**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 6](#_Toc52464060)

[Coding Standard 3 7](#_Toc52464061)

[Coding Standard 4 9](#_Toc52464062)

[Coding Standard 5 11](#_Toc52464063)

[Coding Standard 6 12](#_Toc52464064)

[Coding Standard 7 13](#_Toc52464065)

[Coding Standard 8 14](#_Toc52464066)

[Coding Standard 9 15](#_Toc52464067)

[Coding Standard 10 16](#_Toc52464068)

[Defense-in-Depth Illustration 17](#_Toc52464069)

[Project One 17](#_Toc52464070)

[1. Revise the C/C++ Standards 17](#_Toc52464071)

[2. Risk Assessment 17](#_Toc52464072)

[3. Automated Detection 17](#_Toc52464073)

[4. Automation 17](#_Toc52464074)

[5. Summary of Risk Assessments 18](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 18](#_Toc52464076)

[7. Map the Principles 19](#_Toc52464077)

[Audit Controls and Management 20](#_Toc52464078)

[Enforcement 20](#_Toc52464079)

[Exceptions Process 20](#_Toc52464080)

[Distribution 21](#_Toc52464081)

[Policy Change Control 21](#_Toc52464082)

[Policy Version History 21](#_Toc52464083)

[Appendix A Lookups 21](#_Toc52464084)

[Approved C/C++ Language Acronyms 21](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles for 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 ensures that only properly formatted data enters the system, preventing issues downstream. It should occur as early as possible in the data flow, ideally upon receipt from an external party. |
| 1. Heed Compiler Warnings | To ensure the security and efficiency of your code, it's important to compile it using the highest warning level available for your compiler. By modifying the code to eliminate warnings and utilizing static and dynamic analysis tools, you can effectively detect and address any security flaws. |
| 1. Architect and Design for Security Policies | When designing your software architecture, it's crucial to effectively implement and enforce security policies. This involves dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set, to accommodate varying levels of access as needed. |
| 1. Keep It Simple | Simplicity in design is crucial to minimize errors during implementation, configuration, and use. As security mechanisms become more complex, the effort required to achieve an appropriate level of assurance increases dramatically. |
| 1. Default Deny | Make access decisions based on permission rather than exclusion. By default, access is denied, and the protection scheme specifies the conditions under which access is granted. This approach ensures a proactive and secure access control system. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege principle works by allowing only enough access to perform the required job. This principle reduces the risk of attackers gaining access to critical systems or sensitive data by compromising a user account, device, or application. |
| 1. Sanitize Data Sent to Other Systems | It is crucial to make sure that when data is passed through systems, particularly complex ones, you sanitize the data beforehand. This ensures that someone cannot use a simple injection attack to access and perform commands outside of the code’s intended purpose. |
| 1. Practice Defense in Depth | Defense-in-depth provides multiple, redundant defensive measures in case a security control fails or a vulnerability is exploited. This involves layering different security measures to optimize the protection of web properties, databases, and other resources. |
| 1. Use Effective Quality Assurance Techniques | Using Effective Quality Assurance techniques will increase the likelihood of identifying and later eliminating bugs in your code that may potentially be, or are, security weaknesses. By utilizing multiple testing phases, having colleagues look over your code, and sending it to third parties to inspect, you can ensure that your system has added security. |
| 1. Adopt a Secure Coding Standard | Secure coding standards are used to prevent security vulnerabilities. Used effectively, these security standards prevent, detect, and eliminate errors that could compromise software security. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Obey the one-definition rule; violations result in undefined behavior. |

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

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, two different translation units define a class of the same name with differing definitions. Although the two definitions are functionally equivalent (they both define a class named S with a single, public, nonstatic data member int a), they are not defined using the same sequence of tokens. This code example violates the ODR and results in undefined behavior. |
| // a.cpp  **struct** S {  **int** a;  };    // b.cpp  **class** S {  **public**:  **int** a;  }; |

| **Compliant Code** |
| --- |
| The correct mitigation depends on the programmer's intent. If the programmer intends for the same class definition to be visible in both translation units because of common usage, the solution is to use a header file to introduce the object into both translation units, as shown in this compliant solution. |
| // S.h  **struct** S {  **int** a;  };  // a.cpp  #include "S.h"  // b.cpp  #include "S.h" |

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

| **Principles(s):**  3: Architect and Design for Security Policies  4: Keep It Simple  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL60 |  |
| Parasoft  C/C++test | 2023.1 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| Polyspace Bug Finder | R2024a | CERT C++: DCL60-CPP | Checks for inline constraints not respected (rule partially covered |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not attempt to modify string literals. |

**Reference:** [**https://wiki.sei.cmu.edu/confluence/display/c/STR30-C.+Do+not+attempt+to+modify+string+literals**](https://wiki.sei.cmu.edu/confluence/display/c/STR30-C.+Do+not+attempt+to+modify+string+literals)

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the char pointer str is initialized to the address of a string literal. Attempting to modify the string literal is undefined behavior. |
| **char** \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| As an array initializer, a string literal specifies the initial values of characters in an array as well as the size of the array. This code creates a copy of the string literal in the space allocated to the character array str. The string stored in str can be modified safely. |
| **char** str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):** 2. Heed compiler warning |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | **string-literal-modfication write-to-string-literal** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR30 | Fully implemented |
| Helix QAC | 2024.2 | **C0556, C0752, C0753, C0754**  **C++3063, C++3064, C++3605, C++3606, C++3607** |  |
| Polyspace Bug Finder | R2024a | CERT C: Rule STR30-C | Checks for writing to const qualified object (rule fully covered) |

#### 

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not attempt to create a std::string from a null pointer. According to the C++ Standard, [char.traits.require], Table 62 [[ISO/IEC 14882-2014](https://wiki.sei.cmu.edu/confluence/display/cplusplus/AA.+Bibliography#AA.Bibliography-ISO/IEC14882-2014)], passing a null pointer to this function is undefined behavior because it would result in dereferencing a null pointer. |

**Reference:** [**STR51-CPP. Do not attempt to create a std::string from a null pointer - SEI CERT C++ Coding Standard - Confluence (cmu.edu)**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR51-CPP.+Do+not+attempt+to+create+a+std%3A%3Astring+from+a+null+pointer)

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist (or some other error occurs). |
| #include <cstdlib>  #include <string>    **void** f() {    std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>    **void** f() {  **const** **char** \*tmpPtrVal = std::**getenv**("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");  **if** (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):** 2: Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | assert\_failure |  |
| Klocwork | 2024.2 | **NPD.CHECK.CALL.MIGHT NPD.CHECK.CALL.MUST NPD.CHECK.MIGHT NPD.CHECK.MUST NPD.CONST.CALL NPD.CONST.DEREF NPD.FUNC.CALL.MIGHT NPD.FUNC.CALL.MUST NPD.FUNC.MIGHT NPD.FUNC.MUST NPD.GEN.CALL.MIGHT NPD.GEN.CALL.MUST NPD.GEN.MIGHT NPD.GEN.MUST RNPD.CALL RNPD.DEREF** |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2024a | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Avoid information leakage when passing a class object across a trust boundary; padding bits contain indeterminate values that may contain sensitive information. |

**Reference:** [**DCL55-CPP. Avoid information leakage when passing a class object across a trust boundary - SEI CERT C++ Coding Standard - Confluence (cmu.edu)**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL55-CPP.+Avoid+information+leakage+when+passing+a+class+object+across+a+trust+boundary)

| **Noncompliant Code** |
| --- |
| This noncompliant code example runs in kernel space and copies data from arg to user space. However, padding bits may be used within the object, for example, to ensure the proper alignment of class data members. These padding bits may contain sensitive information that may then be leaked when the data is copied to user space, regardless of how the data is copied. |
| #include <cstddef>    **struct** test {  **int** a;  **char** b;  **int** c;  };  // Safely copy bytes to user space  **extern** **int** copy\_to\_user(**void** \*dest, **void** \*src, std::**size\_t** size);    **void** do\_stuff(**void** \*usr\_buf) {    test arg{1, 2, 3};    copy\_to\_user(usr\_buf, &arg, **sizeof**(arg));  } |

| **Compliant Code** |
| --- |
| This compliant solution serializes the structure data before copying it to an untrusted context. |
| #include <cstddef>  #include <cstring>    **struct** test {  **int** a;  **char** b;  **int** c;  };  // Safely copy bytes to user space.  **extern** **int** copy\_to\_user(**void** \*dest, **void** \*src, std::**size\_t** size);    **void** do\_stuff(**void** \*usr\_buf) {    test arg{1, 2, 3};    // May be larger than strictly needed.    unsigned **char** buf[**sizeof**(arg)];    std::**size\_t** offset = 0;      std::**memcpy**(buf + offset, &arg.a, **sizeof**(arg.a));    offset += **sizeof**(arg.a);    std::**memcpy**(buf + offset, &arg.b, **sizeof**(arg.b));    offset += **sizeof**(arg.b);    std::**memcpy**(buf + offset, &arg.c, **sizeof**(arg.c));    offset += **sizeof**(arg.c);      copy\_to\_user(usr\_buf, buf, offset /\* size of info copied \*/);  } |

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

| **Principles(s):**  1: Validate Input Data  7: Sanitize Data Sent to Other Systems  8.  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | **function-argument-with-padding** | Partially checked |
| Cppcheck Premium | 24.9.0 | premium-cert-dcl39-c | Partially implemented |
| Parasoft C/C++test | 2023.1 | CERT\_C-DCL39-a | A pointer to a structure should not be passed to a function that can copy data to the user space |
| Polyspace Bug Finder | R2024a | CERT C: Rule DCL39-C | Partially checked |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory; Evaluating a pointer—including dereferencing the pointer, using it as an operand of an arithmetic operation, type casting it, and using it as the right-hand side of an assignment—into memory that has been deallocated by a memory management function is undefined behavior. Pointers to memory that have been deallocated are called *dangling pointers*. Accessing a dangling pointer can result in exploitable vulnerabilities. |

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

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

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

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

| **Principles(s):**  2: Heed Compiler Warnings  5: Default Deny  6: Adhere to the Principle of Least Privilege  9: Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | dangling\_pointer\_use | Supported  Astrée reports all accesses to freed allocated memory. |
| Cppcheck Premium | 24.9.0 | **doubleFree deallocret deallocuse** | Partially implemented |
| Coverity | 2017.07 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Parasoft C/C++test | 2023.1 | CERT\_C-MEM30-a | Do not use resources that have been freed |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use a static assertion to test the value of a constant expression; Assertions are a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities. |

**Reference:** [**DCL03-C. Use a static assertion to test the value of a constant expression - SEI CERT C Coding Standard - Confluence (cmu.edu)**](https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression)

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. |
| #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 |
| **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 |
| This portable compliant solution uses static\_assert. |
| #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):**  2: Heed Compiler Warnings  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus  Suite | 7.2.0 | CertC-DCL03 |  |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not abruptly terminate the program; Abnormal process termination is the typical vector for denial-of-service attacks. |

**References:**

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the call to f(), which was registered as an exit handler with std::at\_exit(), may result in a call to std::terminate() because throwing\_func() may throw an exception. |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

| **Compliant Code** |
| --- |
| In this compliant solution, f() handles all exceptions thrown by throwing\_func() and does not rethrow. |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.  **try** {      throwing\_func();    } **catch** (...) {      // Handle error    }  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

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

| **Principles(s):**  9: Use Effective Quality Assurance Techniques  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| Klocwork | 2024.2 | MISRA.CATCH.ALL CERT.ERR.ABRUPT\_TERM |  |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| Polyspace Bug  Finder | R2024a | CERT C++: ERR50-CP | Checks for implicit call to terminate() function (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-008-CPP] | Handle all exceptions thrown before main() begins executing;  if an uncaught exception is thrown after main() has finished executing, there are no further opportunities to handle the exception and it results in implementation-defined behavior. |

**Reference:** [**ERR58-CPP. Handle all exceptions thrown before main() begins executing - SEI CERT C++ Coding Standard - Confluence (cmu.edu)**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR58-CPP.+Handle+all+exceptions+thrown+before+main%28%29+begins+executing)

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the constructor for S may throw an exception that is not caught when globalS is constructed during program startup. |
| **struct** S {    S() noexcept(**false**);  };    **static** S globalS; |

| **Compliant Code** |
| --- |
| This compliant solution makes globalS into a local variable with static storage duration, allowing any exceptions thrown during object construction to be caught because the constructor for S will be executed the first time the function globalS() is called rather than at program startup. This solution does require the programmer to modify source code so that previous uses of globalS are replaced by a function call to globalS(). |
| **struct** S {    S() noexcept(**false**);  };    S &globalS() {  **try** {  **static** S s;  **return** s;    } **catch** (...) {      // Handle error, perhaps by logging it and gracefully terminating the application.    }    // Unreachable.  } |

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

| **Principles(s):**  2. Heed Compiler Warnings  4. Keep It Simple  8.  9. Use Effective Quality Assurance Techniques  10. 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 | 22.10 | potentially-throwing-static-initialization | Partially checked |
| CodeSonar | 8.1p0 | LANG.STRUCT.EXCP.THROW | Use of throw |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-ERR58-a | Exceptions shall be raised only after start-up and before termination of the program |
| Polyspace Bug Finder | R2024a | CERT C++: ERR58-CPP | Checks for exceptions raised during program startup (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Declarations** | [STD-009-CPP] | Do not modify the standard namespaces; Namespaces introduce new declarative regions for declarations, reducing the likelihood of conflicting identifiers with other declarative regions. |

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

| **Compliant Code** |
| --- |
| In this noncompliant code example, a template specialization of std::plus is added to the namespace std in an attempt to allow std::plus to concatenate a std::string and MyString object. However, because the template specialization is of a standard library–provided type (std::string), this code results in undefined behavior. |
| **namespace** nonstd {  **int** x;  } |

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

| **Principles(s):**  3: Architect and Design for Security Policies  4: Keep It Simple  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL58 |  |
| CodeSonar | 8.1p0 | LANG.STRUCT.DECL.SNM | Modification of Standard Namespaces |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-DCL58-a | Do not modify the standard namespaces 'std' and 'posix' |
| Polyspace Bug Finder | R2024a | CERT C++: DCL58-CPP | Checks for modification of standard namespaces (rule fully covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Containers** | [STD-010-CPP] | Use valid iterator ranges; Using a range of two iterators that are invalidated or do not refer into the same container results in undefined behavior. |

**References:** [**CTR53-CPP. Use valid iterator ranges - SEI CERT C++ Coding Standard - Confluence (cmu.edu)**](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR53-CPP.+Use+valid+iterator+ranges)

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the two iterators delimit the range point into the same container, but the first iterator does not precede the second. On each iteration of its internal loop, std::for\_each() compares the first iterator (after incrementing it) with the second for equality; as long as they are not equal, it will continue to increment the first iterator. Incrementing the iterator representing the past-the-end element of the range results in undefined behavior. |
| #include <algorithm>  #include <iostream>  #include <vector>    **void** f(**const** std::vector<**int**> &c) {    std::for\_each(c.end(), c.begin(), [](**int** i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the iterator values passed to std::for\_each() are passed in the proper order. |
| #include <algorithm>  #include <iostream>  #include <vector>    **void** f(**const** std::vector<**int**> &c) {    std::for\_each(c.begin(), c.end(), [](**int** i) { std::cout << i; });  } |

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

| **Principles(s):**  3: Architect and Design for Security Policies  4: Keep It Simple  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | overflow\_upon\_dereference |  |
| Helix QAC | 2024.2 | C++3802 |  |
| Parasoft  C/C++test | 2023.1 | CERT\_CPP-CTR53-a  CERT\_CPP-CTR53-b | Do not use an iterator range that isn't really a range Do not compare iterators from different containers |
| Polyspace Bug Finder | R2024a | CERT C++: CTR53-CPP | Checks for invalid iterator range (rule partially 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.

Integrating security measures into each step of the DevOps toolchain transforms DevOps into DevSecOps. In the "Assess and Plan" phase, we conduct threat modeling. In the "Design" and "Build" phases, we address IDE security. We add static application testing and automated security scans to the "Verify & Test" phase, along with unit, integration, and other tests. Once in production, we continue automated testing with prevention using integrity checks and defense-in-depth measures. Continuous threat detection methods include network monitoring, penetration testing, and performance logs. Just as we test in the QA sense, we should also perform security testing early and often.

### 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-003-CPP | High | Likely | Medium | High(P18) | 1 |
| STD-005-CPP | High | Likely | Medium | High(P18) | 1 |
| STD-002-CPP | Low | Likely | Low | Medium(P9) | 2 |
| STD-008-CPP | Low | Likely | Low | Medium(P9) | 2 |
| STD-009-CPP | High | Unlikely | Medium | Medium(P6) | 2 |
| STD-010-CPP | High | Probable | High | Medium(P6) | 2 |
| STD-007-CPP | Low | Probable | Medium | Low(P4) | 3 |
| STD-001-CPP | High | Unlikely | Medium | Low(P3) | 3 |
| STD-004-CPP | Low | Unlikely | High | Low(P1) | 3 |
| STD-006-CPP | Low | Unlikely | High | Low(P1) | 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 at rest | Encryption at rest protects stored data, including hard drives, phones, computers, and cloud assets. Data protection is achieved through encryption tools, disk encryption, and security for mobile devices and computers. |
| Encryption in flight | Encryption in flight is the process of securing data while it is moving. This can occur between two devices within a network or while the data is transferring outside of a network. Examples of protecting data in transit include email encryption, DLP (Data Loss Prevention) solutions, and robust network security features like firewalls and authentication. It is crucial to also consider the specific path that the data may take and ensure the security of this pathway. |
| Encryption in use | Encryption is used to protect data that is being created, edited, or otherwise in use. To ensure data security, it is important to have control and protection measures in place before the data is accessed. Managing access rights and user identity can also help minimize the risk to this data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the definitive process of confirming an individual's identity. This encompasses various forms, including static passwords, one-time passwords, certifications, and biometric credentials. These forms of identification unequivocally verify that a person is who they claim to be. |
| Authorization | Authorization specifies the access rights and privileges of a user and is an important part of information and computer security. While authentication confirms an identity, authorization determines what a user can and cannot access, limiting potential vulnerabilities when someone interacts with sensitive data they don't need to access or has permissions during access. |
| Accounting | Accounting involves keeping track of activities when interacting with a system. It includes showing timestamps, accessed resources, and data transfer information. This is important for creating a trail of user activity and for forensic analysis and investigation, if needed. |

**\***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 | 09/22/2024 | Milestone 3 | Jillian Ortiz |  |
| 1.2 | 10/13/2024 | Project One | Jillian Ortiz |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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