**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 | Validate input from all untrusted data sources. Input validation can eliminate most software vulnerabilities. Be suspicious of external data sources, including command line arguments, environment variables and user-controlled files. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available to the compiler and eliminate warnings by modifying code. |
| 1. Architect and Design for Security Policies | Create a software architecture and design software to implement and enforce security policies. |
| 1. Keep It Simple | Keep the design as simple as possible. The more complex a design increases the likelihood that errors will be during implementation, configuration, and use. As mechanisms become more complex an effort to achieve an appropriate level of assurance increases dramatically. |
| 1. Default Deny | Base access decisions on permissions rather than exclusion. By default, access is denied and protection schemes identifies conditions under which access is allowed. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should only be access for the least amount of time required to complete the task. This will reduce opportunities attackers have to execute arbitrary code. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems, such as command shells, databases, and commercial off-the-shelf components. Attackers can invoke unused functionality in these components through use of SQL, command, or other injection attacks. |
| 1. Practice Defense in Depth | Manage risk with multiple layers of defense, so if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Internal and external security reviews can make systems more secure. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for the programming language and platform. |

### 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** | **Converting a pointer to integer or integer to pointer** |

| **Noncompliant Code** |
| --- |
| The size of a pointer can be greater than the size of an integer, such as in an implementation where pointers are 64 bits and unsigned integers are 32 bits. This code example is noncompliant on such implementations because the result of converting the 64-bit ptr cannot be represented in the 32-bit integer type: |
| **void** f(**void**) {  **char** \*ptr;    /\* ... \*/    unsigned **int** number = (unsigned **int**)ptr;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The C Standard guarantees that a pointer to void may be converted to or from a pointer to any object type and back again and that the result must compare equal to the original pointer. Consequently, converting directly from a char \* pointer to a uintptr\_t, as in this compliant solution, is allowed on implementations that support the uintptr\_t type. |
| #include <stdint.h>    **void** f(**void**) {  **char** \*ptr;    /\* ... \*/  **uintptr\_t** number = (**uintptr\_t**)ptr;    /\* ... \*/  } |

| **Principles(s):**  Heed Compiler Warnings – Pay attention to warnings in relation to the use of data types.  Architect and Design for Security Policies – By designing security policies will ensure that invalid data types are not used or attempting to convert invalid data types.  Use Effective Quality Assurance Techniques – Testing code will insure invalid conversions do not take place. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | CERT, MISRA, and CWE | Tool that covers all major standards and 30+ languages |
| Helix QAC |  | MISRA, CERT, CWE, CWE 25, ISO | C and C++ Security and Safety analysis tool |
| Parasoft | 2022.2 | MISRA, CWE 25, CERT, JSF, AUTOSAE | C and C++ testing tool that includes static code analysis |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | **STD-002-CPP** | **Range check element access** |

| **Noncompliant Code** |
| --- |
| In this code example, the value returned by the call to get\_index() may be greater than the number of elements stored in the string, result in undefined behavior |
| #include <string>    **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| This solution uses a range check to check that the value returned by the get\_index() function is within a valid range before calling operator[]() |
| #include <string>    **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");    std::**size\_t** i = get\_index();  **if** (i < s.length()) {      s[i] = '1';    } **else** {      // Handle error    }  } |

| **Principles(s):**  ValidateInput Data – Validating user input will ensure it is not outside the bounds.  Adopt a Secure Coding Standard – Adopting secure code can prevent the interception of data values as they are transmitted.  Use Effective Quality Assurance Techniques – Testing data values to make sure no values can be inserted out of bounds. |
| --- |

**Threat Level**

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

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the valid range |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++3162, C++3163, C++3164, C++3165** | [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.MEM.BO** **LANG.MEM.BU** **LANG.MEM.TBA** **LANG.MEM.TO** **LANG.MEM.TU** | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | **STD-003-CPP** | **Do not attempt to modify string literals** |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the char pointer p 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. |
| **char** str[] = "string literal";  str[0] = 'S'; |

| **Principles(s):**  Heed Compiler Warnings – Pay attention to warnings in relation to the use of data types.  Architect and Design for Security Policies – By designing security policies will ensure that invalid data types are not used or attempting to convert invalid data types.  Adopt a Secure Coding Standard – Adopting secure code and following the standards can prevent the modification of string literals. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **PW** | Deprecates conversion from a string literal to "char \*" |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-STR30-a** **CERT\_C-STR30-b** | A string literal shall not be modified Do not modify string literals |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | **STD-004-CPP** | **Exclude user input from format** |

| **Noncompliant Code** |
| --- |
| The incorrect\_password() function in this noncompliant code example is called during identification and authentication to display an error message if the specified user is not found or the password is incorrect. The function accepts the name of the user as a string referenced by user. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    **void** incorrect\_password(**const** **char** \*user) {  **int** ret;    /\* User names are restricted to 256 or fewer characters \*/  **static** **const** **char** msg\_format[] = "%s cannot be authenticated.\n";  **size\_t** len = **strlen**(user) + **sizeof**(msg\_format);  **char** \*msg = (**char** \*)**malloc**(len);  **if** (msg == NULL) {      /\* Handle error \*/    }    ret = snprintf(msg, len, msg\_format, user);  **if** (ret < 0) {      /\* Handle error \*/    } **else** **if** (ret >= len) {      /\* Handle truncated output \*/    }  **fprintf**(stderr, msg);  **free**(msg);  } |

| **Compliant Code** |
| --- |
| Solution fixes the problem by replacing the fprintf() call with a call to fputs(), which outputs msg directly to stderr without evaluating its contents: |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    **void** incorrect\_password(**const** **char** \*user) {  **int** ret;    /\* User names are restricted to 256 or fewer characters \*/  **static** **const** **char** msg\_format[] = "%s cannot be authenticated.\n";  **size\_t** len = **strlen**(user) + **sizeof**(msg\_format);  **char** \*msg = (**char** \*)**malloc**(len);  **if** (msg == NULL) {      /\* Handle error \*/    }    ret = snprintf(msg, len, msg\_format, user);  **if** (ret < 0) {      /\* Handle error \*/    } **else** **if** (ret >= len) {      /\* Handle truncated output \*/    }  **fputs**(msg, stderr);  **free**(msg);  } |

| **Principles(s):**  ValidateInput Data – Validating user input will ensure proper inputs are accepted  Adopt a Secure Coding Standard – Adopting secure code can prevent the interception of data values as they are transmitted.  Use Effective Quality Assurance Techniques – Testing data values to make sure proper input validation takes place.  Architect and Design for Security Policies – By designing security policies will ensure no vulnerabilities through SQL injection can take place. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2022.2 | **CERT\_C-FIO30-a** **CERT\_C-FIO30-b** **CERT\_C-FIO30-c** | Avoid calling functions printf/wprintf with only one argument other than string constant Avoid using functions fprintf/fwprintf with only two parameters, when second parameter is a variable Never use unfiltered data from an untrusted user as the format parameter |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.23 | [V618](https://pvs-studio.com/en/docs/warnings/v618/) | [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **DF4916, DF4917, DF4918** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2023.1 | **SV.FMTSTR.GENERIC** **SV.TAINTED.FMTSTR** |  |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| This example shows both the incorrect and correct techniques for freeing the memory associated with a linked list. In their (intentionally) incorrect example, p is freed before p->next is executed, so that p->next reads memory that has already been freed. |
| #include <stdlib.h>    **struct** node {  **int** value;  **struct** node \*next;  };    **void** free\_list(**struct** node \*head) {  **for** (**struct** node \*p = head; p != NULL; p = p->next) {  **free**(p);    }  } |

| **Compliant Code** |
| --- |
| This solution correct this error by storing a reference to p->next  in q before freeing p: |
| #include <stdlib.h>    **struct** node {  **int** value;  **struct** node \*next;  };    **void** free\_list(**struct** node \*head) {  **struct** node \*q;  **for** (**struct** node \*p = head; p != NULL; p = q) {      q = p->next;  **free**(p);    }  } |

| **Principles(s):**  Validating Input Data – Ensure input data is not attempting to access memory  Architect and Design for Security Policies – Designing and adhering to security policies will ensure no vulnerabilities are introduced.  Keep it simple – Keep code simple will keep code lightweight and make is eas  Use Effective Quality Assurance Techniques – Testing software can ensure there are no unauthorized access to memory addresses. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **DF4866, DF4867, DF4868, DF4871, DF4872, DF4873**  **C++3339, C++4303, C++4304** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-MEM30-a** | Do not use resources that have been freed |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.23 | **[V586](https://pvs-studio.com/en/docs/warnings/v586/)**, [V774](https://pvs-studio.com/en/docs/warnings/v774/) |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **ALLOC.UAF** | Use after free |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| This 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 |
| **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 |

| **Principles(s):**  Architect and Design for Security Policies – Designing and adhering to security policies will ensure no vulnerabilities are introduced.  Use Effective Quality Assurance Techniques – Testing software can ensure there are no errors in code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| DCL03-C | Low | Unlikely | High | **P1** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | **STD-007-CPP** | **Handle all exceptions thrown before main() begins executing** |

| **Noncompliant Code** |
| --- |
| This noncompliant code overflows its buffer if msg is too long, and it has [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior) if msg is a null pointer: |
| **void** complain(**const** **char** \*msg) {  **static** **const** **char** prefix[] = "Error: ";  **static** **const** **char** suffix[] = "\n";  **char** buf[BUFSIZ];    **strcpy**(buf, prefix);  **strcat**(buf, msg);  **strcat**(buf, suffix);  **fputs**(buf, stderr);  } |

| **Compliant Code** |
| --- |
| This compliant solution will not overflow its buffer: |
| **void** complain(**const** **char** \*msg) {    errno\_t err;  **static** **const** **char** prefix[] = "Error: ";  **static** **const** **char** suffix[] = "\n";  **char** buf[BUFSIZ];    err = strcpy\_s(buf, **sizeof**(buf), prefix);  **if** (err != 0) {    }    err = strcat\_s(buf, **sizeof**(buf), msg);  **if** (err != 0) {    }    err = strcat\_s(buf, **sizeof**(buf), suffix);  **if** (err != 0) {    }  **fputs**(buf, stderr);  } |

| **Principles(s):**  Architect and Design for Security Policies – Designing and adhering to security policies will ensure no vulnerabilities are introduced.  Use Effective Quality Assurance Techniques – Testing software can ensure there are no errors in code.  Adopt a Secure Coding Standard – Adopting secure code can prevent exceptions in code |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.STRUCT.EXCP.THROW** | Use of throw |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++4634, C++4636, C++4637, C++4639** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-ERR58-a** | Exceptions shall be raised only after start-up and before termination of the program |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | **STD-008-CPP** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |

| **Noncompliant Code** |
| --- |
| To solve this problem, it may be tempting to use the std::ios\_base::width() method, but there still is a trap, as shown in this noncompliant code example. |
| #include <iostream>    **void** f() {  **char** bufOne[12];  **char** bufTwo[12];    std::cin.width(12);    std::cin >> bufOne;    std::cin >> bufTwo;  } |

| **Compliant Code** |
| --- |
| The best solution for ensuring that data is not truncated and for guarding against buffer overflows is to use std::string instead of a bounded array, as in this compliant solution. |
| #include <iostream>  #include <string>    **void** f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |

| **Principles(s):**  Heed Compiler Warnings – Pay attention to warnings in relation to the use of data types.  Architect and Design for Security Policies – By designing security policies and adhering to security policies will ensure data is correctly allocated.  Use Effective Quality Assurance Techniques – Testing code will ensure no vulnerabilities can be introduced. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO** **LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++5216**  **DF2835, DF2836, DF2839,** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **NNTS.MIGHT** **NNTS.TAINTED** **NNTS.MUST** **SV.UNBOUND\_STRING\_INPUT.CIN** |  |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [S3519](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3519) |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | **STD-009-CPP** | Detect and handle standard library errors |

| **Noncompliant Code** |
| --- |
| The function may fail for other reasons as well, such as the lack of resources.  Depending on the sequence of characters pointed to by utf8, the subsequent call to mbstowcs() may fail or result in the function storing an unexpected sequence of wide characters in the supplied buffer wcs. |
| #include <locale.h>  #include <stdlib.h>    **int** utf8\_to\_wcs(**wchar\_t** \*wcs, **size\_t** n, **const** **char** \*utf8,  **size\_t** \*size) {  **if** (NULL == size) {  **return** -1;    }  **setlocale**(LC\_CTYPE, "en\_US.UTF-8");    \*size = **mbstowcs**(wcs, utf8, n);  **return** 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution checks the value returned by setlocale() and avoids calling mbstowcs() if the function fails. The function also takes care to restore the locale to its initial setting before returning control to the caller. |
| #include <locale.h>  #include <stdlib.h>    **int** utf8\_to\_wcs(**wchar\_t** \*wcs, **size\_t** n, **const** **char** \*utf8,  **size\_t** \*size) {  **if** (NULL == size) {  **return** -1;    }  **const** **char** \*save = **setlocale**(LC\_CTYPE, "en\_US.UTF-8");  **if** (NULL == save) {  **return** -1;    }      \*size = **mbstowcs**(wcs, utf8, n);  **if** (NULL == **setlocale**(LC\_CTYPE, save)) {  **return** -1;    }  **return** 0;  } |

| **Principles(s):**  Heed Compiler Warnings – Pay attention to warnings in relation to the exceptions.  Use Effective Quality Assurance Techniques – Testing code to make sure no exceptions are returned. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2023.1 | **NPD.CHECK.MUST** **NPD.FUNC.MUST** **SV.RVT.RETVAL\_NOTTESTED** | [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-ERR33-a** **CERT\_C-ERR33-b** **CERT\_C-ERR33-c** **CERT\_C-ERR33-d** | The value returned by a function having non-void return type shall be used The value returned by a function having non-void return type shall be used Avoid null pointer dereferencing Always check the returned value of non-void function |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **MISRA C 2012 Rule 22.8**  **MISRA C 2012 Rule 22.9**  **MISRA C 2012 Rule 22.10** | Implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | **STD-010-CPP** | **Use only explicitly signed or unsigned char type for numeric values** |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the char-type variable c may be signed or unsigned. Assuming 8-bit, two's complement character types, this code may print out either i/c = 5 (unsigned) or i/c = -17 (signed). It is much more difficult to reason about the correctness of a program without knowing if these integers are signed or unsigned. |
| **char** c = 200;  **int** i = 1000;  **printf**("i/c = %d\n", i/c); |

| **Compliant Code** |
| --- |
| In this compliant solution, the variable c is declared as unsigned char. The subsequent division operation is now independent of the signedness of char and consequently has a predictable result. |
| unsigned **char** c = 200;  **int** i = 1000;  **printf**("i/c = %d\n", i/c); |

| **Principles(s):**  Heed Compiler Warnings – Pay attention to warnings in relation to the use of data types.  Architect and Design for Security Policies – By designing security policies and adhering to security policies will ensure data is correctly allocated.  Use Effective Quality Assurance Techniques – Testing code will ensure no vulnerabilities can be introduced. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **C1292, C1293, C4401, C4421, C4431, C4441, C4451** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2023.1 | [PORTING.SIGNED.CHAR](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-INT07-a** **CERT\_C-INT07-b** | The plain char type shall be used only for the storage and use of character values signed and unsigned char type shall be used only for the storage and use of numeric values |

### 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



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.

Green Pace will modify their current DevOps process in the “Verify and test” and “Design” stages. The implementation of new static analysis tools to scan code for vulnerabilities, test code, detect dead code, and errors in code according to standards and guidelines. The code will need to be modified to ensure the code follows current standards and code. Each time code is modified, static code analysis will need to be performed to determine if the newly developed code introduces any additional vulnerabilities or error in code. Additionally, penetration testing can be carried out during transition and health check stages to make sure the code does not allow unauthorized access to it via the internet.

### 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 | Low | Probable | High | P2 | L3 |
| STD-002-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-003-CPP | Low | Likely | Low | P9 | L2 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-006-CPP | Low | Unlikely | High | **P1** | Low |
| STD-007-CPP | Low | Likely | Low | **P9** | **L2** |
| STD-008-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-009-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-0010-CPP | Medium | Probable | Medium | **P8** | **L2** |

### 

### Create Policies for Encryption and Triple A

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest is data that is encrypted while it is being stored and only a given number of users can access it. By encrypting data at rest, they are ensuring customer data or code does not get leaked or compromised. |
| Encryption at flight | Encryption in flight is data being transmitted, it is paramount that anytime data is transmitted to customers Green Pace encrypt that data. Code developed for customers may contain sensitive information and encrypting the transmission of data and the data being transmitted is a must. |
| Encryption in use | Encrypting data in use means encrypting data and data requests in real-time and then block suspicious requests. Green Pace will need to make sure the applications they develop, and use encrypt data while it is in use, especially if the data is sensitive, would-be hackers could be access the memory used by the software to obtain security keys and other important information. Encrypting data in use would prevent this. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is part of “User identity and access management solutions” in DiD where a user is required to enter a private username and password to verify their identity and then as an additional layer of security multi-factor authentication, single sign-on, or lifecycle management can be used to increase security. |
| Authorization | Authorization is the method of determining whether a user should have access to sensitive data. It also determines if a user can read, write, delete, or execute files or folders. It is most often paired with authentication and is part of access control which is under physical security in the DiD. |
| Accounting | Accounting is the process of keeping track of what a user is doing with the access they have. It logs sessions statistics and usage information and is used for authorization control, trends, and resource utilization. It could be used in threat assessment by analyzing potential risky habits or practices by users. |

## 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 | 3/27/2023 | Initial Template | Eric Wallace |  |
| 1.1 | 4/2/2023 | Finished the remainin items in template | Eric Wallace |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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