**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 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 12](#_Toc52464063)

[Coding Standard 6 15](#_Toc52464064)

[Coding Standard 7 17](#_Toc52464065)

[Coding Standard 8 18](#_Toc52464066)

[Coding Standard 9 19](#_Toc52464067)

[Coding Standard 10 20](#_Toc52464068)

[Defense-in-Depth Illustration 21](#_Toc52464069)

[Project One 21](#_Toc52464070)

[1. Revise the C/C++ Standards 21](#_Toc52464071)

[2. Risk Assessment 21](#_Toc52464072)

[3. Automated Detection 21](#_Toc52464073)

[4. Automation 21](#_Toc52464074)

[5. Summary of Risk Assessments 22](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 22](#_Toc52464076)

[7. Map the Principles 23](#_Toc52464077)

[Audit Controls and Management 24](#_Toc52464078)

[Enforcement 24](#_Toc52464079)

[Exceptions Process 24](#_Toc52464080)

[Distribution 25](#_Toc52464081)

[Policy Change Control 25](#_Toc52464082)

[Policy Version History 25](#_Toc52464083)

[Appendix A Lookups 25](#_Toc52464084)

[Approved C/C++ Language Acronyms 25](#_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 inputs from untrusted data sources. Doing this helps prevent software vulnerabilities. Remain vigilant of external data sources as cause of concern, i.e., CLI arguments, environmental variables, and networking interfaces. |
| 1. Heed Compiler Warnings | Use highest warning levels that are available when compiling code, eliminate warnings through modification. Utilize static and dynamic analysis’ tools to aid in detecting and eliminating additional security flaws. |
| 1. Architect and Design for Security Policies | Stay mindful when designing software and implement all proper security policies. |
| 1. Keep It Simple | Keep coding simple and concise to avoid complex systems as complex systems are more susceptible to errors and security mechanisms. |
| 1. Default Deny | Standardize access decisions to permissions over exclusion. Default access should be denied with tailored conditions to permit access. |
| 1. Adhere to the Principle of Least Privilege | Process executions should contain the least amount of set privileges. Heightened permissions should only during the time it takes complete a task. Doing these aids in reducing an attacker’s chance of using arbitrary code. |
| 1. Sanitize Data Sent to Other Systems | Sanitize data to dissuade attackers from using injection attacks that manipulate components. |
| 1. Practice Defense in Depth | Always use multiple layers when utilizing DiD. This is so if one layer fails, the remaining layers can still perform their duties. |
| 1. Use Effective Quality Assurance Techniques | Helps increase the chance of detecting and eliminating vulnerabilities. Using multiple testing phases helps to more secure systems. |
| 1. Adopt a Secure Coding Standard | Always encompass secure coding standards that is tailored to the language being utilized. |

### 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 enumeration value that is not in range |

| **Noncompliant Code** |
| --- |
| Validates if the given value is within the enum range |
| Enum EnumType {  First,  Second,  Third  };  Void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  if(enumVar < First || enumVar > Third) {  }  } |

| **Compliant Code** |
| --- |
| Validates, before conversion, the given enum value to ensure the output is not an error |
| enum EnumType {  First,  Second,  Third  };  void func(int intVar) {  if (intVar < First || intVar > Third) {  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** Validate Input Data: Casting to an unspecified result does not allow the input data to  validated to give or revoke access to a system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++ - INT50 |  |
| CodeSonar | 6.2p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2022.1 | C++3013 |  |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-INT50-a | Enum type expression shall only contain values corresponding to the enumerators of the enum |
| PRQA-C++ | 4.4 | 3013 |  |
| PVS-Studio | 7.18 | V1016 |  |

#### 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** |
| --- |
| “pos” is invalidated after first insert() call; following loop iterations contain 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) {  d.insert(pos, items[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| “pos” is given valid iterator, 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):** Default Deny: Firstly, denies access to data inside a container unless valid references or pointers are related to the data inside the container. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | overflow\_upon\_dereference |  |
| CodeSonar | 6.2p0 | ALLOC.UAF | Use After Free |
| Helix QAC | 2022.1 | C++4746, C++4747, C++4748, C++4749 |  |
| Klocwork | 2022.1 | ITER.CONTAINER.MODIFIED |  |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-CTR51-a | No modification with iterating over |
| PVS-Studio | 7.18 | V783 |  |

#### Coding Standard 3

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

| **Noncompliant Code** |
| --- |
| a std::string object is created from the results of a call to std::getenv(). However, given the null pointer, code leads to undefined behavior |
| #include <cstdlib>  #include <string>  void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| Results from the call to std::getenv() are checked for null prior to the std::string object being 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):** Dereferencing a null pointer is undefined behavior. In some situations. The indicated severity is for this more severe case; on platforms where it is not possible to exploit a null pointer dereference to execute arbitrary code, the actual severity is low. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | assert\_failure |  |
| Helix QAC | 2022.1 |  |  |
| Klocwork | 2022.1 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF |  |
| ParasoftC/C++ test | 2021.2 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Avoid storage of already owned pointer value in unrelated smart pointer |

| **Noncompliant Code** |
| --- |
| Two unrelated smart pointers are constructed from the same underlying pointer value. When p2 is destroyed, managing pointer value is deleted. When p1 is destroyed, the same pointer value is deleted, 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** |
| --- |
| The std::shared\_ptr objects are related through copy construction. When p2 is destroyed, use count for the shared pointer value is decremented but still nonzero. When p1 is destroyed, use count for the shared pointer value is decremented to zero, and 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):** Architect and Design for Security Policies: Prevents issues from any stored variables being replaced and destroying later on in the code. This issue also relinquishes ownership of the managed pointer value. |
| --- |

**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 |  |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM56 |  |
| Helix QAC | 2022.2 | C++4271, C++4722, C++4723 |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-MEM56-a | Do not store already-owned pointer value in unrelated smart pointer |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| Local variable space is passed as the expression to the placement of a new operator. resulting pointer of that call results in undefined behavior due to ::operator delete() making attempt to free unreturned memory. |
| #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** |
| --- |
| Removes ::operator delete() call |
| #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):** – Practice Defense in Depth: Can cause a null-pointer or pointer behavior to be undefined and give results previous array new-expression to a command operator (new). |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | invalid\_dynamic\_memory\_allocation dangling\_pointer\_use |  |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM51 |  |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks -Wmismatched-new-delete clang-analyzerunix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| CodeSonar | 6.2p0 | ALLOC.FNH/ALLOC.DF/ALLOC.TM | Free non-heap variable/Double free/Type mismatch |
| Helix QAC | 2022.1 | C++2110, C++2111, C++2112, C++2113, C+ +2118, C++3337, C++3339, C++4262, C+ +4263, C++4264 |  |
| Klocwork | 2022.1 | CL.FFM.ASSIGNFM/CL.FFM.COPY/CL.FMM  FMM.MIGHT/FMM.MUST/FNH.MIGHT/FNH.MUST/FUM.GEN.MIGHT/FUM.GEN.MUST  UNINIT.CTOR.MIGHT/UNINIT.CTOR.MUST/UNINIT.HEAP.MIGHT/UNINIT.HEAP.MUST/UNINIT.STACK.ARRAY.MIGHT/UNINIT.STACK.ARRAY.MUST/UNINIT.STACK.MIGHT/UNINIT.STACK.MUST |  |
| LDRA tool suite | 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 | 2021.2 | 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 |
| Parasoft Insure ++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2021b | CERT C++:MEM51-CPP | Checks for invalid deletion of pointer; Invalid free of pointer; Deallocation of previously deallocated pointer |
| PRQA QA-C++ | 4.4 | 2110, 2111, 2112, 2113, 2118,  3337, 3339, 4262, 4263, 4264 |  |
| PVS-Studio | 7.18 | V515, V554, V611, V701, V748, V773, V1066 |  |
| SonarQube C/C++ Plugin | 4.10 | S1232 |  |

#### 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** |
| --- |
| Uses assert() to assert memory property regarding memory-mapped structure |
| #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** |
| --- |
| Only constant expressions, preprocessor conditional statement can be utilized |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  return error "Structure must not contain padding"  endif |

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

| **Principles(s):** – Use Effective Quality Assurance Techniques: Allows good programs and techniques to be  used to test sections of code for any issues. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 |  |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 6.2p0 | customization | Users can implement a custom check that reports uses of the assert() macro |
| Compass/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 |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

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

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

| **Compliant Code** |
| --- |
| f() handles all exceptions thrown by throwing\_func() and does not rethrow. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.  try {  throwing\_func();  } catch (...) {  // Handle error  }  }    int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  } |

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

| **Principles(s):** Allowing the application to abnormally terminate can lead to resources not being freed, closed, and so on. It is frequently a vector for denial-of-service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Stdlib-use | Partially Checked |
| CodeSonar | 6.2p0 | BADFUNC.ABORT  BACFUNC.EXIT | Use of abort  Use of exit |
| Helix QAC | 2022.1 | C++5014 |  |
| Klocwork | 2022.1 | MISRA.TERMINATE  CERT.ERR.ABRUPT\_TERM |  |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-ERR50-a  CERT\_CPP-ERR50-b  CERT\_CPP-ERR50-c  CERT\_CPP-ERR50-d  CERT\_CPP-ERR50-e  CERT\_CPP-ERR50-f  CERT\_CPP-ERR50-g  CERT\_CPP-ERR50-h  CERT\_CPP-ERR50-i  CERT\_CPP-ERR50-j  CERT\_CPP-ERR50-k  CERT\_CPP-ERR50-l  CERT\_CPP-ERR50-m  CERT\_CPP-ERR50-n | The execution of a function registered with 'std::atexit()' or 'std::at\_quick\_exit()' should not exit via an exception  Never allow an exception to be thrown from a destructor, deallocation, and swap  Do not throw from within destructor  There should be at least one exception handler to catch all otherwise unhandled exceptions  An empty throw (throw;) shall only be used in the compound-statement of a catch handler  Exceptions shall be raised only after start-up and before termination of the program  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  Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s)  Function called in global or namespace scope shall not throw unhandled exceptions  Always catch exceptions  Properly define exit handlers  The 'abort()' function from the 'stdlib.h' or 'cstdlib' library shall not be used  Avoid throwing exceptions from functions that are declared not to throw  The 'quick\_exit()' and '\_Exit()' functions from the 'stdlib.h' or 'cstdlib' library shall not be used |
| Polyspace Bug Finder | R2021b | CERT C++:ERR50-CPP | Checks for implicit call to terminate() function |
| PRQA QA-C++ | 4.4 | 5014 |  |
| PVS-Studio | 7.18 | V667, V2014 |  |
| RuleChecker | 20.10 | Stdlib-use | Partially Checked |
| SonarQube C/C++ Plugin | 4.10 | S990 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous | [STD-008-CPP] | Value-returning functions must return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| The programmer forgot to return the input value for positive input, so not all code paths return a value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| all code paths now return a value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  return a;  } |

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

| **Principles(s):** Failing to return a value from a code path in a value-returning function results in undefined behavior that might be exploited to cause data integrity violations. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Return-implicit | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MSC52 |  |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this rule |
| CodeSonar | 6.2p0 | LANG.STRUCT.MRS | Missing return statement |
| Helix QAC | 2022.1 | C++2888 |  |
| Klocwork | 2022.1 | FUNCRET.GEN  FUNCRET.IMPLICIT |  |
| LDRA tool suite | 9.7.1 | 2 D, 36 S | Fully implemented |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-MSC52-a | All exit paths from a function with non-void return type shall have an explicit return statement with an expression |
| Polyspace Bug Finder | R2021b | CERT C++:MSC52-CPP | Checks for missing return statements |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-009-CPP] | Honor replacement handler requirements |

| **Noncompliant Code** |
| --- |
| a replacement new\_handler is written to attempt to release salvageable resources when the dynamic memory manager runs out of memory |
| #include <new>    void custom\_new\_handler() {  // Returns number of bytes freed.  extern std::size\_t reclaim\_resources();  reclaim\_resources();  }    int main() {  std::set\_new\_handler(custom\_new\_handler);    // ...  } |

| **Compliant Code** |
| --- |
| Custom\_new\_handler() uses the return value from reclaim\_resources(). If it returns 0, then there will be insufficient memory for operator new to succeed. |
| #include <new>    void custom\_new\_handler() noexcept(false) {  // Returns number of bytes freed.  extern std::size\_t reclaim\_resources();  if (0 == reclaim\_resources()) {  throw std::bad\_alloc();  }  }    int main() {  std::set\_new\_handler(custom\_new\_handler);    // ...  } |

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

| **Principles(s):** Failing to meet the required behavior for a replacement handler results in undefined behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2022.1 | C++4776, C++4777, C++4778, C++4779 |  |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-OOP56-a  CERT\_CPP-OOP56-b  CERT\_CPP-OOP56-c | Properly define terminate handlers  Properly define unexpected handlers  Properly define new handlers |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input/Output | [STD-010-CPP] | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| 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** |
| --- |
| 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):** Failing to properly close files may allow an attacker to exhaust system resources and can increase the risk that data written into in-memory file buffers will not be flushed in the event of abnormal program termination. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2022.1 | C++4786, C++4787, C++4788 |  |
| Klocwork | 2022.1 | RH.LEAK |  |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2021b | CERT C++:FIO51-CPP | Checks for resource leak |

### 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.

It is highly recommended that an entity utilizes automated testing when writing code in order to have the work tested in the same manner as a program; with a sole purpose of flagging any errors that may be contained within the project. The DevOps process automation allows sections of code to be automatically tested by a program with little human interaction. This aids a project because it eliminates a lot of man hours. The automation process will protect the main branch of a project by reviewing and testing the code prior to that code being merged back into the main branch. These types of tests protect the project from flawed sections of code that can cause catastrophic consequences to the main project. An automation program allows each developer’s code to be held to the same standard of testing.

### 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 | 4 | 3 |
| STD-002-CPP | High | Probable | High | 6 | 2 |
| STD-003-CPP | High | Likely | Medium | 18 | 1 |
| STD-004-CPP | High | Likely | Medium | 18 | 1 |
| STD-005-CPP | High | Likely | Medium | 18 | 1 |
| STD-006-CPP | Low | Unlikely | High | 1 | 3 |
| STD-007-CPP | Low | Probable | Medium | 4 | 3 |
| STD-008-CPP | Medium | Probable | High | 8 | 2 |
| STD-009-CPP | Low | Probable | High | 2 | 3 |
| STD-010-CPP | Medium | Unlikely | Medium | 4 | 3 |

### Create Policies for Encryption and Triple A

Include all three types of encryptions (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 is encrypted while it is being stored. The data is accessible, but encryption prevents data from being read without a proper key. This provides a company and/or government to remain protected with additional defenses against any in-person. |
| Encryption at flight | Data is encrypted while being transmitted. The data may not be encrypted while its being stored but will become encrypted as it’s being transferred to another location. This encryption protects  sensitive data of an entity even if the data is intercepted by an outside. These encryptions are helpful when employees and/or users are allowed to telecommute to and from work, outside the office. |
| Encryption in use | Data is encrypted while being used and gives certain users specific access to the data depending on an employee’s security level. This encryption protects the businesses databases by creating layers of security to separate user activity from employees. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication utilizes user logins, passwords, and other security features such as biometric scanning and two-feature identification. |
| Authorization | Authorization uses the authentication features to determine the level of access given to a particular user. The security level allows the user to have admin credentials in order to gain access to databases, files, and employee records. |
| Accounting | Accounting uses the features from authentication and authorization to keep records on  when data has been changed or modified in a system and who made these changes. There are also systems that require a user to input a comment explaining why a change was made to the system. |

**\***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
  + - 3 – Architect and Design for Security Policies: Design a system and/or program to go

hand in hand with security features during the beginning of the projects development and do not leave the security features to be at the end of the project.

* + - 4 – Keep It Simple: Make the OS development logs simple to track of any changes or discrepancies.
    - 5 – Default Deny: User accounts are denied access until authorization and authentication is verified.
    - 6 – Adhere to the least privilege: use the least privilege security access required for a

retrieval of data instead of using the maximum-security privilege to prevent accidental data breaches.

* + - 7 – Sanitize Data Sent to Other Systems: Re-check files when suspicious activities are discovered for any security risks even after the data is already stored in the data
    - 8 – Practice Defense in Depth: Allow multi-layer defense features to work together and

notify each other of an attack or a collapse of a defense layer.

* + - 10 – Adopt a Secure Coding Standard: The security standard allows the development

team to be on the same page and uphold certain standards when it comes to security of a system.

* Firewall logs
  + - 1 – Validate Input Data: Validate any exterior data coming into the system to prevent security risks.
    - 5 – Default Deny: User accounts are denied access until authorization and authentication is verified.
    - 6 – Adhere to the least privilege: Adhere to the least privilege: use the least privilege

security access required for a retrieval of data instead of using the maximum-security privilege to prevent accidental data breaches.

* + - 7 – Sanitize Data Sent to Other Systems: Re-check files when suspicious activities are discovered for any security risks even after the data is already stored in the data
    - 8 – Practice Defense in Depth: Allow multi-layer defense features to work together and

notify each other of an attack or a collapse of a defense layer.

* + - 10 – Adopt a Secure Coding Standard: The security standard allows the development

team to be on the same page and uphold certain standards when it comes to security of a system.

* Anti-malware logs
  + - 1 – Validate Input Data: Validate any exterior data coming into the system to prevent security risks.
    - 5 – Default Deny: User accounts are denied access until authorization and authentication is verified.
    - 6 – Adhere to the least privilege: Adhere to the least privilege: use the least privilege security access required for a retrieval of data instead of using the maximum-security privilege to prevent accidental data breaches.
    - 7 – Sanitize Data Sent to Other Systems: Re-check files when suspicious activities are discovered for any security risks even after the data is already stored in the data
    - 8 – Practice Defense in Depth: Allow multi-layer defense features to work together and notify each other of an attack or a collapse of a defense layer.
    - 10 – Adopt a Secure Coding Standard: The security standard allows the development team to be on the same page and uphold certain standards when it comes to security of a system.

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 03/20/2022 | First Revision | Jacob Theisges |  |
| 1.2 | 04/10/2022 | Final Revision | Jacob Theisges |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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