**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 proper testing of any input supplied by the user or application. Adequate testing of these inputs prevents improperly formatted data from entering the information system. Input validation is the first line of defense against any hack. Input Validation has benefits for both user experience and security, which include improved error prevention, greater performance, and cleaner data. All of these benefits are built on top of each other. Because we validate user input, there is less chance for error, which means that our system gets the data it was built for, making our system run better and preventing malicious data from entering it. |
| 1. Heed Compiler Warnings | Compiler warnings can alert you to coding errors and warn you of vulnerabilities that might otherwise go unnoticed. Making sure that your compiler is up to date with the latest software is a principle everyone should follow when practicing secure coding. |
| 1. Architect and Design for Security Policies | Depending on the security needs of the specific client, the security architecture is a set of security principles, methods, and models designed to align with the enterprise's objectives to keep the organization safe from cyber threats. Each organization is different; therefore, each organization will require some personalized framework that will be designed to meet its unique needs. |
| 1. Keep It Simple | Technology is a tool that we use to achieve cybersecurity. Too much dependence and over-complexity of the tools used to build a secure system can spell disaster. Using simplicity as a principle, we ensure that everyone clearly understands the goals and obstacles, and it becomes much easier to work with everyone. If a process is not repeatable by others on the team, then that process might as well be useless. |
| 1. Default Deny | Deny, by default, denies everything inbound and outbound that the firewall has not expressly permitted. This protects us from malware that might get permission one way, but then once inside the “gates”, it is able to do what it wants. If we deny by default, we set up a situation where every action is rejected by default unless permission otherwise was expressly given. |
| 1. Adhere to the Principle of Least Privilege | You cannot abuse, misuse, or lose something you don’t have. The idea of POLP is that you only have access to parts of the system that you need to complete your job and no more. The fewer people that have access to sensitive systems, the less chance it is for them to be compromised. In short, the least amount of access should be granted to each user as possible in order for them to get their job done. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data is needed because it doesn't get deleted when you delete something off of your computer. Because of this, if sanitization does not take place, attackers can access that deleted data if they gain access to the device. Because of the nature of computers, it becomes a security threat to risk not sanitizing any machine which is believed to contain high-risk data. |
| 1. Practice Defense in Depth | Defense in depth is the idea that we cannot use one tool to solve all of our security needs. Because of this, it is preferable to use “multiple lines of defense” to solve our security needs. A simple example of the Defense in Depth in practice is all of the principles in this list work together to practice DiD. Multiple different security principles and tools work together to overlap each other's strengths and weaknesses. |
| 1. Use Effective Quality Assurance Techniques | In today's environment, cyber security should be thought about at every stage of QA because almost everything is connected to the internet now. Good QA cyber security practices would include standards and practices that effectively identify and eliminate potential vulnerabilities in an up-to-date manner. Because of the constant shifting of different threats, it is hard to make this concrete, but rigid testing for known vulnerabilities should be at the forefront of QA. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard will depend on the language and platform, but secure coding standards broadly are a set of rules and standards that help prevent, detect and eliminate known vulnerabilities. It would be wise to adopt a set of these standards to help keep your project secure. |

### 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-LLL] | Do not cast to an out-of-range enumeration value. |

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

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

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

| **Principles(s):** The principle here is validating input. If we don’t validate our inputs then we leave ourselves open for any kind of malicious attack. Here we do that by making sure our enumeration value is within our expected range. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **cast-integer-to-enum** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **LANG.CAST.COERCE**  **LANG.CAST.VALUE** | Coercion Alters Value  Cast Alters Value |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-INT50-a** | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **cast-integer-to-enum** | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-LLL] | Do not pass a nonstandard-layout type object across execution boundaries |

| **Noncompliant Code** |
| --- |
| This noncompliant code example assumes that there is a library whose header is library.h, an application (represented by application.cpp), and that the library and application are not ABI-compatible. Therefore, the contents of library.h constitute an execution boundary. A nonstandard-layout type object S is passed across this execution boundary. The application creates an instance of an object of this type, then passes a reference to the object to a function defined by the library, crossing the execution boundary. Because the layout is not guaranteed to be compatible across the boundary, this results in unexpected behavior. |
| // library.h  **struct** S {  **virtual** **void** f() { /\* ... \*/ }  };    **void** func(S &s); // Implemented by the library, calls S::f()    // application.cpp  #include "library.h"    **void** g() {    S s;    func(s);  } |

| **Compliant Code** |
| --- |
| Because the library and application do not conform to the same ABI, this compliant solution modifies the library and application to work with a standard-layout type. Furthermore, it also adds a static\_assert() to help guard against future code changes that accidentally modify S to no longer be a standard-layout type. |
| // library.h  #include <type\_traits>    **struct** S {  **void** f() { /\* ... \*/ } // No longer virtual  };  static\_assert(std::is\_standard\_layout<S>::value, "S is required to be a standard layout type");    **void** func(S &s); // Implemented by the library, calls S::f()    // application.cpp  #include "library.h"    **void** g() {    S s;    func(s);  } |

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

| **Principles(s):** The principle present here is again, data validation. By preventing object S from crossing the execution boundary we prevent it from having any kind of unexpected behavior which is what data validation is all about. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wdynamic-class-memaccess | Catches instances where the vtable pointer will be overwritten |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | C++4741, C++4742, C++4743 | NA |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork?_gl=1*wlbb2t*_ga*MTk3NzIxMTc2Ny4xNjcwNjc3MTQ3*_ga_87WECW6HCS*MTY3MDY3OTIxOC4yLjEuMTY3MDY3OTIxOS4wLjAuMA..) | 2022.3 | CERT.EXPR.PASS\_NON\_STD\_LAYOUT | NA |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | CERT\_CPP-EXP60-a | Do not pass a nonstandard-layout type object across execution boundaries |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-LLL] | Use valid references, pointers, 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):** The principles practiced here are Defense in Depth, Input Validation, Default Deny. In this example we practice all three of these principles by adding it to a piece of our defense puzzle (DiD), validatating that our pointers, references and iterators are correct (Input Validation) and denying anything else by default. (Default Deny) |
| --- |

**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) | 7.1p0 | **ALLOC.UAF** | Use After Free |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++4746, C++4747, C++4748, C++4749** | NA |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **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) | R2022b | [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-LLL] | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| Because the input is unbounded, the following code could lead to a buffer overflow. |
| #include <iostream>    **void** f() {  **char** buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The best solution for ensuring that data is not truncated and for guarding against buffer overflows is to use std::string instead of a bounded array, as in this compliant solution. |
| #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):** This policy follows the principle of input validation. Not only are we making sure that they are the correct type and in the correct range but we must also make sure that we have ample amount of space available. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stream-input-char-array** | Partially checked + soundly supported |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | No space for null terminator  Buffer overrun  Type overrun |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the 'char' buffer to store input from 'std::cin' |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-LLL] | 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 can be exploited to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #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):** In this standard Keep it Simple and Default Deny are both practiced. Because the deallocation and allocation of memory is far removed it is difficult to diagnose and as a result we only deallocate the memory when needed, this is a default deny practice. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **ALLOC.UAF** | Use after free |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | v7.5.0 | **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 |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **483 S, 484 S** | Partially implemented |

#### Coding Standard 6

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

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

| **Compliant Code** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution; 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):** This standard involves Input Validation, the principle of least privilege. In our example we are testing our constant variables to ensure that they are indeed constant and that they are not changing which would violate the principle of least privilege. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-LLL] | Catch exceptions by lvalue reference |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an object of type S is used to initialize the exception object that is later caught by an exception-declaration of type std::exception. The exception-declaration matches the exception object type, so the variable E is copy-initialized from the exception object, resulting in the exception object being sliced. Consequently, the output of this noncompliant code example is the implementation-defined value returned from calling std::exception::what() instead of "My custom exception". |
| #include <exception>  #include <iostream>    **struct** S : std::exception {  **const** **char** \*what() **const** noexcept override {  **return** "My custom exception";    }  };    **void** f() {  **try** {  **throw** S();    } **catch** (std::exception e) {      std::cout << e.what() << std::endl;    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the variable declared by the exception-declaration is an lvalue reference. The call to what() results in executing S::what() instead of std::exception::what(). |
| #include <exception>  #include <iostream>    **struct** S : std::exception {  **const** **char** \*what() **const** noexcept override {  **return** "My custom exception";    }  };    **void** f() {  **try** {  **throw** S();    } **catch** (std::exception &e) {      std::cout << e.what() << std::endl;    }  } |

**-----\*+**

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

| **Principles(s):** In this standard Input validation is practiced because we are validating that the S is being used correctly and we are making sure that our setting up of the exception test is correct as well. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **catch-class-by-value** | Fully checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-err61-cpp | Checked by clang-tidy; also checks for [VOID ERR09-CPP. Throw anonymous temporaries](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046576) by default |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **LANG.STRUCT.EXCP.CATCH**  **LANG.STRUCT.EXCP.THROW** | Use of catch  Use of throw |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **455 S** | Fully implemented |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Range Check** | [STD-008-LLL] | 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. |
| #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):** In this example Input Validation, the Principle of Least Privilege and Default Deny are practiced. This is because we are validating the range of our element to make sure that it only has accessed to what we want it to. Access outside of the range of the element should be denied by default. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **assert\_failure** | NA |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **LANG.MEM.BO** **LANG.MEM.BU** **LANG.MEM.TBA** **LANG.MEM.TO** **LANG.MEM.TU** | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the valid range |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Pointer Memory** | [STD-009-LLL] | Do not store an already-owned pointer value in an unrelated smart pointer |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, two unrelated smart pointers are constructed from the same underlying pointer value. When the local, automatic variable p2 is destroyed, it deletes the pointer value it manages. Then, when the local, automatic variable p1 is destroyed, it deletes the same pointer value, resulting in a double-free vulnerability. |
| #include <memory>    **void** f() {  **int** \*i = **new** **int**;    std::shared\_ptr<**int**> p1(i);    std::shared\_ptr<**int**> p2(i);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::shared\_ptr objects are related to one another through copy construction. When the local, automatic variable p2 is destroyed, the use count for the shared pointer value is decremented but still nonzero. Then, when the local, automatic variable p1 is destroyed, the use count for the shared pointer value is decremented to zero, and the managed pointer is destroyed. This compliant solution also calls std::make\_shared() instead of allocating a raw pointer and storing its value in a local variable. |
| #include <memory>    **void** f() {    std::shared\_ptr<**int**> p1 = std::make\_shared<**int**>();    std::shared\_ptr<**int**> p2(p1);  } |

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

| **Principles(s):** In this coding standard Input Validation, Architect and Design Policies, and Keep it Simple are the principles followed. This is because we are validating that we only have one pointer that isn’t sharing an underlying pointer, if we adopt design policies then this would be avoided and if our design policy follows a Keep it Simple protocol then there will never be a point in which we know there are two pointers causing a vulnerability. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **dangling\_pointer\_use** | NA |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MEM56** | NA |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-MEM56-a** | Do not store an already-owned pointer value in an unrelated smart pointer |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: MEM56-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem56cpp.html) | Checks for use of already-owned pointers (rule fully covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Close Files** | [STD-010-LLL] | 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):** In this coding standard Default Deny, Architect and Design Policies and the Principle of Least Privilege are the principles that are followed. This is because We make sure that files are closed when they are not in use, this follows the principle of least privilege. It also doubly follows Architect and Design policies and Default Deny because if we adopt a policy of Default Deny then closing files that don’t need to be open will adhere to the Default Deny Policy. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **ALLOC.LEAK** | Leak |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **RESOURCE\_LEAK (partial)** | Partially implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2022.3 | **C2701, C2702, C2703**  **C++2701, C++2702, C++2703** | NA |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **49 D** | Partially implemented |

### 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.

**Security is Everyones Job**

*Security is everyone’s job, “Don’t leave Security to the end,” is a moto that the team should live by if we are going to implement a DevSecOps culture. Simply put, instead of retrofitting security into our product after we have completed it, security should be a part of the project from the very beginning and at every step of the process. To automate these security standards they must be understood on some level by everyone and reconsulted at every level of pre-production and production.*

**Tools we can use**

*Visual Studio unit testing will be our go to for automated detection of security problems. The reason for choosing Visual Studio over other IDE’s is because of the vase amount of customization that we can perform to detect security problems depending on the problem we want to solve.*

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

### 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 | Encryption in rest is simply encrypting data in rest. Data in rest is data that has been stored on a computer hard and will be accessed later. Encryption in rest applies to Green Pace because the company will likely have number of computers with sensitive information on them. Applying encryption in rest will be following our earlier principles of Defense in Depth, the Principle of Least Privilege, and Default Deny. All these principles will help keep sensitive information out of the hands of unwanted agents. |
| Encryption at flight | Encryption at flight is encrypting data that is in transit. Data that is sent over the internet is first split into several different parts then scrambled using an encryption algorithm. Once it is sent to the receiver the receiver uses the *key* (which they will have received earlier) to decrypt the message. This type of encryption is important to Green Pace because as a large company they will be using several different types of communication back and forth with their customers and the need will arise to make sure that information remains secure while in transit. |
| Encryption in use | Encryption in use refers to encrypting data that is “in use” within the RAM while being accessed by a user. This possess a threat to the data present in the RAM but also data at rest and data in transit because it leaves open the possibility that the RAM could be parsed and access to the encryption key enabled. Because Green Pace will access all kinds of data on many different devices, it is emparitive that an Encryption in Use policy be enabled. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication provides a way of identifying a user, this is usually done by username and passwords and there is no reason Green Pace shouldn’t take advantage of the first leg of Triple A. Usernames and passwords will allow us to check the usernames and passwords against the ones we have stored in our database and grant or deny access to the network. This prevents just anyone accessing our network and is the first step in Triple A. |
| Authorization | Once the user has logged in they will either have the authority to perform certain actions or not. This follows a principle of least privilege and default deny principles. If the user doesn’t need access to a specific system to complete their job task then they should be denied access to it by default. At Green Pace sales associates shouldn’t have access to the accounting and payroll systems and accounting shouldn’t have access to IT tools. This limited access can only work to increase the security of Green Pace’s network. |
| Accounting | Accounting is simply monitoring what resources a user accesses during the time they were logged in to the system. This can include things such as time spent on the network, messages sent, payments made or received, files accessed. etc. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 12/10/2022 | Policy Alpha | Dale Ayers | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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