**Green Pace Developer: Security Policy Matthew R. Leclerc**



# 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 | Always validate that the input data is correct and in the expected format before any processing. Failure to do so opens up security breach points through harmful code or unexpected data injection, such as SQL injection or cross-site scripting. |
| 1. Heed Compiler Warnings | Pay attention to all warnings during development: compilers flag potential issues in code, be it uninitialized variables, deprecated methods, or type mismatches that may lead to security vulnerabilities if not handled. |
| 1. Architect and Design for Security Policies | Security needs to be at the heart of a system from its very inception. A well-defined security architecture should implement security policies consistently, which means every component of the system must meet required standards and minimize known threats. |
| 1. Keep It Simple | Simplify system designs and implementations to avoid many security-related weaknesses: 'Complexity is the enemy of security because complex systems increase the likelihood of flaws; these flaws can become vulnerabilities that may be leveraged by attackers. A simpler design will also be easier to secure and maintain. |
| 1. Default Deny | Deny access to all resources by default, granting access only explicitly given through policy. This means that unauthorized access is blocked, and any unintentional gaps in the access control policies are minimized. |
| 1. Adhere to the Principle of Least Privilege | Grant users, applications, and processes only the permissions they need to perform their tasks. This reduces the risk of malicious activity or accidental damage caused by having unnecessary privileges. |
| 1. Sanitize Data Sent to Other Systems | Always make sure to sanitize outgoing data to any other system so as not to send out malicious payloads to enable injection attacks, among other manipulations, to occur within distributed systems. |
| 1. Practice Defense in Depth | Implant several layers of security mechanisms to protect systems against attacks. If one layer fails, others can still provide protection. This approach increases the robustness of the security strategy as a whole. |
| 1. Use Effective Quality Assurance Techniques | Regularly perform quality assurance processes on the system, including static code analysis, dynamic testing, and penetration testing. The identification and fixation of vulnerabilities in the development process reduce the likelihood of their being exploited in production. |
| 1. Adopt a Secure Coding Standard | Adhere to secure coding guidelines and standards that include rules of thumb that are supposed to help in avoiding common security mistakes. Such guidelines like CERT or OWASP go further with best practices on how one can write secure code. |

### 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** |
| --- | --- | --- |
| **SQL Injection** | [STD-001-C] | Avoid side effects in arguments to unsafe macros |

| **Noncompliant Code** |
| --- |
| One problem with unsafe macros is side effects on macro arguments, as shown by this noncompliant code example. |
| |  | | --- | | #define ABS(x) (((x) < 0) ? -(x) : (x))    **void** func(**int** n) {    /\* Validate that n is within the desired range \*/  **int** m = ABS(++n);      /\* ... \*/  } |   The invocation of the ABS() macro in this example expands to   |  | | --- | | m = (((++n) < 0) ? -(++n) : (++n)); | |

| **Compliant Code** |
| --- |
| In this compliant solution, the increment operation ++n is performed before the call to the unsafe macro. |
| #define ABS(x) (((x) < 0) ? -(x) : (x)) /\* UNSAFE \*/    **void** func(**int** n) {    /\* Validate that n is within the desired range \*/    ++n;  **int** m = ABS(n);      /\* ... \*/  } |

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

| **Principles(s):** PRE31-C-EX1: An exception can be made for invoking an unsafe macro with a function call argument provided that the function has no side effects. However, it is easy to forget about obscure side effects that a function might have, especially library functions for which source code is not available; even changing errno is a side effect. Unless the function is user-written and does nothing but perform a computation and return its result without calling any other functions, it is likely that many developers will forget about some side effect. Consequently, this exception must be used with great care. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | **expanded-side-effect-multipliedexpanded-side-effect-not-evaluatedside-effect-not-expanded** | Partially Checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-PRE31** | Fully Implemented |
| CodeSonar | 8.3p0 | **http://lang.preproc.funcmacrolang.struct.se.declang.struct.se.inc/** | Function-Like Macro Side Effects in Expression with Decrement Side Effects in Expression with Increment |
| Coverity | 2017.07 | **ASSERT\_SIDE\_EFFECTS** | Partially implemented Can detect the specific instance where assertion contains an operation/function call that may have a side effect |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-002-CPP] | Do not define a C-style variadic function |

| **Noncompliant Code** |
| --- |
| A variadic function using a function parameter pack is used to implement the add() function, allowing identical behavior for call sites. |
| #include <cstdarg>    **int** add(**int** first, **int** second, ...) {  **int** r = first + second;  **va\_list** va;  **va\_start**(va, second);  **while** (**int** v = **va\_arg**(va, **int**)) {      r += v;    }  **va\_end**(va);  **return** r;  } |

| **Compliant Code** |
| --- |
| A variadic function using a function parameter pack is used to implement the add() function, allowing identical behavior for call sites. |
| #include <type\_traits>    **template** <**typename** Arg, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Arg s) { **return** f + s; }    **template** <**typename** Arg, **typename**... Ts, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Ts... rest) {  **return** f + add(rest...);  } |

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

| **Principles(s):** **DCL50-CPP-EX1**: It is permissible to define a C-style variadic function if that function also has external C language linkage. For instance, the function may be a definition used in a C library API that is implemented in C++. **DCL50-CPP-EX2**: As stated in the normative text, C-style variadic functions that are declared but never defined are permitted. For example, when a function call expression appears in an unevaluated context, such as the argument in a sizeof expression, overload resolution is performed to determine the result type of the call but does not require a function definition. Some template metaprogramming techniques that employ SFINAE use variadic function declarations to implement compile-time type queries, as in the following example. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **function-ellipsis** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL50** |  |
| Clang | 3.9 | cert-dcl50-cpp | Checked by clang-tidy. |
| CodeSonar | 8.3p0 | **LANG.STRUCT.ELLIPSIS** | Ellipsis |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-003-CPP] | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| An uninitialized local variable is evaluated as part of an expression to print its value, resulting in undefined behavior. |
| #include <iostream>    **void** f() {  **int** i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| The object is initialized prior to printing its value. |
| #include <iostream>    **void** f() {  **int** i = 0;    std::cout << i;  } |

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

| **Principles(s):** **EXP53-CPP:** Reading uninitialized variables is undefined behavior and can result in unexpected program behavior. In some cases, these security flaws may allow the execution of arbitrary code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | uninitialized-read | Partially checked |
| Clang | 3.9 | -Wuninitializedclang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory. |
| CodeSonar | 8.3p0 | LANG.STRUCT.RPLLANG.MEM.UVAR | Return pointer to local Uninitialized variable |
| Helix QAC | 2024.4 | DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978 |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-004-C] | Ensure that operations on signed integers do not result in overflow |

| **Noncompliant Code** |
| --- |
| This noncompliant code example can result in a signed integer overflow during the addition of the signed operands si\_a and si\_b. |
| **void** func(**signed** **int** si\_a, **signed** **int** si\_b) {  **signed** **int** sum = si\_a + si\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Compliant solution ensures that the addition operation cannot overflow, regardless of representation |
| #include <limits.h>    **void** f(**signed** **int** si\_a, **signed** **int** si\_b) {  **signed** **int** sum;  **if** (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||        ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {      /\* Handle error \*/    } **else** {      sum = si\_a + si\_b;    }    /\* ... \*/  } |

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

| **Principles(s):** **INT32-C**: Integer overflow can lead to buffer overflows and the execution of arbitrary code by an attacker. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | Integer-overflow | Fully Checked |
| CodeSonar | 8.3p0 | ALLOC.SIZE.ADDOFLOW,ALLOC.SIZE.IOFLOW,ALLOC.SIZE.MULOFLOW,ALLOC.SIZE.SUBUFLOW,MISC.MEM.SIZE.ADDOFLOW,MISC.MEM.SIZE.BAD,MISC.MEM.SIZE.MULOFLOW,MISC.MEM.SIZE.SUBUFLOW | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| Coverity | 2017.07 | TAINTED\_SCALAR, BAD\_SHIFT | Implemented |
| CppCheck Premium | 24.11.0 | premium-cert-int32-c |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-005-C] | Prevent or detect domain and range errors in math functions |

| **Noncompliant Code** |
| --- |
| Example determines the square root of x. |
| #include <math.h>    **void** func(**double** x) {  **double** result;    result = **sqrt**(x);  } |

| **Compliant Code** |
| --- |
| This function has domain errors but no range errors, bounds checking can be used to prevent domain errors: |
| #include <math.h>    **void** func(**double** x) {  **double** result;    **if** (isless(x, 0.0)) {      /\* Handle domain error \*/    }      result = **sqrt**(x);  } |

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

| **Principles(s):** **FLP32-C**: Failure to prevent or detect domain and range errors in math functions may cause unexpected results. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | **P8** | **L2** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | Stdlib-limits | Partially Checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-FLP32** | Partially Implemented |
| CodeSonar | 8.3p0 | **MATH.DOMAIN.ATAN MATH.DOMAIN.TOOHIGH MATH.DOMAIN.TOOLOW MATH.DOMAIN MATH.RANGE MATH.RANGE.GAMMA MATH.DOMAIN.LOG MATH.RANGE.LOG MATH.DOMAIN.FE\_INVALID MATH.DOMAIN.POW MATH.RANGE.COSH.TOOHIGH MATH.RANGE.COSH.TOOLOW MATH.DOMAIN.SQRT** | Arctangent Domain Error Argument Too High Argument Too Low Floating Point Domain Error Floating Point Range Error Gamma on Zero Logarithm on Negative Value Logarithm on Zero Raises FE\_INVALID Undefined Power of Zero cosh on High Number cosh on Low Number sqrt on Negative Value |
| HelixQAC | 2024.4 | C5025, C++ 5033 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-006-C] | Do not subtract or compare two pointers that do not refer to the same array |

| **Noncompliant Code** |
| --- |
| Pointer subtraction is used to determine how many free elements are left in the nums array. |
| #include <stddef.h>    **enum** { SIZE = 32 };    **void** func(**void**) {  **int** nums[SIZE];  **int** end;  **int** \*next\_num\_ptr = nums;  **size\_t** free\_elements;      /\* Increment next\_num\_ptr as array fills \*/      free\_elements = &end - next\_num\_ptr;  } |

| **Compliant Code** |
| --- |
| The number of free elements is computed by subtracting next\_num\_ptr from the address of the pointer past the nums array. While this pointer may not be dereferenced, it may be used in pointer arithmetic. |
| #include <stddef.h>  **enum** { SIZE = 32 };    **void** func(**void**) {  **int** nums[SIZE];  **int** \*next\_num\_ptr = nums;  **size\_t** free\_elements;      /\* Increment next\_num\_ptr as array fills \*/      free\_elements = &(nums[SIZE]) - next\_num\_ptr;  } |

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

| **Principles(s):** **ARR36-C-EX1:**Comparing two pointers to distinct members of the same struct object is allowed. Pointers to structure members declared later in the structure compare greater-than pointers to members declared earlier in the structure. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | **P8** | **L2** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | Pointer-subtraction | Partially Checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ARR36 | Can detect operations on pointers that are unrelated |
| CodeSonar | 8.3p0 | **LANG.STRUCT.CUP**  **LANG.STRUCT.SUP** | Comparison of Unrelated Pointers  Subtraction of Unrelated Pointers |
| Coverity | 2017.07 | **MISRA C 2004 17.2**  **MISRA C 2004 17.3**  **MISRA C 2012 18.2**  **MISRA C 2012 18.3** | Implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-007-C] | Do not attempt to modify string literals |

| **Noncompliant Code** |
| --- |
| 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. (See STR11-C. Do not specify the bound of a character array initialized with a string literal.) 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):** **STR30-C:** Modifying string literals can lead to abnormal program termination and possibly denial-of-service attacks. |
| --- |

**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 |
| Compass/ROSE |  |  | Can detect simple violations of this rule. |
| Coverity | 2017.07 | PW | Deprecates conversion from a string literal to "char \*" |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-008-C] | Allocate sufficient memory for an object |

| **Noncompliant Code** |
| --- |
| Inadequate space is allocated for a struct tm object because the size of the pointer is being used to determine the size of the pointed-to object |
| #include <stdlib.h>  #include <time.h>    **struct** **tm** \*make\_tm(**int** year, **int** mon, **int** day, **int** hour,  **int** min, **int** sec) {  **struct** **tm** \*tmb;    tmb = (**struct** **tm** \*)**malloc**(**sizeof**(tmb));  **if** (tmb == NULL) {  **return** NULL;    }    \*tmb = (**struct** **tm**) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };  **return** tmb;  } |

| **Compliant Code** |
| --- |
| The correct amount of memory is allocated for the struct tm object. When allocating space for a single object, passing the (dereferenced) pointer type to the sizeof operator is a simple way to allocate sufficient memory. Because the sizeof operator does not evaluate its operand, dereferencing an uninitialized or null pointer in this context is well-defined behavior. |
| #include <stdlib.h>  #include <time.h>    **struct** **tm** \*make\_tm(**int** year, **int** mon, **int** day, **int** hour,  **int** min, **int** sec) {  **struct** **tm** \*tmb;    tmb = (**struct** **tm** \*)**malloc**(**sizeof**(\*tmb));  **if** (tmb == NULL) {  **return** NULL;    }    \*tmb = (**struct** **tm**) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };  **return** tmb;  } |

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

| **Principles(s):** **MEM35-C**: Providing invalid size arguments to memory allocation functions can lead to buffer overflows and the execution of arbitrary code with the permissions of the vulnerable process. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | Malloc-size-insufficient | Partially checked  Besides direct rule violations, all undefined behaviour resulting from invalid memory accesses is reported by Astrée. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM35 |  |
| CodeSonar | 8.3p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW ALLOC.SIZE.TRUNC IO.TAINT.SIZE MISC.MEM.SIZE.BAD LANG.MEM.BO LANG.MEM.BU LANG.STRUCT.PARITH LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.MEM.TBA LANG.MEM.TO LANG.MEM.TU** | Addition overflow of allocation size Addition overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Truncation of allocation size Tainted allocation size Unreasonable size argument Buffer Overrun Buffer Underrun Pointer Arithmetic Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun |
| Coverity | [Insert text.] |  | Could check violations of this rule by examining the size expression to malloc() or memcpy() functions. Specifically, the size argument should be bounded by 0, SIZE\_MAX, and, unless it is a variable of type size\_t or rsize\_t, it should be bounds-checked before the malloc() call. If the argument is of the expression a\*b, then an appropriate check is   |  | | --- | | **if** (a < SIZE\_MAX / b && a > 0) ... | |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-C] | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| The file opened by the call to fopen() is not closed before function func() returns. |
| #include <stdio.h>    **int** func(**const** **char** \*filename) {  **FILE** \*f = **fopen**(filename, "r");  **if** (NULL == f) {  **return** -1;    }    /\* ... \*/  **return** 0;  } |

| **Compliant Code** |
| --- |
| The file pointed to by f is closed before returning to the caller. |
| #include <stdio.h>    **int** func(**const** **char** \*filename) {  **FILE** \*f = **fopen**(filename, "r");  **if** (NULL == f) {  **return** -1;    }    /\* ... \*/  **if** (**fclose**(f) == EOF) {  **return** -1;    }  **return** 0;  } |

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

| **Principles(s):** **FIO42-C**: Failing to properly close files may allow an attacker to exhaust system resources and can increase the risk that data written into in-memory file buffers will not be flushed in the event of abnormal program termination. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Klocwork | 2024.4 | RH.LEAK |  |
| LDRA tool suite | 9.7.1 | 49 D | Partially Implemented |
| Parasoft C/C++ test | 2024.2 | CERT\_C\_FIO42-a | Ensures resources are freed |
| PC-lint plus | 1.4 | 429 | Partially Supported |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-C] | Do not modify the object referenced by the return value of certain functions. |

| **Noncompliant Code** |
| --- |
| Modifies the string returned by getenv() by replacing all double quotation marks (") with underscores (\_): |
| #include <stdlib.h>    **void** trstr(**char** \*c\_str, **char** orig, **char** rep) {  **while** (\*c\_str != '\0') {  **if** (\*c\_str == orig) {        \*c\_str = rep;      }      ++c\_str;    }  }    **void** func(**void**) {  **char** \*env = **getenv**("TEST\_ENV");  **if** (env == NULL) {      /\* Handle error \*/    }    trstr(env,'"', '\_');  } |

| **Compliant Code** |
| --- |
| If the programmer does not intend to modify the environment, this compliant solution demonstrates how to modify a copy of the return value: |
| #include <stdlib.h>  #include <string.h>    **void** trstr(**char** \*c\_str, **char** orig, **char** rep) {  **while** (\*c\_str != '\0') {  **if** (\*c\_str == orig) {        \*c\_str = rep;      }      ++c\_str;    }  }    **void** func(**void**) {  **const** **char** \*env;  **char** \*copy\_of\_env;      env = **getenv**("TEST\_ENV");  **if** (env == NULL) {      /\* Handle error \*/    }      copy\_of\_env = (**char** \*)**malloc**(**strlen**(env) + 1);  **if** (copy\_of\_env == NULL) {      /\* Handle error \*/    }    **strcpy**(copy\_of\_env, env);    trstr(copy\_of\_env,'"', '\_');    /\* ... \*/  **free**(copy\_of\_env);  } |

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

| **Principles(s):** **ENV30-C:** Modifying the object pointed to by the return value of getenv(), setlocale(), localeconv(), asctime(), or strerror() is undefined behavior. Even if the modification succeeds, the modified object can be overwritten by a subsequent call to the same function. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | Stdlib-const-pointer-assign | Partially Checked |
| PVS-Studio | 7.35 | V675 |  |
| Polyspace Bug Finder | R2024a | CERT C: Rule ENV30-C | Checks for modification of internal buffer returned from nonreentrant standard function (rule fully covered) |
| Klocwork | 2024.4 | **MISRA.STDLIB.CTYPE.RANGE.2012\_AMD1** **MISRA.STDLIB.ILLEGAL\_REUSE.2012\_AMD1** **MISRA.STDLIB.ILLEGAL\_WRITE.2012\_AMD1** |  |

### 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.

[Insert your written explanations here.]

### 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-002-CPP | High | Probable | Medium | P12 | 5 |
| STD-003-CPP | High | Probable | Medium | P12 | 5 |
| STD-008-CPP | High | Probable | High | P6 | 5 |
| STD-004-CPP | High | Likely | High | P9 | 4 |
| STD-005-CPP | Medium | Probable | Medium | P8 | 3 |
| STD-006-CPP | Medium | Probable | Medium | P8 | 3 |
| STD-007-CPP | Low | Likely | Low | P9 | 2 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | 2 |
| STD-010-CPP | Low | Probable | Medium | P4 | 2 |
| STD-001-CPP | Low | Unlikely | Low | P3 | 1 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### 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 ensures that in-storage data cannot be read by unauthorized people even if attackers have compromised storage drives. Data is secured by secure cryptographic mechanisms such as AES-256. Encryption is used on databases, file storage, and backup to prevent unauthorized access to these. All sensitive information must have full-disk encryption, database encryption, and encrypted storage in accordance with this directive. |
| Encryption in flight | Encryption in transit (or data-in-transit encryption) protects data in transit over networks. Encryption of communications between databases, APIs, servers, and clients is part of that. TLS 1.2+ and HTTPS protocols need to be deployed to defend against interceptions (e.g., man-in-the-middle attacks). VPNs or secure communications need to be used for sharing of external and internal data. |
| Encryption in use | Encryption in operation protects actively used data in memory so that it is secure even while in use. Technologies like homomorphic encryption, secure computing, and secure enclaves (Intel SGX, AMD SEV) protect sensitive data in computations. This policy protects sensitive workloads in secure runtimes and leverages memory encryption where it is possible to do so. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication ensures that only legitimate users have access to systems and data. Multi-factor authentication (MFA), password policies that enforce good password strength and periodic password changes, and integration with SSO (Single Sign-On) and federated identity providers must all be used. Biometric authentication must be used for highly sensitive systems. |
| Authorization | Authorization defines how authenticated users have access to things based on their privilege level and their role. They must follow the Principle of Least Privilege (PoLP), so that only resources that the user requires have their level of privilege granted to them. Role-based access control or attribute-based access control must be used. Regular review of accesses through audits must take place to limit privilege creep. |
| Accounting | Accounting tracks user activity and system activity for audits and for compliance. Tracks user logins, file accesses, changes to databases, and administrative actions. Logs are secured with SIEM technology for analysis and for anomaly detection. Logs have to be retained in accordance with compliance legislation (e.g., HIPAA, GDPR, SOC 2). |

**\***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 | 01/26/2025 | Module 3 | Matt Leclerc |  |
| 1.2 | 02/16/2025 | Module 6 | Matt Leclerc |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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