**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 | User input needs to be scrubbed to prevent security vulnerabilities like SQL injection attacks. Make sure that the given user input matches the criteria that the program is expecting. If the user tries to give input other than what is expected, the program should block it and inform the user that an error occurred. |
| 1. Heed Compiler Warnings | Compilers can identify vulnerabilities and/or bugs in your code. Do not ignore these warnings because they can lead to serious flaws in the program. Address the warnings by correcting the code. This will reduce the risk of exploitation and improve your code security and quality. |
| 1. Architect and Design for Security Policies | Systems should be built with security in mind first and foremost. Including security policies from the get go will help to ensure that security measures are integrated in all parts of the system. Security should not be added to a system as an afterthought. |
| 1. Keep It Simple | Simple code is easier to look back on, analyze, and fix. Complexity is enemy of security. When programming, you should avoid overly complex code. This will reduce the likelihood of introducing vulnerabilities and makes it easier to identify and fix issues as they arise. |
| 1. Default Deny | Access to a system should be denied by default. Access should only be granted if certain, very specific verifications are met. |
| 1. Adhere to the Principle of Least Privilege | Users should only be granted access to a system, or parts of a system that they are necessary to do their jobs. This will help minimize the area of code that can be attacked, and ensures that only authorized users can access sensitive information and resources. |
| 1. Sanitize Data Sent to Other Systems | Data sent to other systems should be sanitized to prevent injection attacks and other threats. This means removing or encoding characters to ensure that data is written in the format that the receiving system expects. This goes hand in hand with user input validation. |
| 1. Practice Defense in Depth | There should be multiple layers to your security system. These can include technical features, administrative features, or physical controls. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques, such as coding reviews and automated testing can help to identify and fix security vulnerabilities early in the development process. Effective quality assurance will improve the overall security and reliability of the software. |
| 1. Adopt a Secure Coding Standard | Follow a secure coding standard when creating system. This will ensure that developers will adhere to the best practices for writing secure code. Secure coding standards will provide guidelines for avoiding common vulnerabilities and help to establish a consistent approach to security across the entire system development process. |

### 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] | Obey the one-definition rule |

**Source:** [**DCL60-CPP. Obey the one-definition rule - SEI CERT C++ Coding Standard - Confluence**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule)

| **Noncompliant Code** |
| --- |
| Structure and class definitions use the same identifier “S.” This will create two different types with the same name which will lead to conflict when attempting to link the two translations. This code violates the one-definition rule. |
| // a.cpp  **struct** S {  **int** a;  };    // b.cpp  **class** S {  **public**:  **int** a;  }; |

| **Compliant Code** |
| --- |
| In this compliant variation of the code, the structure “S” is being defined in the header file and then being included in a.cpp and b.cpp. One definition being used across all files ensure consistency and this adheres to the one-definition rule. |
| // 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):**  3. Architect and design for security policies  4. Keep it simple  10. Adopt a secure coding standard |
| --- |

**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 |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL60 | - |
| CodeSonar | 8.3p0 | LANG.STRUCT.DEF.FDH  LANG.STRUCT.DEF.ODH | Function defined in header file  Object defined in header file |
| Helix QAC | 2024.4 | C++ 1067  C++ 1509  C++ 1510 | - |
| LDRA tool suite | 9.7.1 | 286 S  2887 S | Fully implemented |
| Parasoft C/C++ntest | 2024.2 | CERT\_CPP-DCL60-a | The One Definition Rule shall not be violated |
| Polyspace Bug Finder | R2024a | CERT C++ : DCL60-CPP | Checks for inline constraints not respected (rule partially covered) |
| RuleChecker | 22.10 | Type-compatibility  Definition-duplicate  Undefined-extern  Undefined-extern-pure-virtual  External-file-spreading  Type-file-spreading | Partially checked |

#### Coding Standard 2

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

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

| **Noncompliant Code** |
| --- |
| In this example x is undefined and then y is set to x and then printed. Reading uninitialized memory will result in the program behaving in a way that is undefined. It will make debugging very hard and the program will be inconsistent. |
| int main() {  int x; // x is uninitialized  int y = x; // Reading uninitialized memory  std::cout << y << std::endl; // Output is unpredictable  return 0;  } |

| **Compliant Code** |
| --- |
| In this example of compliant code x is initialized to 0 so it is safe to read the value of x when y is set to equal it. The output will always be whatever x is initialized to. Predictable code is the best code. |
| int main() {  int x = 0; // x is initialized  int y = x; // Safe to read the value of x  std::cout << y << std::endl; // Output is predictable  return 0;  } |

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

| **Principles(s):**  **1. Validate Input Data**  **2. Heed Compiler Warnings**  **8. Practice Defense in Depth**  **9. Use Effective Quality Assurance Techniques**  **10. Adopt a Secure Coding Standard** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Uninitialized-read | Detects reads of uninitialized memory |
| CodeSonar | 8.3p0 | LANG.MEM.UVAR | Checks for uninitialized variable usage |
| Helix QAC | 2024.4 | DF726  DF2961 | Ensures variables are initialized before use |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-EXP53-a | Checks for use of variables before initialization |

#### Coding Standard 3

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

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

| **Noncompliant Code** |
| --- |
| If you do not designate enough space in memory for an expected string, you run the risk of creating overflows. |
| #include <iostream>    **void** f() {  **char** buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| Using std::string will allow for any size string as std::string grows as needed. Using a bounded array that has a fixed size will lead to larger strings being truncated or leading to a buffer overflow. |
| #include <iostream>  #include <string>    **void** f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s:**  **1.** Validate Input Data  2. Heed Compiler Warnings  4. Keep It Simple  8. Practice Defense in Depth  9. Use Effective Quality Assurance Techniques  10. Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Stream-input-char-array | Partially checked + soundly supported |
| CodeSonar | 8.3p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Polyspace Bug Finder | R2024a | 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 desination buffer size |
| - | - | - | - |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CLG] | Exclude user input from format strings. |

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

| **Noncompliant Code** |
| --- |
| User input is used for format string, if a user inputs something malicious it could trick the program into displaying information about memory addresses, which could potentially expose sensitive information. |
| char userInput[100];  // Dangerous! User input is part of the format string  printf(userInput); |

| **Compliant Code** |
| --- |
| The difference for this example is that the user input is treated simply as data and not as part of the format template. This makes it more secure. |
| char userInput[100];  printf("User input is: %s", userInput); |

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

| **Principles(s):**  1. Validate Input Data  5. Default Deny  7. Sanitize Data Sent to Other Systems  8. Practice Defense in Depth  10. Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.3p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| GCC | 4.3.5 | - | Can detect violations of this rule when the” – Wformate – security” flag is used |
| Parasift C/C++ test | 2024.2 | CERT\_C-FIO30-a | Avoid calling functions printf/wprintf with only one argument other than string constant  Avoid using function fprintf/fwprintf with only two parameter, when second parameter is a variable  Never use unfiltered data from an untrusted format parameter |
| Polyspace Bug Finder | R2024a | CERT C: Rule FIO30-C | Checks for tainted string format (rule partially covered) |

#### Coding Standard 5

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

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

| **Noncompliant Code** |
| --- |
| “In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the [vulnerability](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-vulnerability) can be [exploited](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-exploit) to run arbitrary code with the permissions of the vulnerable process.” |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| “In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required.” |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...    s->f();  **delete** s;  } |

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

| **Principles(s):**  2. Heed Compiler Warnings  4. Keep It Simple  8. Practice Defense In Depth  9. Use Effective Quality Assurance Techniques  10. Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | Dangling\_pointer\_use | Astree reports all accesses to freed allocated memory |
| Axivion Bauhas Suite | 7.2.0 | CertC-MEM30 | Detects memory accesses after its deallocation and double memory deallocations |
| Coverity | 2017.07 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Parasoft C/C++ test | 2024.2 | CERT\_C-MEM30-a | Do not use resources that have been freed |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Usa a static assertion to test the value of a constant expression. |

**Source:** [**DCL03-C. Use a static assertion to test the value of a constant expression - SEI CERT C Coding Standard - Confluence**](https://wiki.sei.cmu.edu/confluence/display/c/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. Although the use of the runtime assertion is better than nothing, it needs to be placed in a function and executed. This means that it is usually far away from the definition of the actual structure to which it refers. The diagnostic occurs only at runtime and only if the code path containing the assertion is executed.” |
| #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. Using #error directives allows for clear diagnostic messages. Because this approach evaluates assertions at compile time, there is no runtime penalty.” |
| **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):**  2. Heed Compiler Warnings  4. Keep It Simple  8. Practice Defense In Depth  9. Use Effective Quality Assurance Techniques  10. Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhas Suite | 7.2.0 | CertC-DCL03 | - |
| Clang | 3.9 | Misc-stattic-assert | Checked by clang-tidy |
| CodeSonar | 8.3p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| Compass/ROSE | - | - | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |

#### Coding Standard 7

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

**Source:** [**ERR51-CPP. Handle all exceptions - SEI CERT C++ Coding Standard - Confluence**](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):**  2. Heed Compiler Warnings  3. Architect and Design for Security Policies  8. Practice Defense in Depth  9. Use Effective Quality Assurance Techniques  10. Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |
| CodeSonar | 8.3p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions  Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Close files when they are no longer needed. |

**Source:** [**FIO51-CPP. Close files when they are no longer needed - SEI CERT C++ Coding Standard - Confluence**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+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):**  4. Keep It Simple  6. Adhere to the Principle of Least Privilege |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.3p0 | ALLOC.LEAK | - |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Parasoft Insure++ | - | - | Runtime detection |
| Polyspace Bug Finder | R2024a | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | Do not access an object outside of its lifetime. |

**Source:** [**EXP54-CPP. Do not access an object outside of its lifetime - SEI CERT C++ Coding Standard - Confluence**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP54-CPP.+Do+not+access+an+object+outside+of+its+lifetime)

| **Noncompliant Code** |
| --- |
| “In this noncompliant code example, a pointer to an object is used to call a non-static member function of the object prior to the beginning of the pointer's lifetime, resulting in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior).” |
| **struct** S {  **void** mem\_fn();  };    **void** f() {    S \*s;    s->mem\_fn();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, storage is obtained for the pointer prior to calling S::mem\_fn(). |
| **struct** S {  **void** mem\_fn();  };    **void** f() {    S \*s = **new** S;    s->mem\_fn();  **delete** s;  } |

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

| **Principles(s):**  2. Heed compiler warnings  4. Keep It Simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wdangling-initializer-list | Catches some lifetime issues related to incorrect use of std::initializer\_list<> |
| CodeSonar | 8.3p0 | IO.UAC  ALLOC.UAF | Use after close  Use after free |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-EXP54-a  CERT\_CPP-EXP54-b  CERT\_CPP-EXP54-c | Do not use resources that have been freed  The address of an object with automatic storage shall not be returned from a function  The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist |
| Polyspace Bug Finder | R2024a | CERT C++: EXP54-CPP | Checks for: Non-initialized variable or pointer, Use of previously freed pointer, Pointer or reference to stack variable leaving scope, Accessing object with temporary lifetime |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Range check element access. |

**Source:** [**STR53-CPP. Range check element access - SEI CERT C++ Coding Standard - Confluence**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access)

| **Noncompliant Code** |
| --- |
| “In this noncompliant code example, the value returned by the call to get\_index() may be greater than the number of elements stored in the string, resulting in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior).” |
| #include <string>    **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| “This compliant solution uses the std::basic\_string::at() function, which behaves in a similar fashion to the index operator[] but throws a std::out\_of\_range exception if pos >= size().” |
| #include <stdexcept>  #include <string>  **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");  **try** {      s.at(get\_index()) = '1';    } **catch** (std::out\_of\_range &) {      // Handle error    }  } |

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

| **Principles(s):**  1. Validate Input Data  4. Keep It Simple  8. Practice Defense in Depth  9. Use Effective Quality Assurance Techniques  10. Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | assert\_failure |  |
| CodeSonar | 8.3p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer overrun  Buffer underrun  Tainted buffer access  Typer overrun  Type underrun |
| Helix QAC | 2024.4 | C++ 3162  C++ 3163  C++ 3154  C++ 3165 | Guarantee that container indices are within the valid range |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-STR53-a | Guarantee that container indices are within the valid range |

### 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.

In the Assess and Plan stage, we will identify the coding standards that require enforcement and compliance. This will include evaluating the existing processes and deciding which tools and methods will be needed to integrate automation into the DevSecOps pipeline. During the Design stage, we will architect the integration of automation tools into the DevSecOps pipeline. This will include designing the workflow for static code analysis, security scanning, and automated testing. We will create a blueprint that will specify how the automation tools will be integrated into the code commit stage. In the Build stage, developers will implement the coding standards and incorporate static analysis tools into their development environments. Automation scripts and configurations will be created to ensure that these tools are executed as part of the building process. This stage makes sure that coding standards are enforced from when the code is written. During the Verify and Test stage, automated tests will be executed using frameworks like Google Test. The Transition and Health Check stage will help to ensure that the transition from development to production is smooth and that any potential issues are identified and handled before deployment. During the Monitor and Detect stage, runtime monitoring tools will be used to check the application in the production environment. These tools will give us insight into how the application is running in real-time. In the Respond stage, feedback will be provided to developers to help them improve the application and handle any errors or bugs that occur. Any security vulnerabilities that may have been missed will be addressed here. In the Maintain and Stabilize stage, ongoing maintenance and updates will be performed to make sure that automation tools and processes remain effective and up to date. This is the part of the cycle where we can identify areas for improvement.

### 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(3) | 3 |
| STD-002-CPP | High | Probable | Medium | Low(12) | 1 |
| STD-003-CPP | High | Likely | Medium | Low(18) | 1 |
| STD-004-CLG | High | Likely | Medium | Low(18) | 1 |
| STD-005-CLG | High | Likely | Medium | Low(18) | 1 |
| STD-006-CPP | Low | Unlikely | High | High(1) | 3 |
| STD-007-CPP | Low | Probable | Medium | High(4) | 3 |
| STD-008-CPP | Medium | Unlikely | Medium | High(4) | 3 |
| STD-009-CPP | High | Probable | High | Medium(6) | 2 |
| STD-010-CPP | High | Unlikely | High | Medium(6) | 2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | This is to protect data that is being stored and not currently in use. Data should always be encrypted during storage. Any passwords should be encrypted as well and decryption keys should only be held by people who have access to that information. |
| Encryption in flight | This is about protecting data that is being sent from one place to another. Data should be encrypted while it is being transmitted between systems or over networks. This will prevent unauthorized individuals from intercepting and accessing sensitive information during data transmission. |
| Encryption in use | Data should be encrypted even while it is being processed or used by applications. This involves protecting sensitive information while it is being accessed, manipulated, or stored in memory (even if only temporarily). Ensuring that data is encrypted even while in use will make sure that even if by some chance data does leak, it cannot be read unless it is decrypted. And it should only be able to be decrypted by individuals that are supposed to have access to that data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is the process of verifying the identity of the user or the system that is trying to access data before they are given permission. It ensures that the person or system that is requesting access is who they claim to be. Methods for authentication include passwords and multi-factor authenticators. This is one of the most important parts of protecting sensitive information. |
| Authorization | This determines which users and or systems are allowed to do certain actions or operations after being authenticated. It ensures that users have the appropriate permissions to access specific resources or perform certain tasks. For example, you wouldn’t want a regular system user to be able to have administrative powers for a system and be able to access account information for accounts that they do not have authentication for. |
| Accounting | This involves tracking and recording user activities and any changes in the system. It provides a bread-crumb trail that you can follow which can be used to monitor any thing that happens within the system such as new users accessing the system, anomalies, files accessed, files changed, or security breaches. Implementing accounting measures helps to track user activity within 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 |  |
| 2.0 | 01/26/2025 | Module 3 | Joseph Kawamoto | [Insert text.] |
| 3.0 | 02/14/2025 | Project 1 | Joseph Kawamoto | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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