**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 | Validating user input means only accepting expected inputs, and rejecting anything else. Allowing unlimited user input can cause buffer overflows or SQL injections. A system should only read what it needs to read and nothing more. Following this basic principle will eliminate a large portion of simple threats. It is important to be distrustful of external data sources. |
| 1. Heed Compiler Warnings | Compilers are designed to help you write better and more secure code. There are warnings included that may be able to detect common threats or security flaws in your code. Using the highest possible security warning setting on the compiler will help to avoid common vulnerabilities. |
| 1. Architect and Design for Security Policies | It is important to design code with security in mind from conception. If you write your code first and try to add security second it will often leave holes and weaknesses. Creating an architecture of security from the first phase of production will help to enhance the program’s strength. |
| 1. Keep It Simple | Keeping code simple helps to avoid mistakes. Making things too complicated opens the possibility of forgetting to cover all your bases. The more functions and variables the more exploits are possible. Keeping code simple makes security simple. |
| 1. Default Deny | The default security setting for any program should be to deny access. A person or a system attempting to gain access to a system should have to prove they have permission, rather than convince you they are simply not a threat. Making sure that permission is required to access a system will make it more secure. |
| 1. Adhere to the Principle of Least Privilege | When giving privilege to a person or a system, the least amount of privilege possible should be given. Only a privilege that is absolutely necessary for task execution should be given, and if possible it should be taken away when the task is complete. Keeping privilege levels low lowers the chance of unauthorized access to sensitive information. |
| 1. Sanitize Data Sent to Other Systems | It is important only to send the necessary data when transmitting data to other systems. If more data is sent than required, it could be used to access protected functionality when it is sent back. The original sender is responsible for sanitizing their data as it leaves the protected area of the main system |
| 1. Practice Defense in Depth | Defense in depth is the practice of using multiple layers of security rather than one. There is a balance between the number of layers and the speed of performance, but it is important to use more than one. Using layered defense prevents a major intrusion if one security layer fails. |
| 1. Use Effective Quality Assurance Techniques | Effective quality assurance means testing your code well and often. Performing different tests using different teams, as well as external bodies, can help to make your code more secure. The more people who try to break the code the more certain you can be that the code is secure. |
| 1. Adopt a Secure Coding Standard | Creating a standard for security in coding will help to make sure that the best end product is put forth. There should never be a “we’ll get to it later” mentality. Secure coding should be on the main checklist for every step of production. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### 

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not confuse data types and use the correct type function. |

| **Noncompliant Code** |
| --- |
| The strncpy function does not consider the potential for null bytes in each wide character string. The string copy function assumes a narrow string type. This may result in premature termination of the copy. |
| #include <stddef.h>  #include <string.h>    **void** func(**void**) {  **wchar\_t** wide\_str1[]  = L"0123456789";  **wchar\_t** wide\_str2[] =  L"0000000000";    **strncpy**(wide\_str2, wide\_str1, 10);  } |

| **Compliant Code** |
| --- |
| Using the appropriate function call for the appropriate data type will prevent a buffer overflow. |
| #include <string.h>  #include <wchar.h>    **void** func(**void**) {  **wchar\_t** wide\_str1[] = L"0123456789";  **wchar\_t** wide\_str2[] = L"0000000000";    /\* Use of proper-width function \*/    wcsncpy(wide\_str2, wide\_str1, 10);    **char** narrow\_str1[] = "0123456789";  **char** narrow\_str2[] = "0000000000";    /\* Use of proper-width function \*/  **strncpy**(narrow\_str2, narrow\_str1, 10);  } |

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

| **Principles(s):** Keep it simple and validate user data. Making sure you are suing simple, accurate data types can help avoid attacks. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | P27 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | **wide-narrow-string-cast** **wide-narrow-string-cast-implicit** | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-STR38** | Fully implemented |
| Clang | 3.9 | -Wincompatible-pointer-types |  |
| CodeSonar | 7.3p0 | **LANG.MEM.BO** **LANG.MEM.TBA** | Buffer Overrun Tainted Buffer Access |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Use correct integer precisions |

| **Noncompliant Code** |
| --- |
| This code shows raising 2 to the power of the function argument. If the code is used incorrectly the number could be too large, causing an injection. |
| #include <limits.h>    unsigned **int** pow2(unsigned **int** **exp**) {  **if** (**exp** >= **sizeof**(unsigned **int**) \* CHAR\_BIT) {      /\* Handle error \*/    }  **return** 1 << **exp**;  } |

| **Compliant Code** |
| --- |
| The precision function utilizes popcount, which will only allow the appropriate number of bits to be accessed. This prevents an overflow. |
| #include <stddef.h>  #include <stdint.h>  #include <limits.h>  **extern** **size\_t** popcount(uintmax\_t);  #define PRECISION(umax\_value) popcount(umax\_value)  unsigned **int** pow2(unsigned **int** **exp**) {  **if** (**exp** >= PRECISION(UINT\_MAX)) {      /\* Handle error \*/    }  **return** 1 << **exp**;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard and validate user input. By avoiding integer overflow we prevent attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 |  | Supported: Astrée reports overflows due to insufficient precision. |
| CodeSonar | 7.3p0 | **LANG.ARITH.BIGSHIFT** | Shift Amount Exceeds Bit Width |
| Helix QAC | 2023.1 | **C0582**  **C++3115** |  |
| Parasoft C/C++test | 2022.2 | **CERT\_C-INT35-a** | Use correct integer precisions when checking the right hand operand of the shift operator |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do attempt to modify string literals. |

| **Noncompliant Code** |
| --- |
| Trying to modify a string literal is undefined behavior. By creating the char pointer str is initialized to a string literal address, which creates an error. |
| **char** \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| Declaring the str array as a char type allows the array to be modified. |
| **char** str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):** Heed Compiler Warnings. Following default error warnings in the compiler will help prevent error crashes. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | **string-literal-modfication** **write-to-string-literal** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-STR30** | Fully implemented |
| Compass/ROSE |  |  | Can detect simple violations of this rule |
| Coverity | 2017.07 | **PW** | Deprecates conversion from a string literal to "char \*" |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Reset strings on fgets() or fgetws() failure |

| **Noncompliant Code** |
| --- |
| Forgetting to reset the string leave potential information in the buffer, which might be used for an SQL injection attack. |
| #include <stdio.h>    **enum** { BUFFER\_SIZE = 1024 };  **void** func(**FILE** \*file) {  **char** buf[BUFFER\_SIZE];    **if** (**fgets**(buf, **sizeof**(buf), file) == NULL) {      /\* Set error flag and continue \*/    }  } |

| **Compliant Code** |
| --- |
| Resetting the buffer means that no undesired information is left behind after a failed function call. |
| #include <stdio.h>    **enum** { BUFFER\_SIZE = 1024 };    **void** func(**FILE** \*file) {  **char** buf[BUFFER\_SIZE];    **if** (**fgets**(buf, **sizeof**(buf), file) == NULL) {      /\* Set error flag and continue \*/      \*buf = '\0';    }  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques and Architect and Design for Security Policies. Knowing that a buffer overflow can lead to an injection will help write better code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.MEM.UVAR** | Uninitialized Variable |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **DF4861, DF4862, DF4863** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-FIO40-a** | Reset strings on fgets() or fgetws() failure |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Detect and handle memory allocation errors. |

| **Noncompliant Code** |
| --- |
| Memory allocation is not checked, so no error will be thrown if the memory allocation fails. |
| #include <cstring>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** **int**[size];    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

| **Compliant Code** |
| --- |
| Adding an allocation check ensures proper memory allocation and protection. |
| #include <cstring>  #include <new>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** (std::**nothrow**) **int**[size];  **if** (!copy) {      // Handle error  **return**;    }    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

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

| **Principles(s):** Architect and Design for Security Policies and Heed Compiler Warnings. Applying best coding practices will leave you less vulnerable to memory error attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **45 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-MEM52-a** **CERT\_CPP-MEM52-b** | Check the return value of new Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| [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++: MEM52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem52cpp.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |

#### 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** |
| --- |
| Assertions should be placed in a function and called, not directly used in a structure such as this. |
| #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** |
| --- |
| Static assertions allow incorrect assumptions to be diagnosed at runtime. |
| #include <assert.h>    **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    static\_assert(**sizeof**(**struct** timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**),                "Structure must not have any padding"); |

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

| **Principles(s):** Adopt a Secure Coding Standard and Use Effective Quality Assurance Techniques. Testing for security threats is essential. |
| --- |

**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 |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| No exception is caught so the program terminates. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| The try catch loop handle exceptions, allowing the program to continue. |
| **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):** Keep it Simple and Default Deny. Anytime an exception is thrown it needs to be handled. This is a simple way to deny access to an unwanted party. |
| --- |

**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** |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Containers** | [STD-008-CPP] | Guarantee that container indices and iterators are within the valid range. |

| **Noncompliant Code** |
| --- |
| The size of pos is not declared correctly, so it is possible for a negative value to be entered. |
| #include <cstddef>    **void** insert\_in\_table(**int** \*table, std::**size\_t** tableSize, **int** pos, **int** value) {  **if** (pos >= tableSize) {      // Handle error  **return**;    }    table[pos] = value;  } |

| **Compliant Code** |
| --- |
| Pos is declared as size\_t, preventing errors. |
| #include <cstddef>    **void** insert\_in\_table(**int** \*table, std::**size\_t** tableSize, std::**size\_t** pos, **int** value) {  **if** (pos >= tableSize) {      // Handle error  **return**;    }    table[pos] = value;  } |

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

| **Principles(s):** ValidateInput Data and Architect and Design for Security Policies. Allowing a user to input outside the proper range is a high likelihood for attack. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **45 D, 47 S, 476 S, 489 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-CTR50-a** | Guarantee that container indices are within the valid range |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: CTR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcctr50cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2891, 3139, 3140** |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **OOP** | [STD-009-CPP] | Do not delete a polymorphic object without a virtual destructor |

| **Noncompliant Code** |
| --- |
| The destructor is implicitly declared which does not make it virtual. |
| **struct** Base {  **virtual** **void** f();  };    **struct** Derived : Base {};    **void** f() {    Base \*b = **new** Derived();    // ...  **delete** b;  } |

| **Compliant Code** |
| --- |
| The virtual destructor is implied explicitly. |
| **struct** Base {  **virtual** ~Base() = **default**;  **virtual** **void** f();  };    **struct** Derived : Base {};    **void** f() {    Base \*b = **new** Derived();    // ...  **delete** b;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques and Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **non-virtual-public-destructor-in-non-final-class** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-OOP52** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wdelete-non-virtual-dtor |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.STRUCT.DNVD** | delete with Non-Virtual Destructor |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Random Numbers** | [STD-010-CPP] | Ensure your random number generator is properly seeded. |

| **Noncompliant Code** |
| --- |
| The default seed is used in this example so the random numbers will be the same every time. |
| #include <random>  #include <iostream>    **void** f() {    std::mt19937 engine;    **for** (**int** i = 0; i < 10; ++i) {      std::cout << engine() << ", ";    }  } |

| **Compliant Code** |
| --- |
| This example uses a random number as the seed, which creates a random pattern on each call. |
| #include <random>  #include <iostream>    **void** f() {    std::random\_device dev;    std::mt19937 engine(dev());    **for** (**int** i = 0; i < 10; ++i) {      std::cout << engine() << ", ";    }  } |

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

| **Principles(s):** Architect and Design for Security Policies and Keep it Simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **default-construction** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MSC51** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **HARDCODED.SEED** **MISC.CRYPTO.TIMESEED** | Hardcoded Seed in PRNG Predictable Seed in PRNG |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-MSC51-a** | Properly seed pseudorandom number generators |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

Automation can be a useful tool in assisting with the DevOps process. During pre-production automation can be useful in both building as well as verifying and testing development. Automation can assist in these steps by adhering to the principles of Architect and Design for Security Policies and Using Effective Quality Assurance Techniques. Feeding pseudocode into an automated checker can flag potential security risk or flawed designs. Using an automated unit tester can also help in early and pre-production stages to find bugs or security omissions.

In production, automation is especially helpful in the monitor and detect as well as the maintain and stabilize phases. Automation helps in these phases to follow the principles of ValidateInput Data and Practicing Defense in Depth. Automation can help to detect data overflows or SQL injections. It can also throw additional errors not necessarily coded in production. Additionally, an automated security system adds multiple levels of security.

### 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 | Low | P27 | L1 |
| STD-002-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-003-CPP | Low | Likely | Low | P9 | L2 |
| STD-004-CPP | Low | Probable | Medium | P$ | L3 |
| 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 | Likely | High | P9 | L2 |
| STD-009-CPP | Low | Likely | Low | P9 | L2 |
| STD-010-CPP | Medium | Likely | Low | P18 | L1 |

### 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 the encryption of static data. Static data is data that is stored on a personal hard drive or on a massive database. These static data centers need to be encrypted to avoid access. If an attacker were able to gain access they could have the information personal to one or on billion people. |
| Encryption at flight | Encryption at flight is the encryption of data as it is transmitted from one place to another. When a file is transferred that data is vulnerable to being “pulled out of the air.” If the data is not encrypted, anyone who can access the transmission stream can steal that information. Encrypting moving data is a major form of security. |
| Encryption in use | Encryption in use is the encryption of current, developing, or created information. Something, like this paper, that a person is making right there. For example, if someone has access to keylogging software or screen sharing software they can intercept the data before it is stored or transmitted. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication should be an important step in allowing access to a system. This would allow the company to know who is accessing their system and when. This is an important time to follow the principle of Default Deny. |
| Authorization | Authorization within a system needs to be limited. Following the principle of Adhere to the Principle of Least Privilege makes sure that as few people as possible have access to information or permissions. |
| Accounting | Accounting is important in figuring out what went wrong so it can be fixed. The person who accessed the system should have had to leave a trail behind. If a company can’t gather who the attacker was or where they were accessing data from, they are not properly accounting. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

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

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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