**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

# Madeline McNeeley

# Decemember 2022

# CS405

## 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 | Data input validation should be present in order to mitigate software vulnerability. External data and data sources should be considered a risk. Some examples of this are network interfaces, command line prompts and scripts, user files, and environment variables. |
| 1. Heed Compiler Warnings | Compiler warnings should be thoroughly checked and addressed. Code should be modified to eliminate the compiler warnings and dynamic and static testing efforts should be implemented to detect other security flaws. |
| 1. Architect and Design for Security Policies | Architecture, design, and implementation should consider security factors and adhere to security policies. For example, authentication, private methods, and defined network permissions. |
| 1. Keep It Simple | Code design should as simplified and clean as possible. The less redundancy and clutter, the easier it is to find and eliminate security risk. |
| 1. Default Deny | Defualt behavior should deny access to an application or network until credentials are proven. |
| 1. Adhere to the Principle of Least Privilege | Processes should execute with the lowest amout of priviledges and permissions possible to preform intended functionality. This principle pertains to programs, processes, users, databases, networks, and whole systems. |
| 1. Sanitize Data Sent to Other Systems | Data should be sanitized before it is passed to a different system or to a subsystem. Removing SQL injections from SQL scripts before it is passed to another method could be used as an example for this. |
| 1. Practice Defense in Depth | Layering security efforts can be used to stop security risk if one layer of security is breached. Risk can be overseen and eliminated with a variety of strategies to prevent program exploitation. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques can be used to identify and eliminate vulnerabilities. An example of this would be a third party review of security. |
| 1. Adopt a Secure Coding Standard | Secure coding standards should be in place for each platform and technology stack. Different languages can vary in terms of security factors and vulnerabilities. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | INT50-CPP | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. On a two's complement system, the valid range of values that can be represented by EnumType are [0..3], so if a value outside of that range were passed to f(), the cast to EnumType would result in an unspecified value, and using that value within the if statement results in unspecified behavior. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);    if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

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

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

| **Principles(s):** Default deny- The correct approaching to this should be checking that the values can be represented by the enumeration type before preforming the conversion to guarantee the conversion does bot result in an unspecified value. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.2.0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Astree | 22.1.0 | cast-integer-to-enum | Partially checked |
| Parasoft | 2022.1 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| Rule Checker | 22.10 | cast-integer-to-enum | Partially Checked |

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/INT50-CPP.+Do+not+cast+to+an+out-of-range+enumeration+value>

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | EXP63-CPP | Do not rely on the value of a moved-from object |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, elements matching 42 are removed from the given container. The contents of the container are then printed to the standard output stream. However, if any elements were removed from the container, the range-based for loop iterates over an invalid iterator range, resulting in unspecified behavior. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(std::vector<int> &c) {  std::remove(c.begin(), c.end(), 42);  for (auto v : c) {  std::cout << "Container element: " << v << std::endl;  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, elements removed by the standard algorithm are skipped during iteration. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(std::vector<int> &c) {  auto e = std::remove(c.begin(), c.end(), 42);  for (auto i = c.begin(); i != c.end(); i++) {  if (i < e) {  std::cout << \*i << std::endl;  }  }  } |

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

| **Principles(s):** Keep it simple & Default deny. Verification should be in place to ensure elements are actually removed before preforming new action on the data structure. Keeping the code clean is keeping it simple. Default deny only preforms action when condition is met. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-EXP63-a | Do not rely on the value of a moved-from object |
| Polyspace Bug Finder | R2022b | CERT C++: EXP63-CPP | Checks for read operations that reads the value of a moved-from object (rule fully covered) |
| Helix QAC | 2022.3 | C++4701, C++4702, C++4703 | Partially Check |

#### <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP63-CPP.+Do+not+rely+on+the+value+of+a+moved-from+object>

#### 

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | 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;  } |

| **Compliant Code** |
| --- |
| To solve this problem, it may be tempting to use the std::ios\_base::width() method, but there still is a trap, as shown in this noncompliant code example |
| #include <iostream>    void f() {  char bufOne[12];  char bufTwo[12];  std::cin.width(12);  std::cin >> bufOne;  std::cin >> bufTwo;  } |

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

| **Principles(s):** 1. Validate Input Data. In this scenario, the parameters of the input data are not defined. Therefore, input data that that is bigger than the allotted memory of the variable could lead to buffer ocerflow.  Default deny. Validation should exist for when a user inputs data. If the input data is incorrect, the application should clear the cin and re-prompt to user to avoid a system crash. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the 'char' buffer to store input from 'std::cin' |
| Polyspace Bug Finder | R2022b | CERT C++: STR50-CPP | Checks for:  Use of dangerous standard function  Missing null in string array  Buffer overflow from incorrect string format specifier  Destination buffer overflow in string manipulation  Insufficient destination buffer size |
| RuleChecker | 22.1 | Stream-input-char-array | Partially checked |

#### <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>

#### 

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STR02-C | Sanitize data passed to complex subsystems |

| **Noncompliant Code** |
| --- |
| Data sanitization requires an understanding of the data being passed and the capabilities of the subsystem. |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| It is necessary to ensure that all valid data is accepted, while potentially dangerous data is rejected or sanitized. Doing so can be difficult when valid characters or sequences of characters also have special meaning to the subsystem and may involve validating the data against a grammar |
| static char ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  "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):** Sanitize Data Sent to Other Systems – Sanitizing data from other systems can help to avoid malicious data and vulnerability from being introduced to the system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | IO.INJ.COMMAND  IO.INJ.FMT  IO.INJ.LDAP  IO.INJ.LIB  IO.INJ.SQL  IO.UT.LIB  IO.UT.PROC | Command injection  Format string injection  LDAP injection  Library injection  SQL injection  Untrusted Library Load  Untrusted Process Creation |
| 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  Protect against SQL injection |
| Polyspace Bug Finder | R2022b | CERT C: Rec. STR02-C | Checks for:  Execution of externally controlled command  Command executed from externally controlled path  Library loaded from externally controlled path |
| LDRA tool suite | 9.7.1 | 108 D, 109 D | Partially implemented |

#### <https://wiki.sei.cmu.edu/confluence/display/c/STR02-C.+Sanitize+data+passed+to+complex+subsystems>

#### 

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM52-CPP | Detect and handle memory allocation errors |

| **Noncompliant Code** |
| --- |
| An array of int is created using ::operator new[](std::size\_t) and the results of the allocation are not checked. |
| #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** |
| --- |
| With std::nothrow, the new operator returns either a pointer or null pointer to the allocated space. |
| #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):** Head compiler errors. Implementing code that throws exceptions or allocatotio errors will help detect when memory is not being allocated properly. Handling these errors can mitigate the program terminalting unexpectedly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | Checked\_return | Finds inconsistencies in how function call return values are handled |
| Parasoft C/C++test | 2022.1 | [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 |
| Polyspace Bug Finder | R2022b | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |
| LDRA tool suite | 9.7.1 | 45 D | Runtime Detection |

#### <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors>

#### 

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | DCL03-C | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** Default Deny. A Preprocessor conditional statement can be used to check constant expressions. If the condition is met, the program errors out without introducing a security risk.  Use Effective Quality Assurance Techniques. Unit testing is an effective way to check quality. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 7.1p0 | customization | Users can implement a custom check that reports uses of the assert() macro |
| ECLAIR | 1.2 | CC2.DCL03 | 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 |
| LDRA tool suite | 9.7.1 | 44 S | Fully Implemented |

#### <https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression>

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR51-CPP | Handle all exceptions. |

| **Noncompliant Code** |
| --- |
| Neiter f() or main() catch exceptions thrown by throwing\_func() |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| 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):** Heed Compiler Warnings - Adding exception handling helps identify issues in code sooner.  Adopt a Secure Coding Standard – adding exception handling each time new changes are implemented ensure new changes are meeting established securitity standards. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| 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 |
| Poltspace Bug Finder | R2022b | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| RuleChecker | 22.10 | Main-function-catch-all-early-catch-all | Partially checked |

#### <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions>

#### 

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declaration | DCL60-CPP | Obey the one-definition rule |

| **Noncompliant Code** |
| --- |
| Two different translations unites are defining a class of the same name, with different definitions. They are not using the same sequence of tokens |
| // a.cpp  struct S {  int a;  };    // b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| Using a struct class allows for the class to be used in various places, and only one sequence token will be used for it. |
| // S.h  struct S {  int a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):** Keep it simple – one definition per variavble and data type ensure that code is readable and ensures that memory allocation is well established. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | type-compatibility  definition-duplicate  undefined-extern  undefined-extern-pure-virtual  external-file-spreading  type-file-spreading | Partially checked |
| CodeSonar | 7.1p0 | LANG.STRUCT.DEF.FDH  LANG.STRUCT.DEF.ODH | Function defined in header file  Object defined in header file |
| Parasoft C/C++ test | 2022.1 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| RuleChecker | 22.10 | type-compatibility  definition-duplicate  undefined-extern  undefined-extern-pure-virtual  external-file-spreading  type-file-spreading | Partially checked |

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule>

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| MSC. | MSC52-CPP | Value-returning functions must return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| The code below is missing a return 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** |
| --- |
| This was fixed by added another return statement for when a is not less than zero. |
| 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):** Default deny – Where conditional statements are present, having default behavior will ensure that every edge case or behavior is accounted for.  Coding standard – Returning with no value leads to undefined behavior and creates risk. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this rule, such as function-try-blocks |
| CodeSonar | 7.1P0 | LANG.STRUCT.MRS | Missing return statement |
| Parasoft C/C++test | 2022.10 | CERT\_CPP-MSC52-a | All exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression |
| Polyspace Bug Finder | R2022b | CERT C++ MSC52-CPP | Checks for missing return statements (rule partially covered) |

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/MSC52-CPP.+Value-returning+functions+must+return+a+value+from+all+exit+paths>

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| I/O | FIO51-CPP | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| In the code below, a stream object is constructed bot necer destructed. Terminating the stream does bot properly deconstruct it. |
| #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** |
| --- |
| Here the file is closed when the stram is finished. |
| #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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | ALLOC.LEAK | Leak |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Parasoft Insure++ | 2022.10 | RH.LEAK | Runtime detection |
| Polyspace Bug Finder | R2022b | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered) |

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed>

### 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.

DevSecOps is the notion of incorporating security into every phase of the DevOps process. The intention behind this to build a more securite system and catch vulnerabilities earlier by not saving security for the end of the deployment cycle. In the stages of pre-production, assessment and planning security architecture and implementation. After assessment, design can begin with security being in the forefront. After new implementations are build, static application testing and automated security testing can begin. After the pre-production stage, health checks are verified, and any vulnerabilities that were missed in the preproduction stage can be monitored for, detected, and addressed.

### 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 | High | Unlikely | Medium | High | 2 |
| INT50-CPP | Medium | Unlikely | Medium | P4 | L3 |
| EXP63-CPP | Medium | Probable | Medium | P8 | L2 |
| STR50-CPP | High | Likely | Medium | P18 | L1 |
| STR02-C | High | Likely | Medium | P18 | L1 |
| MEM52-CPP | High | Likely | Medium | P18 | L1 |
| DCL03-C | Low | Unlikely | High | P1 | L3 |
| ERR51-CPP | Low | Probable | Medium | P4 | L3 |
| DCL60-CPP | High | Unlikely | High | P3 | L3 |
| MSC52-CPP | Medium | Probable | Medium | P8 | L2 |
| FIO51-CPP | Medium | Unlikely | Medium | P4 | L3 |

### 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 type of encryption protects data storage. This could include logical souraces such as cload storage and databases such as physical storage like server and hard drives. InnoDB and VeraCrypt are examples. of encryption in rest tools |
| Encryption at flight | Protects data while it transers locations. This includes but is not limited to data being sent in an email, data being saved from a websote. For email, there are specific encryption toold such as PGP and S/MIME. For web related data transfer, HTTPS certification be be used to to ensure network level attacks do not pose a threat. |
| Encryption in use | Protects data during CRUD operations\_ Whe it is being created, updated, or read. Encryption in use occurs between the encryption in rest and at flight states. This policy is important as memory can be maliciously accessed and encryption keys can then be exposed. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying a user’s indentity through credentials. Credentials can come in the form of username/email and password, pin, biometrics, two-factor. This policy should exists for all existing and new users to prevent unauthorized use of the system. Authentication should be present on every teir of the sytesm to ensure databases, endpoints, user interfaces, and codebases are only being accessed by verified users. |
| Authorization | Authorization is the se urity policy that verifies what access and permissions a new or existing user has to the system. Authorization should adhere to the Principle of Least Privilege, meaning permission should be granted when absolutely necessary. Depending on the user, authorization should vary as to who can make changes to the database, what files can be accessed and edited, who can add new users to the system, etc. |
| Accounting | Accounting is the policy that refers to the act of keeping track of changes that happen within the system. The records kept for this are intented to be used to find security breaches at a faster rate by looking at trends in the network activity. If a user is hitting an endpoint in unusual, rapid succession it chould signify a DDOS attack. Actions such as file access evernts, data access and transfer, database updated, and user creation should be dated and timestamped to ensure transparency. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 12/10/2022 | Enhancement/update | Madeline McNeeley |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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