**Green Pace Developer: Security Policy Guide**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available for your compiler and eliminate warnings by modifying code. Use static and dynamic analysis tools to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set. |
| 1. Keep It Simple | Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. |
| 1. Default Deny | Base access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for your target development language and platform. |

Source: [SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682)

### 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-CLG] | Declare identifiers before using them. Do not declare a function with an implicit return type. |

Source: [DCL31-C. Declare identifiers before using them - SEI CERT C Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/c/DCL31-C.+Declare+identifiers+before+using+them)

| **Noncompliant Code** |
| --- |
| The following code does not declare the foo() function as it is supposed to. Thus, UINT\_MAX is incorrectly converted to -1. |
| #include <limits.h>  #include <stdio.h>  foo(void) {  return UINT\_MAX;  }  int main(void) {  long long int c = foo();  printf(“%lld\n”, c);  return 0;  } |

| **Compliant Code** |
| --- |
| The following code correctly defines the return type of foo() as an unsigned int. |
| #include <limits.h>  #include <stdio.h>  unsigned int foo(void) {  return UINT\_MAX;  }  int main(void) {  long long int c = foo();  printf(“%lld\n”, c);  return 0;  } |

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

| **Principles(s):**  #1: Input must match defined data types.  #4: Explicit variable declarations are easier to understand and track than implicit declarations.  #8: Variable, function and return declarations add a layer of defense. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.04 | Type-specifier  Function-return-type  Implicit-function-declaration  Undeclared-parameter | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL31 | Fully implemented |
| ÉCLAIR | 1.2 | CC2.DCL31 | Fully implemented |
| Parasoft C/C++ test | 2022.1 | CERT\_C\_DCL31-a | All functions shall be declared before use |

#### Coding Standard 2

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

Source: [EXP53-CPP. Do not read uninitialized memory - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory)

| **Noncompliant Code** |
| --- |
| The local variable is uninitialized resulting in undefined behavior. |
| #include <iostream>  void f() {  int i;  std::cout << i;  } |
|  |
| An int \* object is allocated by a new-expression, but the memory it points to is not initialized. This results in undefined behavior. |
| #include <iostream>  void f() {  int \*i = new int;  std::cout << i << “,” << \*i;  } |

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

|  |
| --- |
| The memory is direct-initialized to the value 12 prior to printing its value. |
| #include <iostream>  void f() {  int \*i = new int(12);  std::cout << i << “, ” << \*i;  } |

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

| **Principles(s):**  #1: Input shall be within ranges defined by variable initialization.  #2: Promptly correct all compiler warnings related to uninitialized variables. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Uninitialized-read | Partially checked |
| LDRA tool suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially implemented |
| Parasoft C/C++ test | 2022.1 | CERT\_CPP-EXP53-a | Avoid use before initialization |
| CodeSonar | 7.0p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local Uninitialized variable |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator. |

Source: [STR50-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator)

| **Noncompliant Code** |
| --- |
| The input is unbounded and could lead to buffer overflow. |
| #include <iostream>  void f() {  char buf[12];  std::cin >> buf;  } |

|  |
| --- |
| The read() function does not guarantee that the string will be null terminated, this results in undefined behavior if the character array does not contain a null terminator. |
| #include <fstream>  #include <string>  void f(std::istream &in) {  char buffer[32];  try {  in.read(buffer, sizeof(buffer));  } catch (std::ios\_base::failure &e) {  // Handle error  }  std::string str(buffer);  //…  } |

| **Compliant Code** |
| --- |
| A buffer overflow is prevented by using std::string instead of a bounded array. |
| #include <iostream>  #include <string>  void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo; |

|  |
| --- |
| This solution constructs the std::string object based on the number of characters read from the input stream. |
| #include <fstream>  #include <string>  void f(std::istream &in) {  char buffer[32];  try {  in.read(buffer, sizeof(buffer));  } catch (std::ios\_base::failure &e) {  // Handle error  }  std::string str(buffer, in.gcount());  //…  } |

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

| **Principles(s):**  #1: Storage for input string must allow for null terminator space.  #7: String lengths bounded by null terminator prevent occurrences of SQL injection and buffer overruns. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun Type overrun |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| Parasoft C/C++ test | 2022.1 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer |
| Polyspace Bug Finder | R2022a | CERT C++: STR50-CPP | Rule partially covered. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CLG] | Sanitize data passed to complex subsystems. |

Source: [STR02-C. Sanitize data passed to complex subsystems - SEI CERT C Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/c/STR02-C.+Sanitize+data+passed+to+complex+subsystems)

| **Noncompliant Code** |
| --- |
| An email address is inputted to a buffer and then uses this string as an argument in a call to system(). There is no validation resulting in a possible SQL Injection. |
| sprint(buffer, “/bin/mail %s < /tmp/email”, addr);  system(buffer); |

| **Compliant Code** |
| --- |
| A list of acceptable characters is defined and any unacceptable character are rejected. |
| static char ok\_chars[] = “abcdefjhijklmnopqrstuvwxyz”  “ABCDEFGHIJKLMNOPQRSTUVWXYZ”  [1234567890\_-.@](mailto:1234567890_-.@);  char user\_data[] = “Bad char 1:} Bad char 2:{“;  char \*cp = user\_data; /\* Cursor into string \*/  const char \*end = user\_data + strlen( user\_data);  for (cp += strspn(cp, ok\_chars); cp != end; cp += strspn(cp, ok\_chars)) {  \*cp = ‘\_’;  } |

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

| **Principles(s):**  #1: Validate input to prevent SQL injection.  #3: Use prepared SQL statements to prevent injection. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 6.5 | TAINTED\_STRING | Fully implemented |
| LDRA tool suite | 9.7.1 | 108 D, 109 D | Partially implemented |
| Parasoft C/C++ test | 2022.1 | CERT\_C-STR02-a  CERT\_C-STR02-b  CERT\_C-STR02-c | Protect against command injection Protect against file name injection |
| Polyspace Bug Finder | R2022a | CERT C: Rec. STR02-C | Checks for:   * Execution of externally controlled command * Command executed from externally controlled path * Library loaded from externally controlled path |

#### Coding Standard 5

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

Source: [MEM50-CPP. Do not access freed memory - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory)

| **Noncompliant Code** |
| --- |
| In this 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 permission of the vulnerable process. |
| #include <new>  struct S {  void f();  };  void g() noexcept(false) {  S \*s = new S;  //…  delete s;  //…  s->f();  } |

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

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

| **Principles(s):**  #2: Use static tools to detect, identify, and mitigate freed memory issues. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | dangling\_pointer\_use |  |
| CodeSonar | 7.0p0 | ALLOC.UAF | Use after free |
| 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 |
| LDRA tool suite | 9.7.1 | 483 S, 484 S | Partially implemented |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | Incorporate diagnostic tests using assertions. |

Source: [MSC11-C. Incorporate diagnostic tests using assertions - SEI CERT C Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/c/MSC11-C.+Incorporate+diagnostic+tests+using+assertions)

| **Noncompliant Code** |
| --- |
| Using the assert() macro to verify that a memory allocation succeeded would be inappropriate because doing so might lead to an abrupt termination of the process, opening the possibility of a denial-of-service attack. |
| char \*dupstring(const char \*c\_str) {  size\_t len;  char \*dup;  len = strlen(c\_str);  dup = (char \*)malloc(len + 1);  assert(NULL != dup);  memcpy(dup, c\_str, len + 1);  return dup;  } |

| **Compliant Code** |
| --- |
| This code properly detects and handle possible memory exhaustion. |
| char \*dupstring(const char \*c\_str) {  size\_t len;  char \*dup;  len = strlen(c\_str);  dup = (char \*)malloc(len + 1);  /\* Detect and handle memory allocation error \*/  if (NULL == dup) {  return NULL;  }  memcpy(dup, c\_str, len + 1);  return dup;  } |

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

| **Principles(s):**  #9: Use assertions to test / check code throughout the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| Coverity | 2017.07 | ASSERT\_SIDE\_EFFECT | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| Parasoft C/C++ test | 2022.1 | CERT\_C-MSC11-a | Assert liberally to document internal assumptions and invariants |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions. |

Source: [ERR51-CPP. Handle all exceptions - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions)

| **Noncompliant Code** |
| --- |
| 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();  } |

|  |
| --- |
| The thread entry point function thread\_start() does not catch exceptions thrown by throwing\_func(). |
| #include <thread>  void throwing\_func() noexcept(false);  void thread\_start() {  throwin\_func();  }  void f() {  std::thread t(thread\_start);  t.join();  } |

| **Compliant Code** |
| --- |
| All exceptions are handled and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);  void f() {  throwing\_func();  }  int main() {  try {  f();  } catch (…) {  // Handle error  }  } |

|  |
| --- |
| The thread\_start() handles all exceptions and does not rethrow, allowing the thread to terminate normally. |
| #include <thread>  void throwing\_func() noexcept(false);  void thread\_start(void) {  try {  throwin\_func();  } catch (…) {  // Handle error  }  }  void f() {  std::thread t(thread\_start);  t.join();  } |

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

| **Principles(s):**  #3: Design code using try/catch/throw to prevent code from stopping |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | main-function-catch-all  early-catch-all | Partially checked |
| CodeSonar | 7.0p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Parasoft C/C++test | 2022.1 | 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. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [STD-008-CLG] | Exclude user input from format strings. |

Source: [FIO30-C. Exclude user input from format strings - SEI CERT C Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/c/FIO30-C.+Exclude+user+input+from+format+strings)

| **Noncompliant Code** |
| --- |
| The code accepts the user input as a string. Integer overflow is not checked as well as user input which could result in SQL injection. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = “%s cannot be authenticated.\n”;  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  } else if (ret >= len)  /\* Handle truncated output \*/  }  fprintf(stderr, msg);  free(msg);  } |

| **Compliant Code** |
| --- |
| Fixes the problem by replacing the fprintf() call with a call to fputs(), which outputs msg directly to stderr without evaluating its contents. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = “%s cannot be authenticated.\n”;  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, leng, msg\_format, user);  if (ret < 0) {  /\* Handle error \*/  } else if (ret >= len)  /\* Handle truncated output \*/  }  fputs(msg, stderr);  free(msg);  } |

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

| **Principles(s):**  #10: Closing open files is best practice. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO30 | Partially implemented |
| CodeSonar | 7.0p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| Coverity | 2017.07 | TAINTED\_STRIN | Implemented |
| LDRA tool suite | 9.7.1 | 86 D | Partially Implemented |

#### Coding Standard 9

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

Source: [OOP52-CPP. Do not delete a polymorphic object without a virtual destructor - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP52-CPP.+Do+not+delete+a+polymorphic+object+without+a+virtual+destructor)

| **Noncompliant Code** |
| --- |
| When b is deleted, it results in undefined behavior because Base does not have a virtual destructor. The implicity 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** |
| --- |
| 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):**  #3: Destructing a polymorphic object that does not have a virtual destructor declared results in undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | non-virtual-public-destructor-in-non-final-class | Partially checked |
| CodeSonar | 7.0p0 | LANG.STRUCT.DNVD | Delete with Non-Virtual Destructor |
| LDRA tool suite | 9.7.1 | 303 S | Partially implemented |
| RuleChecker | 20.10 | Non-virtual-public-destructor-in-non-final-class | Partially checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-010-CPP] | Use valid iterator ranges. |

Source: [CTR53-CPP. Use valid iterator ranges - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR53-CPP.+Use+valid+iterator+ranges)

| **Noncompliant Code** |
| --- |
| The two iterators that delimit the range point into the same container, but the first iterator does not precede the second. |
| #include <algorithm>  #include <iostream>  #include <vector>  void f(const std::vector<int> &c) {  std::for\_each(c.end(), c.begin(), [](int i) { std::cout << I; });  } |

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

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

| **Principles(s):**  #3: Protect iterators from generating overflow errors.  #8: Preventing iterator overflow is a layer of defense. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | LANG.MEM.BO | Buffer Overrun |
| Helix QAC | 2022.2 | C++3802 |  |
| Parasoft C/C++ test | 2022.1 | CERT\_CPP-CTR53-a  CERT\_CPP-CTR53-b | Do not use an iterator range that isn’t really a range do not compare iterators from different containers |
| PRQA QA-C++ | 4.4 | 3802 |  |

### 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.

Unlike DevOps, DevSecOps integrates security measures into each step of the DevOps process. Within the Pre-production side there are four phases. These phases include, assess and plan, design, build, and verify and test. Within the assess and plan phase, threat modeling and security toll training and selections are added. IDE security is now addressed in the design and build phases. Finally, static testing, automated security scans, integration and unit tests are added to the verify and test phase.

DevSecOps adds security measure to the production side as well. Integrity check and defense-in-depth measures are used. Network monitoring, performance logs, and penetration testing is conducted as well. Testing is done throughout the whole process instead of at the very end.

### 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-CLG | LOW | Unlikely | LOW | 3 | 3 |
| STD-002-CPP | HIGH | PROBABLE | MEDIUM | 12 | 1 |
| STD-003-CPP | HIGH | LIKELY | MEDIUM | 18 | 1 |
| STD-004-CLG | HIGH | LIKELY | MEDIUM | 18 | 1 |
| STD-005-CPP | HIGH | LIKELY | MEDIUM | 18 | 1 |
| STD-006-CLG | LOW | UNLIKELY | HIGH | 1 | 3 |
| STD-007-CPP | LOW | PROBABLE | MEDIUM | 4 | 3 |
| STD-008-CLG | HIGH | LIKELY | MEDIUM | 18 | 1 |
| STD-009-CPP | LOW | LIKELY | LOW | 9 | 2 |
| STD-010-CPP | HIGH | PROBABLE | HIGH | 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 in rest | This indicates the data that is stored somewhere without being used by and/or transmitted to anyone (including software, third-parties, human beings, and so on). Encrypting our at-rest data could help us to better deal with a possible Data Breach. |
| Encryption at flight | The data is being transmitted somewhere to somewhere else. It is important that data is encrypted while in-transit. There are some threats with encryption at flight. This can include eavesdropping and Man-in-the-Middle attacks. |
| Encryption in use | Whenever the data is not just being stored passively on a hard drive or external storage media, but is being processed by one or more applications-and therefore in process of being generated, viewed, updated, appended, erased, and so on. |

[Data Encryption in-transit and at-rest - Definitions and Best Practices (ryadel.com)](https://www.ryadel.com/en/data-encryption-in-transit-at-rest-definitions-best-practices-tutorial-guide/)

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authenticating users is the first step in a secure identification system. The system needs to make sure the person accessing a system is who they say they are. The method of authentication a person can fall into three categories: what they know, who they are, and what they have. Someone’s password can authenticate what they know. A fingerprint or other bio metric tests can authenticate people. Access cards to enter a building can be used to authenticate a person. |
| Authorization | The right people should have the right access level to areas of a network. Mandatory Access Control (MAC) is the level of security a person is granted relating to the security of the content being accessed. Discretionary Access Control (DAC) is when access to a file or area is given by the owner of that area. Finally, Role-Based Access Control (RBAC) is when access is determined by the role within an organization. |
| Accounting | After a person begins logging into a network and working, their usage should be monitored. Knowing what files a person is accessing or attempting to access can inform whether more or less authorization is needed. Suspicious activity can prompt questions as to whether the person accessing the network was authenticated correctly. |

[The AAA Framework for Identity Access Security - Cipher](https://cipher.com/blog/the-aaa-framework-for-identity-access-security/)

**\***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 | 07/15/2022 | Initial Template | Thomas Cogley |  |
| 2.0 | 08/05/2022 | Final Template | Thomas Cogley |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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