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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Input validation is the principle that ensures any data that is input into the system is gathered appropriately. Additionally, the input data is gathered in a way that does not leave the system vulnerable to bypasses, overflows, or other intrusive possibilities. This requires the capturing of the data be validated as appropriate and non-suspect by means of capturing limitations / verification. |
| 1. Heed Compiler Warnings | Compilers contain pre-built and updated warnings that assist in avoiding logical, security, and other flaws in the code. Thus, the warning system should be set to the highest level with the programmer adjusting the program to fix these warnings / errors, rather than bypassing the compiler warning. |
| 1. Architect and Design for Security Policies | Build and design software with security in mind. Utilize the security policies laid out by the security designer and lead architect of the project. |
| 1. Keep It Simple | This principle is as it says, keep it simple. Whether creating functions, methods, classes, or the main structure, focus on simplicity. This is to ensure clean and clear logic, decrease errors, and enhance code readability. |
| 1. Default Deny | This principle focuses on denial by default. Code safely by focusing on denial first, and ensure that access is only given if there is no doubt. |
| 1. Adhere to the Principle of Least Privilege | Only give the user access to what is specifically needed to complete the task. There is no need for a single user to have access to the entire system to complete minor tasks. Additionally, abstraction can be implemented to add another security checkpoint and prevent the user from gaining direct access to a system. This also helps to mitigate the amount of damage a user can do if their account is compromised. |
| 1. Sanitize Data Sent to Other Systems | This principle focuses on ensuring data sent is clean data by first abstracting then validating the data. Stripping data of potentially harmful characters and ensuring that there are no malicious contents. |
| 1. Practice Defense in Depth | This principle focuses on ensuring a layered approach to defense to maximize system protection and minimize security flaws. This approach is designed to ensure that where one layered would fail, another would catch and protect the system. |
| 1. Use Effective Quality Assurance Techniques | This principle focuses on not only ensuring that the system functions as intended, but also checks for unintended functionality or security flaws. This must be approached in a process driven and methodical way. |
| 1. Adopt a Secure Coding Standard | This principle focuses on ensuring code is written without common flaws, weaknesses, and vulnerabilities. Additionally, this ensures that code readability and uniformity. |

### 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 declare or define a reserved identifier |

Source[: https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL51-CPP.+Do+not+declare+or+define+a+reserved+identifier](:%20https:/wiki.sei.cmu.edu/confluence/display/cplusplus/DCL51-CPP.+Do+not+declare+or+define+a+reserved+identifier)

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a user-defined literal operator"" x is declared. However, literal suffix identifiers are required to start with an underscore; literal suffixes without the underscore prefix are reserved for future library implementations. |
| #include <cstddef>    unsigned int operator"" x(const char \*, std::size\_t); |

| **Compliant Code** |
| --- |
| In this compliant solution, the user-defined literal is named operator"" \_x, which is not a reserved identifier. |
| #include <cstddef>    unsigned int operator"" \_x(const char \*, std::size\_t); |

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

| **Principles(s):** Heed Compiler Warnings: The compiler will attempt to protect against this error, however it is not comprehensive and a logic/ security flaw can be created. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **86 S, 218 S, 219 S, 580 S** | Fully implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.ID.NU.MK**  **LANG.STRUCT.DECL.RESERVED** | Macro name is C keyword  Declaration of reserved name |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: DCL51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl51cpp.html) | Checks for redefinitions of reserved identifiers (rule partially covered) |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-DCL51-a** **CERT\_CPP-DCL51-b** **CERT\_CPP-DCL51-c** **CERT\_CPP-DCL51-d** **CERT\_CPP-DCL51-e** **CERT\_CPP-DCL51-f** | Do not #define or #undef identifiers with names which start with underscore Do not redefine reserved words Do not #define nor #undef identifier 'defined' The names of standard library macros, objects and functions shall not be reused The names of standard library macros, objects and functions shall not be reused (C90) The names of standard library macros, objects and functions shall not be reused (C99) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Do not rely on side effects in unevaluated operands |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP52CPP.+Do+not+rely+on+side+effects+in+unevaluated+operands>

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the expression a++ is not evaluated. |
| #include <iostream>  void f() {    int a = 14;    int b = sizeof(a++);    std::cout << a << ", " << b << std::endl;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the variable a is incremented outside of the sizeof operator. |
| #include <iostream>  void f() {    int a = 14;    int b = sizeof(a);    ++a;    std::cout << a << ", " << b << std::endl;  } |

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

| **Principles(s):** Keep It Simple: Do not over complicate a step. Keep things simple and separate function usage that may cause logical errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: EXP52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp52cpp.html) | Checks for logical operator operand with side effects |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-EXP52-a** **CERT\_CPP-EXP52-b** **CERT\_CPP-EXP52-c CERT\_CPP-EXP52-d CERT\_CPP-EXP52-e** | The operand of the sizeof operator shall not contain any expression which has side effects Object designated by a volatile lvalue should not be accessed in the operand of the sizeof operator The function call that causes the side effect shall not be the operand of the sizeof operator The operand of the 'typeid' operator shall not contain any expression that has side effects The operand of the 'typeid' operator shall not contain a function call that causes side effects |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **54 S, 133 S** | Partially implemented |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **sizeof** | Partially checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Use valid references, pointers, and iterators to reference elements of a basic\_string |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR52CPP.+Use+valid+references%2C+pointers%2C+and+iterators+to+reference+elements+of+a+basic_string>

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

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

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

| **Principles(s):** Validate Input Data: Assign values to a new value to prevent errors and pointer exceptions. |
| --- |

**Threat Level**

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

**Automation**

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

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Exclude user input from format strings |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/FIO30-C.+Exclude+user+input+from+format+strings>

| **Noncompliant Code** |
| --- |
| The incorrect\_password() function in this noncompliant code example is called during identification and authentication to display an error message if the specified user is not found or the password is incorrect. The function accepts the name of the user as a string referenced by user. This is an exemplar of [untrusted data](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-untrusteddata) that originates from an unauthenticated user. The function constructs an error message that is then output to stderr using the C Standard fprintf() function. |
| #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, len, msg\_format, user);    if (ret < 0) {      /\* Handle error \*/    } else if (ret >= len) {      /\* Handle truncated output \*/    }    fprintf(stderr, msg);    free(msg);  } |

| **Compliant Code** |
| --- |
| This compliant solution 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, len, 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):** Practice Defense in Depth: Never allow user input to directly interact with the system. Additionally, always find ways to abstract user input to a safe convention. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **IO.INJ.FMT MISC.FMT** | Format string injection Format string |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **TAINTED\_STRING** | Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-FIO30-a** **CERT\_C-FIO30-b** **CERT\_C-FIO30-c** | Avoid calling functions printf/wprintf with only one argument other than string constant Avoid using functions fprintf/fwprintf with only two parameters, when second parameter is a variable Never use unfiltered data from an untrusted user as the format parameter |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rule FIO30-C](https://www.mathworks.com/help/bugfinder/ref/certcrulefio30c.html) | Checks for tainted string format (rule partially covered) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Allocate sufficient memory for an object |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/MEM35C.+Allocate+sufficient+memory+for+an+object>

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, inadequate space is allocated for a struct tm object because the size of the pointer is being used to determine the size of the pointed-to object: |
| #include <stdlib.h>  #include <time.h>    struct tm \*make\_tm(int year, int mon, int day, int hour,                     int min, int sec) {    struct tm \*tmb;    tmb = (struct tm \*)malloc(sizeof(tmb));    if (tmb == NULL) {      return NULL;    }    \*tmb = (struct tm) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };    return tmb;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the correct amount of memory is allocated for the struct tm object. When allocating  space for a single object, passing the (dereferenced) pointer type to the sizeof operator is a simple way to allocate sufficient memory. Because the sizeof operator does not evaluate its operand, dereferencing an uninitialized or null pointer in this context is well-defined behavior. |
| #include <stdlib.h>  #include <time.h>    struct tm \*make\_tm(int year, int mon, int day, int hour,                     int min, int sec) {    struct tm \*tmb;    tmb = (struct tm \*)malloc(sizeof(\*tmb));    if (tmb == NULL) {      return NULL;    }    \*tmb = (struct tm) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };    return tmb;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard: Ensuring that code is compliant with well defined behavior is essential to creating secure code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW ALLOC.SIZE.TRUNC IO.TAINT.SIZE MISC.MEM.SIZE.BAD LANG.MEM.BO LANG.MEM.BU LANG.STRUCT.PARITH LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.MEM.TBA LANG.MEM.TO LANG.MEM.TU** | Addition overflow of allocation size Addition overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Truncation of allocation size Tainted allocation size Unreasonable size argument Buffer Overrun Buffer Underrun Pointer Arithmetic Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **400 S, 487 S, 115 D** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-MEM35-a** | Do not use sizeof operator on pointer type to specify the size of the memory to be allocated via 'malloc', 'calloc' or 'realloc' function |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rule MEM35-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemem35c.html) | Checks for:   * Pointer access out of bounds * Memory allocation with tainted size   Rule partially covered. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CLG | Understand the termination behavior of assert() and abort() |

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

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

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

| **Principles(s):** Use Effective Quality Assurance Techniques: Comprehension of proper test building is important to overall program security and compliance. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | CERT\_C-ERR06-a | Do not use assertions |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **586** | Fully supported |

#### Coding Standard 7

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

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

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

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

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

| **Principles(s):** Architect and Design for Security Policies: Ensuring that errors and exceptions are caught and properly handled is necessary to creating secure software. Ensure that all errors and exceptions are handled in appropriate ways. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **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 |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input / Output | STD-008-CPP | Close files when they are no longer needed |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51CPP.+Close+files+when+they+are+no+longer+needed>

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

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

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

| **Principles(s):** Architect and Design for Security Policies: Ensuring that files are closed when no longer needed is required to prevent unauthorized access. This helps to bolster overall system security. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **ALLOC.LEAK** | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **DF4786, DF4787, DF4788** | [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | STD-009-CPP | Guarantee that container indices and iterators are within the valid range |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR50CPP.+Guarantee+that+container+indices+and+iterators+are+within+the+valid+range>

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows a function, insert\_in\_table(), that has two int parameters, pos and value, both of which can be influenced by data originating from untrusted sources. The function performs a range check to ensure that pos does not exceed the upper bound of the array, specified by tableSize, but fails to check the lower bound. Because pos is declared as a (signed) int, this parameter can assume a negative value, resulting in a write outside the bounds of the memory referenced by table. |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, int pos, int value) {    if (pos >= tableSize) {      // Handle error      return;    }    table[pos] = value;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the parameter pos is declared as size\_t, which prevents the passing of negative arguments. |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, std::size\_t pos, int value) {    if (pos >= tableSize) {      // Handle error      return;    }    table[pos] = value;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard: Ensuring that common flaws are accounted for by following common procedures and coding standards. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.MEM.BO LANG.MEM.BU LANG.MEM.TO LANG.MEM.TU LANG.MEM.TBA LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.STRUCT.PARITH** | Buffer overrun Buffer underrun Type overrun Type underrun Tainted buffer access Pointer before beginning of object Pointer past end of object Pointer Arithmetic |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: CTR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcctr50cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-CTR50-a** | Guarantee that container indices are within the valid range |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **45 D, 47 S, 476 S, 489 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X** | Partially implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming (OOP) | STD-010-CPP | Do not use pointer-to-member operators to access nonexistent members |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP55-CPP.+Do+not+use+pointer-to-member+operators+to+access+nonexistent+members>

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a pointer-to-member object is obtained from D::g but is then upcast to be a B::\*. When called on an object whose dynamic type is D, the pointer-to-member call is well defined. However, the dynamic type of the underlying object is B, which results in undefined behavior. |
| struct B {    virtual ~B() = default;  };    struct D : B {    virtual ~D() = default;    virtual void g() { /\* ... \*/ }  };    void f() {    B \*b = new B;      // ...      void (B::\*gptr)() = static\_cast<void(B::\*)()>(&D::g);    (b->\*gptr)();    delete b;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the upcast is removed, rendering the initial code ill-formed and emphasizing the underlying problem that B::g() does not exist. This compliant solution assumes that the programmer's intention was to use the correct dynamic type for the underlying object. |
| struct B {    virtual ~B() = default;  };    struct D : B {    virtual ~D() = default;    virtual void g() { /\* ... \*/ }  };    void f() {    B \*b = new D; // Corrected the dynamic object type.      // ...    void (D::\*gptr)() = &D::g; // Moved static\_cast to the next line.    (static\_cast<D \*>(b)->\*gptr)();    delete b;  } |

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

| **Principles(s):** Architect and Design for Security Policies: Avoiding null pointer situations are important to ensuring that the system is secure. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-OOP55-a** | A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: OOP55-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop55cpp.html) | Checks for pointers to member accessing non-existent class members (rule fully covered). |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.MEM.UVAR** | Uninitialized Variable |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

Automation is a multi-step process and as such should be implemented in a variety of steps. The first stage where automation should be implemented is during the pre-production verification/ testing process. A series of static and dynamic tests should be developed/ deployed. The second stage in which automation should be implemented is at any time the code base must be analyzed (i.e. transitions and health checks). Additionally, automation can be deployed throughout other stages as deemed necessary.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-002-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-003-CPP | High | Probable | High | P6 | L2 |
| STD-004-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-005-CPP | High | Probable | High | **P6** | **L2** |
| STD-006-CLG | Medium | Unlikely | Medium | **P4** | **L3** |
| STD-007-CPP | Low | Probable | Medium | **P4** | **L3** |
| STD-008-CPP | Medium | Unlikely | Medium | **P4** | **L3** |
| STD-009-CPP | High | Likely | High | **P9** | **L2** |
| STD-010-CPP | High | Probable | High | **P6** | **L2** |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Data at rest is inactive data. This data is not moving between devices. While this data is the least susceptible of the three states, it can be accessed if the database/ storage is ever compromised. Thus, while data is not being accessed, it should be encrypted. |
| Encryption at flight | Data at/ in flight is data moving from one device to another. This data is exposed to threats from the internet or susceptible to data skimming through Wi-Fi. This is the most commonly targeted data by malicious actors. Thus, while data is in flight it should be encrypted. If the data is intercepted it may be rendered useless to the interceptor. |
| Encryption in use | Data in use is data that is being actively read, modified, or processing. This data is susceptible to various malware and physical data breaches, thus this data should be encrypted and require some form of authentication to access, modify, or write. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying that the user is the intended user for specific data access or system permissions. This policy is to help reinforce a zero trust perspective and ensure that data is only accessed by those who are members of the system. |
| Authorization | Authorization is the process of assigning specific privileges to users that grant access to parts, but not the whole system. This policy helps to ensure that if a user is compromised, then the entirety of the system is not compromised. |
| Accounting | Accounting is the process of tracking/ logging what each user is doing in the system. This policy is important to tracking down malicious behavior and threat identification. Additionally, it can help to repair compromised systems by knowing what and how a user accessed a specific part of 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
* 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 |  |
| 1.1 | 06/16/2023 | Initial Compilation | Kelly Lewis | Instructor |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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