**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 6](#_Toc52464060)

[Coding Standard 3 7](#_Toc52464061)

[Coding Standard 4 9](#_Toc52464062)

[Coding Standard 5 11](#_Toc52464063)

[Coding Standard 6 13](#_Toc52464064)

[Coding Standard 7 15](#_Toc52464065)

[Coding Standard 8 17](#_Toc52464066)

[Coding Standard 9 19](#_Toc52464067)

[Coding Standard 10 21](#_Toc52464068)

[Defense-in-Depth Illustration 23](#_Toc52464069)

[Project One 23](#_Toc52464070)

[1. Revise the C/C++ Standards 23](#_Toc52464071)

[2. Risk Assessment 23](#_Toc52464072)

[3. Automated Detection 23](#_Toc52464073)

[4. Automation 23](#_Toc52464074)

[5. Summary of Risk Assessments 24](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 24](#_Toc52464076)

[7. Map the Principles 25](#_Toc52464077)

[Audit Controls and Management 26](#_Toc52464078)

[Enforcement 26](#_Toc52464079)

[Exceptions Process 26](#_Toc52464080)

[Distribution 27](#_Toc52464081)

[Policy Change Control 27](#_Toc52464082)

[Policy Version History 27](#_Toc52464083)

[Appendix A Lookups 27](#_Toc52464084)

[Approved C/C++ Language Acronyms 27](#_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 | Data received from any untrusted source should be validated against properly formatted examples. Doing so can limit the considerable vulnerability that is implicated by processing input data. Examples of sources that require input validation include user input, files not owned by the processing application, data received from unknown or untrusted domains, among others. |
| 1. Heed Compiler Warnings | Leaving compiler warnings on and setting the compiler to react with the most sensitivity will ensure that possible vulnerabilities are limited. External or internal IDE tools should be used to identify vulnerabilities associated with warnings. Suppressing warnings in code should be avoided and limiting appearance of warnings through syntax or logic changes should be opted for instead. |
| 1. Architect and Design for Security Policies | All security policies should be kept in mind when designing and developing code. This involves implementing secure code by following and enforcing these principles. |
| 1. Keep It Simple | Designs should not be over-engineered, nor should they be engineered beyond the scope of the demands that the software requires. In doing so, vulnerabilities associated with increasing complexity can be avoided or limited. |
| 1. Default Deny | Permissions should only be granted on the basis that they are required for the completion of a given task. Rather than excluding permissions, they should instead be granted when required. |
| 1. Adhere to the Principle of Least Privilege | Privileges necessary for a task and the processes they implicate should only be activated during the times they are required for the completion of the process and should be deactivated when they are no longer required. The activation and deactivation of privileges should occur at the latest and soonest possible moments, respectively. |
| 1. Sanitize Data Sent to Other Systems | It is important to assume that no validation will occur when sending data to other systems. It is therefore the responsibility of the system to clean any data to be sent when invoking these other systems. |
| 1. Practice Defense in Depth | Multiple layers of defensive strategies should be implemented with points of redundancy to limit points of failure themselves as well as their effects. This can prevent vulnerabilities that arise from suboptimal security practices. |
| 1. Use Effective Quality Assurance Techniques | A variety of techniques should be employed in concert to identify and subsequently address possible sources of vulnerability. Both in-house and independent reviews of design and security practices should be conducted, using techniques like unit testing, automation, integration testing, and others. |
| 1. Adopt a Secure Coding Standard | Having standards upon which systems may be designed and developed will ensure that these principles have a medium through which they may be followed. The target language should be coded in following a thorough standard that underscores the importance of security. |

### 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** | **Include the appropriate type information in function declarators** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Function declarators must be declared with the appropriate type information, including a return type and parameter list. If type information is not properly specified in a function declarator, the compiler cannot properly check function type information. When using standard library calls, the easiest (and preferred) way to obtain function declarators with appropriate type information is to include the appropriate header file.  Attempting to compile a program with a function declarator that does not include the appropriate type information typically generates a warning but does not prevent program compilation. These warnings should be resolved. (See MSC00-C. Compile cleanly at high warning levels.) |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses the identifier-list form for parameter declarations: |
| int max(a, b)  int a, b;  {  return a > b ? a : b;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, int is the type specifier, max(int a, int b) is the function declarator, and the block within the curly braces is the function body: |
| int max(int a, int b) {  return a > b ? a : b;  } |

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

| **Principles(s):**   * 1. Validate Input Data: This standard requires that data types are used in concordance with the expected value for that variable. The implication of this standard is thus that input should further be validated to ensure that the received value is appropriate for the variable’s data type. * 4. Keep It Simple: By requiring that data types are used in concordance with the received value, the policy of keeping it simple is upheld. It is thus more important that inputs are properly validated to ensure that the input is appropriate for the receiving variable. * 3. & 10. Architect and Design for Security Policies & Adopt a Secure Coding Standard: As part of a larger policy of secure coding, this standard aligns with these principles in that it sets strict guidelines for how coding should be performed to limit vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **function-prototype**  **implicit-function-declaration** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL07** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.FUNCS.PROT LANG.STRUCT.DECL.IMPT** | Incomplete function prototype Implicit Type |
| [GCC](https://wiki.sei.cmu.edu/confluence/display/c/GCC) | 4.3.5 |  | Can detect violation of this recommendation when the -Wstrict-prototypes flag is used |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **C1304, C2050, C3331, C3335, C3408, C3450** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2023.1 | **MISRA.FUNC.PROT\_FORM.KR.2012** **MISRA.FUNC.NOPROT.DEF** **MISRA.CAST.FUNC\_PTR.2012** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **21 S** **135 S** **170 S** | Fully implemented |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **718, 746, 936, 9074** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rec. DCL07-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.dcl07c.html) | Checks for:   * Cast between function pointers with different types * Function declared implicitly.   Rec. fully covered. |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 23.04 | **function-prototype**  **implicit-function-declaration** | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do not begin integer constants with 0 when specifying a decimal value** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | The C Standard defines octal constants as a 0 followed by octal digits (0 1 2 3 4 5 6 7). Programming errors can occur when decimal values are mistakenly specified as octal constants. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a decimal constant is mistakenly prefaced with zeros so that all the constants are a fixed length: |
| i\_array[0] = 2719;  i\_array[1] = 4435;  i\_array[2] = 0042; |

| **Compliant Code** |
| --- |
| To avoid using wrong values and to make the code more readable, do not preface constants with zeroes if the value is meant to be decimal: |
| i\_array[0] = 2719;  i\_array[1] = 4435;  i\_array[2] = 42; |

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

| **Principles(s):**   * 7. It is critical that data is sanitized prior to sending to another system. If values are meant to be decimals but declared as octals, they will not be received properly. * 4. By requiring that data types are used in concordance with the received value, the policy of keeping it simple is upheld. It is thus more important that inputs are properly validated to ensure that the input is appropriate for the receiving variable. * 3. & 10. As part of a larger policy of secure coding, this standard aligns with these principles in that it sets strict guidelines for how coding should be performed to limit vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **octal-constant** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL18** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.TYPE.OC** | Octal constant |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **C0339, C1272** |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Use valid references, pointers, and iterators to reference elements of a basic\_string** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Since std::basic\_string is a container of characters, this rule is a specific instance of CTR51-CPP. Use valid references, pointers, and iterators to reference elements of a container. As a container, it supports iterators just like other containers in the Standard Template Library. However, the std::basic\_string template class has unusual invalidation semantics. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example copies input into a std::string, replacing semicolon (;) characters with spaces. This example is noncompliant because the iterator loc is invalidated after the first call to insert(). The behavior of subsequent calls to insert() is undefined. |
| #include <string>    void f(const std::string &input) {  std::string email;    // Copy input into email converting ";" to " "  std::string::iterator loc = email.begin();  for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {  email.insert(loc, \*i != ';' ? \*i : ' ');  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the value of the iterator loc is updated as a result of each call to insert() so that the invalidated iterator is never accessed. The updated iterator is then incremented at the end of the loop. |
| #include <string>    void f(const std::string &input) {  std::string email;    // Copy input into email converting ";" to " "  std::string::iterator loc = email.begin();  for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {  loc = email.insert(loc, \*i != ';' ? \*i : ' ');  }  } |

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

| **Principles(s):**   * 3. & 10. As part of a larger policy of secure coding, this standard aligns with these principles in that it sets strict guidelines for how coding should be performed to limit vulnerabilities. |
| --- |

**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) | 7.3p0 | **ALLOC.UAF** | Use After Free |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **DF4746, DF4747, DF4748, DF4749** | [Insert text.] |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-STR52-a** | Use valid references, pointers, and iterators to reference elements of a basic\_string |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C++: STR52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr52cpp.html) | Checks for use of invalid string iterator (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL injection** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | SQL injection vulnerabilities arise in applications where elements of a SQL query originate from an untrusted source. Without precautions, the untrusted data may maliciously alter the query, resulting in information leaks or data modification. The primary means of preventing SQL injection are sanitization and validation, which are typically implemented as parameterized queries and stored procedures. |

| **Noncompliant Code** |
| --- |
| This code is noncompliant because the query is not validated and thus does not ensure that an injection is protected against. |
| #include <iostream>  #include <string>  #include <sql>  int main () {  std::string query;  std::cin >> query;  sql::result result = sql::query (query);  return 0;  } |

| **Compliant Code** |
| --- |
| This code is compliant because the query is first run through a function to ensure validity and protection against a SQL injection attack. |
| #include <iostream>  #include <string>  #include <sql>  void safe\_query (std::string& query) {  bool safe = true;  // Validate query and set safe to false  // if invalid or unsafe  if (safe) {  sql::query (query);  }  }  int main () {  std::string query;  std::cin >> query;  safe\_query (query);  return 0;  } |

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

| **Principles(s):**   * 3. & 10. As part of a larger policy of secure coding, this standard aligns with these principles in that it sets strict guidelines for how coding should be performed to limit vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | 6 | 2 |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Evaluating a pointer—including dereferencing the pointer, using it as an operand of an arithmetic operation, type casting it, and using it as the right-hand side of an assignment—into memory that has been deallocated by a memory management function is undefined behavior. Pointers to memory that has been deallocated are called dangling pointers. Accessing a dangling pointer can result in exploitable vulnerabilities.  It is at the memory manager's discretion when to reallocate or recycle the freed memory. When memory is freed, all pointers into it become invalid, and its contents might either be returned to the operating system, making the freed space inaccessible, or remain intact and accessible. As a result, the data at the freed location can appear to be valid but change unexpectedly. Consequently, memory must not be written to or read from once it is freed. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):**   * 3. & 10. As part of a larger policy of secure coding, this standard aligns with these principles in that it sets strict guidelines for how coding should be performed to limit vulnerabilities. |
| --- |

**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 | **dangling\_pointer\_use** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MEM50** |  |
| [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. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **ALLOC.UAF** | Use after free |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Understand the termination behavior of assert() and abort()** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Because assert() calls abort(), cleanup functions registered with atexit() are not called. If the intention of the programmer is to properly clean up in the case of a failed assertion, then runtime assertions should be replaced with static assertions where possible. (See DCL03-C. Use a static assertion to test the value of a constant expression.) When the assertion is based on runtime data, the assert should be replaced with a runtime check that implements the adopted error strategy (see ERR00-C. Adopt and implement a consistent and comprehensive error-handling policy). |

| **Noncompliant Code** |
| --- |
| This noncompliant code example defines a function that is called before the program exits to clean up: |
| void cleanup(void) {  /\* Delete temporary files, restore consistent state, etc. \*/  }    int main(void) {  if (atexit(cleanup) != 0) {  /\* Handle error \*/  }    /\* ... \*/    assert(/\* Something bad didn't happen \*/);    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the call to assert() is replaced with an if statement that calls exit() to ensure that the proper termination routines are run: |
| void cleanup(void) {  /\* Delete temporary files, restore consistent state, etc. \*/  }    int main(void) {  if (atexit(cleanup) != 0) {  /\* Handle error \*/  }    /\* ... \*/    if (/\* Something bad happened \*/) {  exit(EXIT\_FAILURE);  }    /\* ... \*/  } |

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

| **Principles(s):**   * 3. & 10. As part of a larger policy of secure coding, this standard aligns with these principles in that it sets strict guidelines for how coding should be performed to limit vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | CERT\_C-ERR06-a | Do not use assertions |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **586** | Fully supported |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | When an exception is thrown, control is transferred to the nearest handler with a type that matches the type of the exception thrown. If no matching handler is directly found within the handlers for a try block in which the exception is thrown, the search for a matching handler continues to dynamically search for handlers in the surrounding try blocks of the same thread. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

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

| **Principles(s):**   * 3. & 10. As part of a larger policy of secure coding, this standard aligns with these principles in that it sets strict guidelines for how coding should be performed to limit vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Detect and handle memory allocation errors** |
| --- | --- | --- |
| **Memory Allocation** | [STD-008-CPP] | The default memory allocation operator, ::operator new(std::size\_t), throws a std::bad\_alloc exception if the allocation fails. Therefore, you need not check whether calling ::operator new(std::size\_t) results in nullptr. The nonthrowing form, ::operator new(std::size\_t, const std::nothrow\_t &), does not throw an exception if the allocation fails but instead returns nullptr. The same behaviors apply for the operator new[] versions of both allocation functions. Additionally, the default allocator object (std::allocator) uses ::operator new(std::size\_t) to perform allocations and should be treated similarly. |

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

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

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

| **Principles(s):**   * 3. & 10. As part of a larger policy of secure coding, this standard aligns with these principles in that it sets strict guidelines for how coding should be performed to limit vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **45 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-MEM52-a** **CERT\_CPP-MEM52-b** | Check the return value of new Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: MEM52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem52cpp.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Use a static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Static Assertions** | [STD-009-CPP] | Assertions are a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities (see MSC11-C. Incorporate diagnostic tests using assertions). The runtime assert() macro has some limitations, however, in that it incurs a runtime overhead and because it calls abort(). Consequently, the runtime assert() macro is useful only for identifying incorrect assumptions and not for runtime error checking. As a result, runtime assertions are generally unsuitable for server programs or embedded systems. |

| **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** |
| --- |
| This portable compliant solution uses static\_assert: |
| static\_assert(sizeof(float) == 4);  int main () {  float f = 1.1234f;  // perform bit arithmetic on f  return 0;  } |

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

| **Principles(s):**   * 3. & 10. As part of a larger policy of secure coding, this standard aligns with these principles in that it sets strict guidelines for how coding should be performed to limit vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| [ECLAIR](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 10

| **Coding Standard** | **Label** | **Do not depend on the order of evaluation for side effects** |
| --- | --- | --- |
| Expressions | [STD-010-CPP] | In C++, modifying an object, calling a library I/O function, accessing a volatile-qualified value, or calling a function that performs one of these actions are ways to modify the state of the execution environment. These actions are called side effects. All relationships between value computations and side effects can be described in terms of sequencing of their evaluations. |

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

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

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

| **Principles(s):**   * 3. & 10. As part of a larger policy of secure coding, this standard aligns with these principles in that it sets strict guidelines for how coding should be performed to limit vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.EXP30** | Fully implemented |
| [GCC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/GCC) | 4.9 |  | Can detect violations of this rule when the -Wsequence-point flag is used |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**IncAndDecMixedWithOtherOperators**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-881) | Partially implemented |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wunsequenced | Can detect simple violations of this rule where path-sensitive analysis is not required |

### 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.

At Green Pace, it is critical that security is part of the software development workflow. In effect, this means that secure practices are not only demonstrated during the coding process by following the principles and standards outlined in this document, but also in that testing automation is used thoroughly to identify possible vulnerabilities.

In the pre-production phase, secure coding practices should be employed throughout this portion of the development cycle. Automation should be introduced during the build and verification phases of pre-production. Upon completion of the build phase, the verification and testing phase will allow for automated testing to locate security issues and present possible solutions.

In the production phase, automated testing must occur during the phase of transition/health check, monitoring and detection, and maintenance and stabilization. In essence, the production phase needs to be incorporating automation throughout this phase in order to identify vulnerabilities at the earliest possible moment.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| **Rule** | **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- | --- |
| **STD-001-CPP** | Low | Unlikely | Low | **P3** | **L3** |
| **STD-002-CPP** | Low | Unlikely | Low | P3 | L3 |
| **STD-003-CPP** | High | Probable | High | P6 | L2 |
| **STD-004-CPP** | High | Probable | High | 6 | 2 |
| **STD-005-CPP** | High | Likely | Medium | P18 | L1 |
| **STD-006-CPP** | Medium | Unlikely | Medium | P4 | L3 |
| **STD-007-CPP** | Low | Probable | Medium | P4 | L3 |
| **STD-008-CPP** | High | Likely | Medium | P18 | L1 |
| **STD-009-CPP** | Low | Unlikely | High | P1 | L3 |
| **STD-010-CPP** | Medium | Probable | Medium | P8 | 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 | Data that is physically stored and not actively being moved from one device or network to another is covered by this policy. Examples of this type of data include that on personal computers, portable storage devices (like flash drives), and external hard drives. This type of data must be encrypted in order to prevent breaches and attacks.  Tools that have resting-data-encryption by default, like Google Cloud and AWS, should be preferred for cloud based solutions. Company laptops must be encrypted per the system configuration. |
| Encryption at flight | Data that is actively being moved from one device or network to another is covered by this policy. This includes both local networks and public networks. This type of data needs to be encrypted to prevent possible breaches in the data communications process.  Strict TLS and SSL rules should be implemented, and HTTPS protocol is required. Services like Cloudfare that offer Universal SSL enabled should be preferred. Cloud based solutions like Google Cloud and AWS should be preferred for the default encryption of data in flight. |
| Encryption in use | Data that is currently being updated, processed, erased, accessed, or read by a system is covered by this policy. An example of data in use is that of a database, which is highly prone to attacks and breaches.  Data stored in databases must be encrypted depending on the sensitivity of that information. All data fields in database applications should be encrypted using AES-256. Data loss and breaches of information are prevented by encrypting the data throughout its lifecycle. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authenticating data is critical to ensure that data loss is prevented. User logins must be validated to affirm the identity of the individual. Examples of authentication include passwords, physical USB keys, and biometric identity verification. |
| Authorization | Authorization grants levels of access to the user. The information about the user’s level of access are stored alongside their identity to ensure that they are granted the correct level of access. This policy dictates what the user is allowed to do and what they are allowed access to, such as files.  Addition of new users must be accompanied by a corresponding level of authorization that indicates what the user will have the ability to do, see, and edit. |
| Accounting | Accounting ensures that all activity is tracked during a user’s session. Additionally, information about their session length, what data was interfaced with, and other key elements are recorded. This policy is critical in tracking which files are accessed by users, as well as what changes were made to database entries. |

**\***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 |