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



# Green Pace Secure Development Policy

# Kelly Illescas

## 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 | Validating input data defines a standard by which all input can be tested. In this way, only the information desired can be input into a particular program. Specific parameters can be developed to be sure all data input by any user falls within the specifications of the program or system. |
| 1. Heed Compiler Warnings | A compiler can detect a bug in the system, yet still compile because it does not interfere with the program. However, it can lead to more issues that are difficult to discover in the programming down the line. Therefore, it is important to heed all compiler warnings and try to fix the problems being brought to light as soon as possible so new errors are easier to fix in the long run. |
| 1. Architect and Design for Security Policies | Security architecture and design is a set of policies defined for a program that is used to identify potential risks. Additionally, the policies are used to address any necessities for the program to be run correctly. These policies include relationships and dependencies, benefits of control re-use, form, and drivers such as risk management, good practice, financial considerations, and legal and regulatory policies. |
| 1. Keep It Simple | In any program, simplicity is key to making the program run efficiently and smoothly. To ensure that a user finds a program uncomplicated, avoiding complexity is very important. |
| 1. Default Deny | By default, access to a program will be denied. A user who has permission and can enter the proper credentials will be allowed to access the program. |
| 1. Adhere to the Principle of Least Privilege | When a hacker would like to access particular data necessary to infiltrate a system, it is in the program’s best interest to only give access permission to what is needed at the lowest level. That way, the hacker may not be able to access parts of the program with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Hackers are able to use SQL injection when data is not sanitized. To sanitize, all unnecessary data that is sent from one system should be permanently destroyed so it cannot be recovered by a hacker. |
| 1. Practice Defense in Depth | Defense in depth dictates that there should be many security formats layered atop one another. These layers make it more difficult for someone to hack into a system, because even if the hacker gets past one layer, there are many more behind it that can offer more and better protection. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance means designing tests that will assist in finding program bugs to help prevent hacking. Some particular tests that can be used in programming are unit testing, component testing, integration testing, end-to-end testing, performance testing, regression testing, sanity testing, and system testing. |
| 1. Adopt a Secure Coding Standard | Secure coding standards are ways to prevent vulnerabilities in programs. These standards include guidelines that aid in reducing errors and other security vulnerabilities during development. |

### 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] | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| Copying data to a buffer that is not large enough to hold that data results in a buffer overflow. Buffer overflows occur frequently when manipulating strings. |
| #include <iostream>    **void** f() {  **char** buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The best solution for guarding against buffer overflows is to use std::string. |
| #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):** Copying string data to a buffer that is too small to hold that data results in a buffer overflow. Attackers can exploit this condition to execute arbitrary code with the permissions of the vulnerable process. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | stream-input-char-array | Partially checked + soundly supported |
| CodeSonar | 7.3p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Helix QAC | 2023.1 | C++5216  DF2835, DF2836, DF2839 |  |
| Klocwork | 2023.1 | NNTS.MIGHT  NNTS.TAINTED  NNTS.MUST  SV.UNBOUND\_STRING\_INPUT.CIN |  |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP\_STR50-b  CERT\_CPP\_STR50-c  CERT\_CPP\_STR50-e  CERT\_CPP\_STR50-f  CERT\_CPP\_STR50-g | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the ‘char’ buffer to store input from ‘std::cin’ |
| Polyspace Bug Finder | R2023a | CERT C++:STR50-CPP | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation * Insufficient destination buffer size   Rule partially covered. |
| RuleChecker | 22.10 | stream-input-char-array | Partially checked |
| SonarQube C/C++ Plugin | 4.10 | S3519 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Ensure that integer conversions do not result in lost or misinterpreted data |

| **Noncompliant Code** |
| --- |
| Type range errors, including loss of data (truncation) and loss of sign (sign errors), can occur when converting from a value of an unsigned integer type to a value of a signed integer type. |
| #include <limits.h>    **void** func(**void**) {    unsigned **long** **int** u\_a = ULONG\_MAX;  **signed** **char** sc;    sc = (**signed** **char**)u\_a; /\* Cast eliminates warning \*/    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Validate ranges when converting from unsigned to signed. The following solution converts a value of unsigned long int type to a value of signed char type: |
| #include <limits.h>    **void** func(**void**) {    unsigned **long** **int** u\_a = ULONG\_MAX;  **signed** **char** sc;  **if** (u\_a <= SCHAR\_MAX) {      sc = (**signed** **char**)u\_a;  /\* Cast eliminates warning \*/    } **else** {      /\* Handle error \*/    }  } |

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

| **Principles(s):** Integer truncation errors can lead to buffer overflows and the execution of arbitrary code by an attacker. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4. 10.6 and 10.7 |
| CodeSonar | 7.3p0 | LANG.CAST.PC.AV  LANG.CAST.PC.CONST2PTR  LANG.CAST.PC.INT  LANG.CAST.COERCE  LANG.CAST.VALUE  ALLOC.SIZE.TRUNC  MISC.MEM.SIZE.TRUNC  LANG.MEM.TBA | Cast: arithmetic type/void pointer  Conversion: integer constant to pointer  Conversion: pointer/integer  Coercion alters value  Cast alters value  Truncation of allocation size  Truncation of size  Tainted buffer access |
| Compass/ROSE |  |  | Can detect violations of this rule. However, false warnings may be raised if limits.h is included |
| Cppcheck | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |
| Helix QAC | 2023.1 | **C2850, C2855, C2890, C2895, C2900, C2905,**  **C++2850, C++2855, C++2890, C++2895, C++2900, C++2905,  C++3000, C++3010**  **DF2851, DF2852, DF2853,  DF2856, DF2857, DF2858, DF2891, DF2892, DF2893, DF2896, DF2897, DF2898, DF2901, DF2902, DF2903, DF2906, DF2907, DF2908** |  |
| Klocwork | 2023.1 | PORTING.CAST.SIZE |  |
| LDRA tool suite | 9.7.1 | 93 S, 433 S, 434 S | Partially implemented |
| Parasoft C/C++ test | 2022.2 | **CERT\_C-INT31-a CERT\_C-INT31-b CERT\_C-INT31-c CERT\_C-INT31-d CERT\_C-INT31-e CERT\_C-INT31-f CERT\_C-INT31-g CERT\_C-INT31-h CERT\_C-INT31-i CERT\_C-INT31-j CERT\_C-INT31-k CERT\_C-INT31-l CERT\_C-INT31-m CERT\_C-INT31-n CERT\_C-INT31-o** | An expression of essentially Boolean type should always be used where an operand is interpreted as a Boolean value An operand of essentially Boolean type should not be used where an operand is interpreted as a numeric value An operand of essentially character type should not be used where an operand is interpreted as a numeric value An operand of essentially enum type should not be used in an arithmetic operation Shift and bitwise operations should not be performed on operands of essentially signed or enum type An operand of essentially signed or enum type should not be used as the right hand operand to the bitwise shifting operator An operand of essentially unsigned type should not be used as the operand to the unary minus operator The value of an expression shall not be assigned to an object with a narrower essential type The value of an expression shall not be assigned to an object of a different essential type category Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category The second and third operands of the ternary operator shall have the same essential type category The value of a composite expression shall not be assigned to an object with wider essential type If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type If a composite expression is used as one (second or third) operand of a conditional operator then the other operand shall not have wider essential type Avoid integer overflows |
| Polyspace Bug Finder | R2023a | CERT C: Rule INT31-C | Checks for:   * Integer conversion overflow * Call to memset with unintended value * Sign change integer conversion overflow * Tainted sign change conversion * Unsigned integer conversion overflow   Rule partially covered. |

#### Coding Standard 3

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

| **Noncompliant Code** |
| --- |
| In this example, input is copied into std::string, replacing semicolon characters with spaces. This 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** |
| --- |
| Below, 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):** Using an invalid reference, pointer, or iterator to a string object could allow an attacker to run arbitrary code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | ALLOC.UAF | Use After Free |
| Helix QAC | 2023.1 | DF746, DF747, DF748, DF749 | [Insert text.] |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-STR52-a | Use valid references, pointers, and iterators to reference elements of a basic\_string |
| Polyspace Bug Finder | R2023a | CERT C++:STR52-CPP | Checks for use of invalid string iterator (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Sanitize data passed to complex subsystems |

| **Noncompliant Code** |
| --- |
| This example inputs an email address to a buffer and then uses a string as an argument in a call to system(): |
| **sprintf**(buffer, "/bin/mail %s < /tmp/email", addr);  **system**(buffer); |

| **Compliant Code** |
| --- |
| This example exemplifies the whitelisting approach, which defines a list of acceptable characters of manageable size: |
| **static** **char** ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"                           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"                           "1234567890\_-.@";  **char** user\_data[] = "Bad char 1:} Bad char 2:{";  **char** \*cp = user\_data; /\* Cursor into string \*/  **const** **char** \*end = user\_data + **strlen**( user\_data);  **for** (cp += **strspn**(cp, ok\_chars); cp != end; cp += **strspn**(cp, ok\_chars)) {    \*cp = '\_';  } |

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

| **Principles(s):** Failure to [sanitize](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-sanitize) data passed to a complex subsystem can lead to an injection attack, data integrity issues, and a loss of sensitive data. |
| --- |

**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=87152428) | 22.04 |  | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **IO.INJ.COMMAND** **IO.INJ.FMT** **IO.INJ.LDAP** **IO.INJ.LIB** **IO.INJ.SQL** **IO.UT.LIB** **IO.UT.PROC** | Command injection Format string injection LDAP injection Library injection SQL injection Untrusted Library Load Untrusted Process Creation |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | **TAINTED\_STRING** | Fully implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2023.1 | **NNTS.TAINTED** **SV.TAINTED.INJECTION** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **108 D, 109 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-STR02-a** **CERT\_C-STR02-b** **CERT\_C-STR02-c** | Protect against command injection Protect against file name injection Protect against SQL injection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rec. STR02-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.str02c.html) | Checks for:   * Execution of externally controlled command * Command executed from externally controlled path * Library loaded from externally controlled path   Rec. partially covered. |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| *S* is dereferenced after it has been deallocated, allowing the code to be run arbitrarily. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| *S* is now deallocated after 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):** Reading previously dynamically allocated memory after it has been deallocated can lead to [abnormal program termination](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-abnormaltermination) and [denial-of-service attacks](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-denial-of-service). Writing memory that has been deallocated can lead to the execution of arbitrary code with the permissions of the vulnerable process. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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 |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++4303, C++4304** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2023.1 | **UFM.DEREF.MIGHT** **UFM.DEREF.MUST** **UFM.FFM.MIGHT** **UFM.FFM.MUST** **UFM.RETURN.MIGHT** **UFM.RETURN.MUST** **UFM.USE.MIGHT** **UFM.USE.MUST** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **483 S, 484 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: MEM50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem50cpp.html) | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | **4303, 4304** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.24 | [**V586**](https://pvs-studio.com/en/docs/warnings/v586/), [V774](https://pvs-studio.com/en/docs/warnings/v774/) |  |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| This example uses assert() 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** |
| --- |
| In this solution, a preprocessor conditional statement is used: |
| **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):** Static assertion is a valuable diagnostic tool for finding and eliminating software defects that may result in [vulnerabilities](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability) at compile time. The absence of static assertions, however, does not mean that code is incorrect. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** |  |
| [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) | 7.3p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/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 |
| [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-CPP] | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| Here, 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** |
| --- |
| Here, main() handles all exceptions, allowing for 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):** Allowing the application to abnormally terminate can lead to resources not being freed, closed, and so on. It is frequently a vector for denial-of-service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **main-function-catch-all** **early-catch-all** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR51** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **MISRA.CATCH.ALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **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 |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | **4035, 4036, 4037** |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all** **early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Namespaces | [STD-008-CPP] | Do not modify the standard namespaces |

| **Noncompliant Code** |
| --- |
| Here, the declaration of x is added to the namespace std, resulting in undefined behavior. |
| **namespace** std {  **int** x;  } |

| **Compliant Code** |
| --- |
| The declaration of x can be placed into a nonstandard namespace (without a reserved name) |
| **namespace** nonstd {  **int** x;  } |

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

| **Principles(s):** Altering the standard namespace can cause undefined behavior in the C++ standard library. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL58** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.STRUCT.DECL.SNM** | Modification of Standard Namespaces |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++3180, C++3181, C++3182** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) |  | **CERT.DCL.STD\_NS\_MODIFIED** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-DCL58-a** | Do not modify the standard namespaces 'std' and 'posix' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: DCL58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl58cpp.html) | Checks for modification of standard namespaces (rule fully covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **4032, 4035, 4631** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.24 | [V1061](https://pvs-studio.com/en/docs/warnings/v1061/) |  |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [S3470](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3470) |  |

#### Coding Standard 9

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

| **Noncompliant Code** |
| --- |
| In this example, a pointer to an object is used to call a non0static member function of the object prior to the beginning of the pointer’s lifetime, resulting in undefined behavior. |
| **struct** S {  **void** mem\_fn();  };    **void** f() {    S \*s;    s->mem\_fn();  } |

| **Compliant Code** |
| --- |
| Here, 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):** Referencing an object outside of its lifetime can result in an attacker being able to run arbitrary code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **return-reference-local** **dangling\_pointer\_use** | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | **-Wdangling-initializer-list** | Catches some lifetime issues related to incorrect use of std::initializer\_list<> |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **IO.UAC** **ALLOC.UAF** | Use after close Use after free |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++4003, C++4026**  **DF2812, DF2813, DF2814, DF2930, DF2931, DF2932, DF2933, DF2934,** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **CL.FFM.ASSIGN** **CL.FFM.COPY** **LOCRET.ARG** **LOCRET.GLOB** **LOCRET.RET** **UFM.DEREF.MIGHT** **UFM.DEREF.MUST** **UFM.FFM.MIGHT** **UFM.FFM.MUST** **UFM.RETURN.MIGHT** **UFM.RETURN.MUST** **UFM.USE.MIGHT** **UFM.USE.MUST** **UNINIT.HEAP.MIGHT** **UNINIT.HEAP.MUST** **UNINIT.STACK.ARRAY.MIGHT** **UNINIT.STACK.ARRAY.MUST** **UNINIT.STACK.ARRAY.PARTIAL.MUST** **UNINIT.STACK.MIGHT** **UNINIT.STACK.MUST** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **42 D, 53 D, 77 D, 1 J, 71 S, 565 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **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 |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: EXP54-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp54cpp.html) | 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   Rule partially covered. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| OOP Handling | [STD-010-CPP] | Do not invoke virtual functions from constructors or destructors |

| **Noncompliant Code** |
| --- |
| Through calls to virtual functions from the constructor and destructor, the base class attempts to seize and release an object’s resources. However, the B::B() constructor calls B::seize() rather than D::seize(), and the B::~B() destructor calls B::release() rather than D::release(). |
| **struct** B {    B() { seize(); }  **virtual** ~B() { release(); }    **protected**:  **virtual** **void** seize();  **virtual** **void** release();  };    **struct** D : B {  **virtual** ~D() = **default**;    **protected**:  **void** seize() override {      B::seize();      // Get derived resources...    }    **void** release() override {      // Release derived resources...      B::release();    }  }; |

| **Compliant Code** |
| --- |
| For each class to seize and release its own resources, constructors and destructors call a nonvirtual, private function (\_mine) instead of calling a virtual function. |
| **class** B {  **void** seize\_mine();  **void** release\_mine();    **public**:    B() { seize\_mine(); }  **virtual** ~B() { release\_mine(); }    **protected**:  **virtual** **void** seize() { seize\_mine(); }  **virtual** **void** release() { release\_mine(); }  };    **class** D : **public** B {  **void** seize\_mine();  **void** release\_mine();    **public**:    D() { seize\_mine(); }  **virtual** ~D() { release\_mine(); }    **protected**:  **void** seize() override {      B::seize();      seize\_mine();    }    **void** release() override {      release\_mine();      B::release();    }  }; |

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

| **Principles(s):** |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **virtual-call-in-constructor** **invalid\_function\_pointer** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-OOP50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-alpha.cplusplus.VirtualCall | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.STRUCT.VCALL\_IN\_CTOR**  **LANG.STRUCT.VCALL\_IN\_DTOR** | Virtual Call in Constructor  Virtual Call in Destructor |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++4260, C++4261, C++4273, C++4274, C++4275, C++4276, C++4277, C++4278, C++4279, C++4280, C++4281, C++4282** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2023.1 | **CERT.OOP.CTOR.VIRTUAL\_FUNC** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **467 S, 92 D** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-OOP50-a** **CERT\_CPP-OOP50-b** **CERT\_CPP-OOP50-c** **CERT\_CPP-OOP50-d** | Avoid calling virtual functions from constructors Avoid calling virtual functions from destructors Do not use dynamic type of an object under construction Do not use dynamic type of an object under destruction |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: OOP50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop50cpp.html) | Checks for virtual function call from constructors and destructors (rule fully covered) |

### 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.

I feel that Green Pace’s established plan for security in the development process is sufficient for taking care of a program’s design, build, analysis, and maintenance. To automate the processes defined by the standards in this risk assessment, we would simply need to build a system of checks into the process using the tools mentioned in the tables above. This system could implement the checks during both the build and test stages.

### 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 | Likely | Medium | P18 | L1 |
| STD-002-CPP | High | Probable | High | P6 | L2 |
| STD-003-CPP | High | Probable | High | P6 | L2 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-009-CPP | High | Probable | High | P6 | L2 |
| STD-010-CPP | Low | Unlikely | Medium | P2 | L3 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest applies to data that is stored on a hard disk or solid-state drive. This type of encryption forces a hacker to either possess the encryption keys or hack the system in order to bypass encryption. Anti-virus software and firewalls are not enough to protect all sensitive data; therefore, using encryption at rest, such as with an algorithm that needs a decryption key to open it, is ideal to add another layer to protect from malicious intent. |
| Encryption in flight | Encryption in flight is encryption of data that moves over a network. If data is not encrypted, intruders can monitor or intercept that data, and use it for malicious purposes. Therefore, any sensitive data that is sent must be encrypted using SSL (Secure Sockets Layer) or PGP (Pretty Good Privacy). |
| Encryption in use | Encryption in use refers to encrypting data that is currently in use. This is perhaps the most critical to protect, as data is often accessed by users or applications. This type of encryption would be ideal because it can encrypt and decrypt data in almost real time, allowing a user to view information in plain text while it is actively being encrypted on the back end. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication occurs when a user has entered the system. Users are given certain permissions across sites and programs, and excluded from others. |
| Authorization | Authorization is used to ensure a user is who he or she claims to be through use of username/password, and even more enhanced security features such as two-factor authentication. |
| Accounting | Accounting is a valuable tool that measures what has been accessed or modified, by whom, and when it occurred. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 03/19/2023 | Coding Standards Revision | Kelly Illescas |  |
| 2.0 | 04/07/2023 | Automation Testing Revision | Kelly Illescas |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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