**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 | Data from untrusted data sources can lead to improper input. Through proper input validation, we can circumvent a majority of software vulnerability exploits. |
| 1. Heed Compiler Warnings | Compiler warnings give a great indication of issues that are found in your code. Going through and eliminating these compiler warnings means that your code is more secure than if the warning were there. |
| 1. Architect and Design for Security Policies | Design and build your software with security in mind. Make sure that the proper variables are used for the incoming/outgoing data. One can also create subsystems that communicate with each other depending on the level of privileges needed. |
| 1. Keep It Simple | The more complex and convoluted the design of the software, the higher the chance for vulnerabilities to be present. Keeping things simple makes it easier to implement security changes or perform security maintenance on the system. |
| 1. Default Deny | Design the software to automatically deny access to information unless certain parameters are met. |
| 1. Adhere to the Principle of Least Privilege | Only permit needed information to each level of permission. If someone only needs access to a certain subset of data, they only get access to that subset of data. |
| 1. Sanitize Data Sent to Other Systems | If a certain functionality is unused in a system, remove it. Attackers can evoke unused functionality as a means to gather extra information. Removing these functions and sanitizing the information transmitted means that only the essentials are used or are present. |
| 1. Practice Defense in Depth | Having multiple layers of security with overlaps in functionality means that it will be much harder for hackers to gain information. This also allows the system to remain safe if one of the layers of defense are breached. |
| 1. Use Effective Quality Assurance Techniques | Utilizing security testing and other QA techniques helps to identify vulnerabilities and issues before changes/programs go live. This helps to prevent data leaks and successful attacks in the future through proactive steps now. |
| 1. Adopt a Secure Coding Standard | Develop/apply a secure coding standard that will be used. This allows for a common format that is proven to be effective., that also allows for security changes/upgrades to be implemented across the board in a quicker/simpler fashion. |

### 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-C] | Implement abstract data types using opaque types. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example is based on the managed string library developed by CERT [[Burch 2006](https://wiki.sei.cmu.edu/confluence/display/c/AA.+Bibliography#AA.Bibliography-Burch06)]. In this example, the managed string type and the functions that operate on this type are defined in the string\_m.h header file as follows: |
| **struct** string\_mx {  **size\_t** size;  **size\_t** maxsize;  unsigned **char** strtype;  **char** \*cstr;  };    **typedef** **struct** string\_mx string\_mx;    /\* Function declarations \*/  **extern** errno\_t strcpy\_m(string\_mx \*s1, **const** string\_mx \*s2);  **extern** errno\_t strcat\_m(string\_mx \*s1, **const** string\_mx \*s2);  /\* ... \*/ |

| **Compliant Code** |
| --- |
| This compliant solution reimplements the string\_mx type as a private type, hiding the implementation of the data type from the user of the managed string library. To accomplish this, the developer of the private data type creates two header files: an external string\_m.h header file that is included by the user of the data type and an internal file that is included only in files that implement the managed string abstract data type.  In the external string\_m.h file, the string\_mx type is defined to be an instance of struct string\_mx, which in turn is declared as an [incomplete type](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-incompletetype): |
| **struct** string\_mx;  **typedef** **struct** string\_mx string\_mx;    /\* Function declarations \*/  **extern** errno\_t strcpy\_m(string\_mx \*s1, **const** string\_mx \*s2);  **extern** errno\_t strcat\_m(string\_mx \*s1, **const** string\_mx \*s2);  /\* ... \*/ |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-unspecifiedbehavior). |
| **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** **class** EnumType {  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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-C] | Do not confuse narrow and wide character strings and functions |

| **Noncompliant Code** |
| --- |
| This noncompliant code example incorrectly uses the strncpy() function in an attempt to copy up to 10 wide characters. However, because wide characters can contain null bytes, the copy operation may end earlier than anticipated, resulting in the truncation of the wide string. |
| |  | | --- | | #include <stddef.h>  #include <string.h>    **void** func(**void**) {  **wchar\_t** wide\_str1[] = L"0123456789";  **wchar\_t** wide\_str2[] = L"0000000000";    **strncpy**(wide\_str2, wide\_str1, 10);  } | |

| **Compliant Code** |
| --- |
| This compliant solution uses the proper-width functions. Using wcsncpy() for wide character strings and strncpy() for narrow character strings ensures that data is not truncated and buffer overflow does not occur. |
| |  | | --- | | #include <string.h>  #include <wchar.h>    **void** func(**void**) {  **wchar\_t** wide\_str1[] = L"0123456789";  **wchar\_t** wide\_str2[] = L"0000000000";  /\* Use of proper-width function \*/  wcsncpy(wide\_str2, wide\_str1, 10);    **char** narrow\_str1[] = "0123456789";  **char** narrow\_str2[] = "0000000000";  /\* Use of proper-width function \*/  **strncpy**(narrow\_str2, narrow\_str1, 10);  } | |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C] | Do not attempt to modify string literals |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the char pointer str is initialized to the address of a string literal. Attempting to modify the string literal is [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior): |
| |  | | --- | | **char** \*str = "string literal";  str[0] = 'S'; | |

| **Compliant Code** |
| --- |
| As an array initializer, a string literal specifies the initial values of characters in an array as well as the size of the array. (See [STR11-C. Do not specify the bound of a character array initialized with a string literal](https://wiki.sei.cmu.edu/confluence/display/c/STR11-C.+Do+not+specify+the+bound+of+a+character+array+initialized+with+a+string+literal).) This code creates a copy of the string literal in the space allocated to the character array str. The string stored in str can be modified safely. |
| **char** str[] = "string literal";  str[0] = 'S'; |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-LLL] | Do not subtract or compare two pointers that do not refer to the same array |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, pointer subtraction is used to determine how many free elements are left in the nums array: |
| #include <stddef.h>    **enum** { SIZE = 32 };    **void** func(**void**) {  **int** nums[SIZE];  **int** end;  **int** \*next\_num\_ptr = nums;  **size\_t** free\_elements;    /\* Increment next\_num\_ptr as array fills \*/    free\_elements = &end - next\_num\_ptr;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the number of free elements is computed by subtracting next\_num\_ptr from the address of the pointer past the nums array. While this pointer may not be dereferenced, it may be used in pointer arithmetic. |
| #include <stddef.h>  **enum** { SIZE = 32 };    **void** func(**void**) {  **int** nums[SIZE];  **int** \*next\_num\_ptr = nums;  **size\_t** free\_elements;    /\* Increment next\_num\_ptr as array fills \*/    free\_elements = &(nums[SIZE]) - next\_num\_ptr;  } |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-C] | 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Guarantee exception safety |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows a flawed copy assignment operator. The implicit invariants of the class are that the array member is a valid (possibly null) pointer and that the nElems member stores the number of elements in the array pointed to by array. The function deallocates array and assigns the element counter, nElems, before allocating a new block of memory for the copy. As a result, if the new expression throws an exception, the function will have modified the state of both member variables in a way that violates the implicit invariants of the class. Consequently, such an object is in an indeterminate state and any operation on it, including its destruction, results in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <cstring>    **class** IntArray {  **int** \*array;  std::**size\_t** nElems;  **public**:  // ...    ~IntArray() {  **delete**[] array;  }      IntArray(**const** IntArray& that); // nontrivial copy constructor  IntArray& operator=(**const** IntArray &rhs) {  **if** (**this** != &rhs) {  **delete**[] array;  array = nullptr;  nElems = rhs.nElems;  **if** (nElems) {  array = **new** **int**[nElems];  std::**memcpy**(array, rhs.array, nElems \* **sizeof**(\*array));  }  }  **return** \***this**;  }    // ...  }; |

| **Compliant Code** |
| --- |
| In this compliant solution, the copy assignment operator provides the [strong exception safety](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-strongexceptionsafety) guarantee. The function allocates new storage for the copy before changing the state of the object. Only after the allocation succeeds does the function proceed to change the state of the object. In addition, by copying the array to the newly allocated storage before deallocating the existing array, the function avoids the test for self-assignment, which improves the performance of the code in the common case [[Sutter 2004](https://wiki.sei.cmu.edu/confluence/display/cplusplus/AA.+Bibliography#AA.Bibliography-Sutter04)]. |
| #include <cstring>    **class** IntArray {  **int** \*array;  std::**size\_t** nElems;  **public**:  // ...    ~IntArray() {  **delete**[] array;  }    IntArray(**const** IntArray& that); // nontrivial copy constructor    IntArray& operator=(**const** IntArray &rhs) {  **int** \*tmp = nullptr;  **if** (rhs.nElems) {  tmp = **new** **int**[rhs.nElems];  std::**memcpy**(tmp, rhs.array, rhs.nElems \* **sizeof**(\*array));  }  **delete**[] array;  array = tmp;  nElems = rhs.nElems;  **return** \***this**;  }    // ...  }; |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output (FIO) | [STD-008-CPP] | Do not alternately input and output from a file stream without an intervening positioning call |

| **Noncompliant Code** |
| --- |
| This noncompliant code example appends data to the end of a file and then reads from the same file. However, because there is no intervening positioning call between the formatted output and input calls, the behavior is [undefined](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <fstream>  #include <string>    **void** f(**const** std::string &fileName) {  std::fstream file(fileName);  **if** (!file.is\_open()) {  // Handle error  **return**;  }    file << "Output some data";  std::string str;  file >> str;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::basic\_istream<T>::seekg() function is called between the output and input, eliminating the [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <fstream>  #include <string>    **void** f(**const** std::string &fileName) {  std::fstream file(fileName);  **if** (!file.is\_open()) {  // Handle error  **return**;  }    file << "Output some data";    std::string str;  file.seekg(0, std::ios::beg);  file >> str;  } |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory Protection | [STD-009-CPP] | Explicitly construct and destruct objects when manually managing object lifetime. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a class with nontrivial initialization (due to the presence of a user-provided constructor) is created with a call to std::malloc(). However, the constructor for the object is never called, resulting in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior) when the class is later accessed by calling s->f(). |
| #include <cstdlib>    **struct** S {  S();    **void** f();  };    **void** g() {  S \*s = **static\_cast**<S \*>(std::**malloc**(**sizeof**(S)));    s->f();    std::**free**(s);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the constructor and destructor are both explicitly called. Further, to reduce the possibility of the object being used outside of its lifetime, the underlying storage is a separate variable from the live object. |
| #include <cstdlib>  #include <new>    **struct** S {  S();    **void** f();  };    **void** g() {  **void** \*ptr = std::**malloc**(**sizeof**(S));  S \*s = **new** (ptr) S;    s->f();    s->~S();  std::**free**(ptr);  } |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions and Error Handling | [STD-010-CPP] | Do not leak resources when handling exceptions |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, pst is not properly released when process\_item throws an exception, causing a resource leak. |
| #include <new>    **struct** SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.  **void** process\_item() noexcept(**false**);  };    **void** f() {  SomeType \*pst = **new** (std::**nothrow**) SomeType();  **if** (!pst) {  // Handle error  **return**;  }    **try** {  pst->process\_item();  } **catch** (...) {  // Process error, but do not recover from it; rethrow.  **throw**;  }  **delete** pst;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the exception handler frees pst by calling delete. |
| #include <new>    **struct** SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.    **void** process\_item() noexcept(**false**);  };    **void** f() {  SomeType \*pst = **new** (std::**nothrow**) SomeType();  **if** (!pst) {  // Handle error  **return**;  }  **try** {  pst->process\_item();  } **catch** (...) {  // Process error, but do not recover from it; rethrow.  **delete** pst;  **throw**;  }  **delete** pst;  } |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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.

[Insert your written explanations here.]

### 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 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 | [Insert text.] |
| Encryption at flight | [Insert text.] |
| Encryption in use | [Insert text.] |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | [Insert text.] |
| Authorization | [Insert text.] |
| Accounting | [Insert text.] |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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