicitly declared as defaulted. An implicitly declared destructor is an inline public member of its class.  The implicitly declared destructor is not declared as virtual even in the presence of other virtual functions. |
| struct Base {  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the destructor for Base has an explicitly declared virtual destructor, ensuring that the polymorphic delete operation results in well-defined behavior. |
| struct Base {  virtual ~Base() = default;  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

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

| **Principles(s):** Keep it simple – emphasizes straightforward design practices that include virtual destructors to ensure proper cleanup of derived class resources, preventing memory leaks and undefined behavior in object deletion scenarios. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astree](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | non-virtual-public-destructor-in-non-final-class | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | LANG.STRUCT.DNVD | Delete with non-virtual destructor |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 303 S | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-OOP52-a | Define a virtual destructor in classes used as base classes which have virtual functions |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Do not return from a function declared [[noreturn]] |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, if the value 0 is passed, control will flow off the end of the function, resulting in an implicit return and undefined behavior. |
| #include <cstdlib>    [[noreturn]] void f(int i) {  if (i > 0)  throw "Received positive input";  else if (i < 0)  std::exit(0);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the function does not return on any code path. |
| #include <cstdlib>    [[noreturn]] void f(int i) {  if (i > 0)  throw "Received positive input";  std::exit(0);  } |

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

| **Principles(s):** Keep it simple – advocates for clear and predictable function behavior, ensuring that functions annotated with [[noreturn]] do not unexpectedly return control flow, thereby maintaining code simplicity and reliability. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astree](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Invalid-noreturn | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | LANG.STRUCT.RFNR | Return from noreturn |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-MSC53-a | Never return from functions that should not return |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | CERT C++: MSC53-CPP | Checks for [[noreturn]] functions returning to caller (rule fully covered) |

### 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 should be integrated throughout the entire DevSecOps pipeline to enforce security standards and ensure compliance without disrupting Green Pace's existing DevOps processes. Key areas for automation include the create phase, where IDE security plug-ins can automatically scan code for vulnerabilities as developers write it, providing immediate feedback. In the verify phase, automated tools like SAST, DAST, and IAST should be used to continuously scan for code vulnerabilities and security risks during testing. The preprod phase can also leverage automation through tools like Chaos Monkey and input fuzzing, which simulate potential system failures and unexpected inputs to test the system’s resilience.

To further enforce security, automated processes can be introduced in the release and prevent phases to ensure software signing, signature verification, and integrity checks are applied consistently. Monitoring and detection in the detect phase can use automated network monitoring, RASP, and UEBA to identify security breaches in real time. Finally, response orchestra-ion tools in the respond phase should be automated to ensure rapid reaction to threats, minimizing the impact of security incidents. By embedding automation at each stage, Green Pace can enhance security without manual intervention, ensuring ongoing compliance with security standards while maintaining development efficiency.

### 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 | Medium | Unlikely | Medium | P4 | L3 |
| STD-002-CPP | High | Probable | High | P6 | L2 |
| 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 | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-009-CPP | Low | Likely | Low | P9 | L2 |
| STD-010-CPP | Medium | Unlikely | Low | P2 | L3 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 | Data encryption at rest means turning stored data into a format that can't be read using algorithms like AES. This is to protect against unauthorized access or theft. This policy says that all sensitive data, like databases, file systems, and backups, must be encrypted while it's being stored. It also says that encryption keys must be managed securely, and that access to encrypted data must be limited using role-based access controls (RBAC). Regular checks and monitoring are really important to make sure everyone is following the rules and to spot any unauthorized access attempts. Encrypting data at rest helps to keep it safe from breaches, makes sure we follow the rules, and keeps sensitive information confidential. |
| Encryption in flight | Encryption in transit means protecting data as it moves between systems, like when it's sent over the internet or internal networks, by encrypting it. This stops others from seeing or using the data. This policy says that all sensitive data sent between users, servers, or devices must be encrypted using protocols like TLS (Transport Layer Security) or IPSec. Organizations should make sure that web traffic, email, APIs, and any external or internal communication channels are encrypted. Encryption in transit makes sure that even if data is seen by someone else, they can't read it unless they have the right decryption keys. This helps to stop data breaches while data is being sent, keeps data private and secure, and follows rules like GDPR and HIPAA. |
| Encryption in use | When data is being processed or used by applications, encryption helps keep it secure. This is important to protect the data from attackers who might gain access to the system. To make sure sensitive data is safe, it needs to be encrypted using techniques like homomorphic encryption or trusted execution environments (TEEs) while it's being processed. Organizations should use these encryption methods for important operations like financial transactions or handling personal data. Encryption in use stops unauthorized access to sensitive information while it's being used, which reduces the risk of data breaches. This helps keep the data confidential and intact during processing, especially in high-risk environments. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authenticating means making sure a user, system, or device is who they say they are. To make sure only the right people can get in, all systems need to use strong authentication methods. These could be things like using more than one way to prove who you are, like a password and a fingerprint, or a special code. Users should have to use at least two of these methods before they can get to important systems or data. We need to keep these methods up to date to stop any weak points. Strong authentication makes sure only the right people can get in, and keeps out anyone who shouldn't be there, like hackers. When organizations use strong authentication, they protect their systems and follow the rules for security and safety. |
| Authorization | Determining what actions or resources a user can access is called authorization. It's important to have controls in place to make sure users can only access data and systems they need for their jobs. This is usually done using role-based access control (RBAC) or similar methods. Access rights should be regularly reviewed and updated based on job responsibilities or changes in role. This helps prevent unauthorized access to sensitive information and reduces the risk of data leaks or insider threats. Effective authorization protects system resources and ensures they are only used by those with the right permissions. |
| Accounting | Accounting means keeping a record of user activities on a system to make sure it's secure and follows the rules. This policy says that all access to systems, data, and important actions must be recorded, including when they happened, who did them, and what they did. The records should be kept safe and checked regularly for anything suspicious or against the rules. Accounting helps make sure people are responsible for what they do by showing exactly who did what and when. This is important for investigating problems and understanding how things happened. It's essential for being open, following the rules, and quickly finding any security problems. |

**\***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 |  |
| 2.0 | 10/11/2024 | Project One | Elise Smith |  |
| [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 |