**Green Pace Developer: Security Policy Guide**



# Green Pace Secure Development Policy

# Revised By: Marcus Spruill

# August 11, 2024

## 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 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 12](#_Toc52464063)

[Coding Standard 6 14](#_Toc52464064)

[Coding Standard 7 16](#_Toc52464065)

[Coding Standard 8 18](#_Toc52464066)

[Coding Standard 9 20](#_Toc52464067)

[Coding Standard 10 22](#_Toc52464068)

[Defense-in-Depth Illustration 24](#_Toc52464069)

[Project One 24](#_Toc52464070)

[1. Revise the C/C++ Standards 24](#_Toc52464071)

[2. Risk Assessment 24](#_Toc52464072)

[3. Automated Detection 24](#_Toc52464073)

[4. Automation 24](#_Toc52464074)

[5. Summary of Risk Assessments 25](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 25](#_Toc52464076)

[7. Map the Principles 26](#_Toc52464077)

[Audit Controls and Management 27](#_Toc52464078)

[Enforcement 27](#_Toc52464079)

[Exceptions Process 27](#_Toc52464080)

[Distribution 28](#_Toc52464081)

[Policy Change Control 28](#_Toc52464082)

[Policy Version History 28](#_Toc52464083)

[Appendix A Lookups 28](#_Toc52464084)

[Approved C/C++ Language Acronyms 28](#_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 | Input data must be formatted properly to be correctly processed and stored. For example, this includes removing characters such as non-digits for operands for math operations. Non-validated data is a vector for undefined behavior. |
| 1. Heed Compiler Warnings | Compiler warnings alert developers to issues in code before compilation and encourage them to address them. Common compiler warnings include deprecated functions within libraries, declaring unmodified literals as constants, and implicit conversion warnings between data types (short to int). |
| 1. Architect and Design for Security Policies | Design documentation for architecture and security defines how software components, or separate systems, exchange and process data. Permission requirements are described, and it notes events if the requirements are not met. |
| 1. Keep It Simple | Superfluous complexity makes code more difficult to test and understand. Code should be straightforward to accomplish its task without extra memory or processing resources. |
| 1. Default Deny | Actions should be denied by default, with exceptions granted for permissions. This restricts levels of permissions to authorized users and reduces access vectors for malicious actors. |
| 1. Adhere to the Principle of Least Privilege | Granting only required permissions to users to perform their tasks. Greater permissions than required for users do not impact their task but introduces the risk that sensitive data is accessed or overwritten. |
| 1. Sanitize Data Sent to Other Systems | Data variables and objects should have unnecessary internal data removed. This helps reduces risks of bugs and vectors of attack to external systems. It can also improve efficiency by sanitizing data before storing it in secured systems, rather than validating stored data on each read operation from a secured system. |
| 1. Practice Defense in Depth | Defense in Depth is a security strategy to secure an attack vector by multiple, distinct layers of security that validates against an attack method. |
| 1. Use Effective Quality Assurance Techniques | The testing phase of the SDLC advises developers to test their code often to discover bugs and vulnerabilities early, which greatly reduces the time and resources to address them compared to later phases. The testing performed should have a defined procedure and its results should be recorded to be compared against if needed. |
| 1. Adopt a Secure Coding Standard | Coding standards and conventions are guidelines that ensure that developers are practicing secure development. It also aims to reduce vulnerabilities by providing secure solutions to common problems, or by tracking unpatched vulnerabilities in distributed libraries so that IDEs or external code analysis programs can inform developers of. |

### 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** | **Do not declare or define a reserved identifier** |
| --- | --- | --- |
| **Data Type** | DCL-051-CPP | Certain identifiers are reserved for use as part of the standard library functions. User defined functions, variables and enumerations should not use reserved names to avoid ambiguity. |

| **Noncompliant Code** |
| --- |
| MAX\_SIZE is user-defined but it conflicts with the name of the header macro in <cstdint> to denote the upper limit of std:size\_t. |
| #include <cinttypes> // for int\_fast16\_t    void f(std::int\_fast16\_t val) {    enum { MAX\_SIZE = 80 };    // ...  } |

| **Compliant Code** |
| --- |
| The enumerated value uses the non-conflicting name BufferSize. |
| #include <cinttypes> // for std::int\_fast16\_t    void f(std::int\_fast16\_t val) {    enum { BufferSize = 80 };    // ...  } |

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

| **Principles(s):** Heed Compiler Warnings – Compilers will warn when user-defined identifiers names are shared with included libraries. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.1 | **MISRA.DEFINE.WRONGNAME** **MISRA.DEFINE.WRONGNAME.UNDERSCORE** **MISRA.UNDEF.WRONGNAME** **MISRA.UNDEF.WRONGNAME.UNDERSCORE** **MISRA.STDLIB.WRONGNAME** **MISRA.STDLIB.WRONGNAME.UNDERSCORE** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **86 S, 218 S, 219 S, 580 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-DCL51-a** **CERT\_CPP-DCL51-b** **CERT\_CPP-DCL51-c** **CERT\_CPP-DCL51-d** **CERT\_CPP-DCL51-e** **CERT\_CPP-DCL51-f** | Do not #define or #undef identifiers with names which start with underscore Do not redefine reserved words Do not #define nor #undef identifier 'defined' The names of standard library macros, objects and functions shall not be reused The names of standard library macros, objects and functions shall not be reused (C90) The names of standard library macros, objects and functions shall not be reused (C99) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: DCL51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl51cpp.html) | Checks for redefinitions of reserved identifiers (rule partially covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that division and remainder operations do not result in divide-by-zero errors** |
| --- | --- | --- |
| **Data Value** | INT-033-C | When the divisor of a division or remainder operation is 0, it will result in a divide-by-zero error. Additionally, an overflow can occur if the dividend is the minimum for a signed integer type and the divisor is -1. |

| **Noncompliant Code** |
| --- |
| The if statement checks only for an overflow error if the dividend is the minimum value of the data type, and the divisor is equal to -1. It does not check if the divisor is equal to 0 to prevent a divide-by-zero error. |
| #include <limits.h>    void func(signed long s\_a, signed long s\_b) {    signed long result;    if ((s\_a == LONG\_MIN) && (s\_b == -1)) {      /\* Handle error \*/    } else {      result = s\_a / s\_b;    }    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The if statement checks if the divisor is equal to zero, and for signed overflow. |
| #include <limits.h>    void func(signed long s\_a, signed long s\_b) {    signed long result;    if ((s\_b == 0) || ((s\_a == LONG\_MIN) && (s\_b == -1))) {      /\* Handle error \*/    } else {      result = s\_a / s\_b;    }    /\* ... \*/  } |

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

| **Principles(s):** Validate Input Data – The denominator should be validated to prevent a divide-by-zero exception. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | int-division-by-zero  int-modulo-by-zero | Fully checked |
| CodeSonar | 8.1p0 | LANG.ARITH.DIVZERO  LANG.ARITH>FDIVZERO | Division by zero  Float Division By Zero |
| Coverity | 2017.07 | DIVIDE\_BY\_ZERO | Fully implemented |
| Cppcheck | 1.66 | zerodiv  zerodivcond | Context sensitive analysis of division by zero Not detected for division by struct member / array element / pointer data that is 0 Detected when there is unsafe division by variable before/after test if variable is zero |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do not attempt to create a std::string from a null pointer** |
| --- | --- | --- |
| **String Correctness** | STR-051-CPP | String is not a pointer-type, and thus it cannot be assigned null. |

| **Noncompliant Code** |
| --- |
| The return value of std::getenv() is immediately stored in a string variable without a validation check against null. Strings are not allowed to be null because it is not a pointer type. |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| The return value of getenv() is stored in a c string. If the c string is null, the string is assigned an empty value. Otherwise, the string is assigned the value stored in the c string. |
| #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):** Heed Compiler Warnings – The compiler will warn when an incompatible assignment is attempted. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **assert\_failure** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.MEM.NPD** | Null Pointer Dereference |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **DF4770, DF4771, DF4772, DF4773, DF4774** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-STR51-a** | Avoid null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Exclude user input from format strings** |
| --- | --- | --- |
| **SQL Injection** | FIO-030-C | Attackers can use formatted strings that contain a tainted value, a value from a source that has not been sanitized, to crash processes, view memory contents, or even write to arbitrary memory locations. |

| **Noncompliant Code** |
| --- |
| The example below calls the incorrect\_password() function to display an error if the user-input user name is incorrect.  The contents of msg originates from untrusted user input which is passed as the format string to fprintf() to output it to stderr. |
| #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** |
| --- |
| The fprintf() call is substituted by a fputs() call, which outputs the contents of msg directly to stderr. |
| #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);  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems - The format string is untrusted user input that can be used to write an arbitrary memory address. Such user input should be as a variadic argument to fprint(), or output the contents directly via fputs(). |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-FIO30** | Partially implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **IO.INJ.FMT MISC.FMT** | Format string injection Format string |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **TAINTED\_STRING** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Properly deallocate dynamically allocated resources** |
| --- | --- | --- |
| **Memory Protection** | MEM-051-CPP | Undefined behavior occurs if a deallocation attempt is made on a resource that have not been dynamically allocated. |

| **Noncompliant Code** |
| --- |
| Variables i1 and i2 are uninitialized and are only assigned a value in a try block. If the catch block attempts to deallocate i2 before the try block could assign a value, the resulting behavior would be undefined. |
| #include <new>    void f() {    int \*i1, \*i2;    try {      i1 = new int;      i2 = new int;    } catch (std::bad\_alloc &) {      delete i1;      delete i2;    }  } |

| **Compliant Code** |
| --- |
| The i1 and i2 variables are initialized with a nullptr, thus allowing the deallocating delete statements to have definite behavior. |
| #include <new>    void f() {    int \*i1 = nullptr, \*i2 = nullptr;    try {      i1 = new int;      i2 = new int;    } catch (std::bad\_alloc &) {      delete i1;      delete i2;    }  } |

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

| **Principles(s):** Heed Compiler Warnings – The compiler will produce a warning for declared variables that are used in operations before initialization. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **invalid\_dynamic\_memory\_allocation dangling\_pointer\_use** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MEM51** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | **clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete clang-analyzer-unix.MismatchedDeallocator** | Checked by clang-tidy, but does not catch all violations of this rule |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **ALLOC.FNH ALLOC.DF ALLOC.TM ALLOC.LEAK** | Free non-heap variable Double free Type mismatch Leak |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Incorporate diagnostic tests using assertions** |
| --- | --- | --- |
| **Assertions** | MSC-011-C | Asserting that a variable is a certain value or within a certain range, or that an object or construct exists. This can be useful for narrowing the cause of errors in the absence of error checking during development. |

| **Noncompliant Code** |
| --- |
| The code below uses assert for memory allocation, a resource that may not be available during runtime depending on system resources. |
| char \*dupstring(const char \*c\_str) {    size\_t len;    char \*dup;      len = strlen(c\_str);    dup = (char \*)malloc(len + 1);    assert(NULL != dup);      memcpy(dup, c\_str, len + 1);    return dup;  } |

| **Compliant Code** |
| --- |
| The code below uses an alternative solution to detect potential memory exhaustion that will exit if it was not allocated. |
| char \*dupstring(const char \*c\_str) {    size\_t len;    char \*dup;      len = strlen(c\_str);    dup = (char\*)malloc(len + 1);    /\* Detect and handle memory allocation error \*/    if (NULL == dup) {        return NULL;    }      memcpy(dup, c\_str, len + 1);    return dup;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – Using assert() will produce output denoting success or failure of the inputted expression. This may help isolate errors to more precise areas. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.FUNCS.ASSERTS** | Not enough assertions |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **ASSERT\_SIDE\_EFFECT** | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-MSC11-a** | Assert liberally to document internal assumptions and invariants |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Do not abruptly terminate the program** |
| --- | --- | --- |
| **Exceptions** | ERR-050-CPP | Programs should exit gracefully to allow open streams to flush data properly, temporary files to be deleted, and other external processes to shut down. |

| **Noncompliant Code** |
| --- |
| Function f() is registered as an exit handler and may additionally result in a call to std::terminate() because an exception may be thrown. |
| #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** |
| --- |
| Function f() handles exceptions thrown by throwing\_func(). |
| #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):** Adopt a Secure Coding Standard – Proper cleanup of resources ensure that lingering fragments in data streams cannot be dumped or modified, and reduce the opportunities that temporary files are accessed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **BADFUNC.ABORT BADFUNC.EXIT** | Use of abort Use of exit |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++5014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **MISRA.TERMINATE** **CERT.ERR.ABRUPT\_TERM** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **122 S** | Enhanced Enforcement |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not cast to an out-of-range enumeration value** |
| --- | --- | --- |
| **Data Value** | INT-050-CPP | Enumeration values have a range as defined by its enumeration type in the C++ Standard. Unspecified behavior will occur if an out-of-bounds value is cast to an enumerated variable. |

| **Noncompliant Code** |
| --- |
| The example casts the integer intVar to an enumeration enumVar before bounds checking. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| The example casts the integer intVar to an enumeration enumVar after bounds checking. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** Validate Input Data – Input data should be verified to prevent exceptions, such as an out-of-bounds when accessing an out of range index. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **cast-integer-to-enum** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-INT50** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.CAST.COERCE**  **LANG.CAST.VALUE** | Coercion Alters Value  Cast Alters Value |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++3013** |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Range check element access** |
| --- | --- | --- |
| **String Correctness** | STR-053-CPP | Strings contain chars that can be accessed by their index within the string. If an attempt to access an out-of-range index is made, then undefined behavior will occur. |

| **Noncompliant Code** |
| --- |
| The value returned by get\_index() may be larger than the number of elements in a string. |
| #include <string>    extern std::size\_t get\_index();    void f() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| std::basic\_string::at() is used in place of the index operator[] but will through an std::out\_of\_range exception if the position returned is greater than the size of the string. |
| #include <stdexcept>  #include <string>  extern std::size\_t get\_index();    void f() {    std::string s("01234567");    try {      s.at(get\_index()) = '1';    } catch (std::out\_of\_range &) {      // Handle error    }  } |

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

| **Principles(s):** **Principles(s):** Heed Compiler Warnings – The compiler should produce a warning that there is no exception handler for a direct index access attempt. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **assert\_failure** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **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 |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++3162, C++3163, C++3164, C++3165** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the valid range |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do not let exceptions escape from destructors or deallocation functions** |
| --- | --- | --- |
| **Data Type** | DCL-057-CPP | Destruction and deallocation functions can be interrupted by thrown exceptions. Unexpected termination of those functions can result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| The noexcept assertion can not be met because the class destructor itself may throw an exception, thus undefined behavior can occur. |
| #include <stdexcept>    bool perform\_dealloc(void \*);    void operator delete(void \*ptr) noexcept(false) {  if (perform\_dealloc(ptr)) {  throw std::logic\_error("Something bad");  }  } |

| **Compliant Code** |
| --- |
| The destructor invokes operations that are not able to throw exceptions. |
| #include <cstdlib>  #include <stdexcept>    bool perform\_dealloc(void \*);  void log\_failure(const char \*);    void operator delete(void \*ptr) noexcept(true) {  if (perform\_dealloc(ptr)) {  log\_failure("Deallocation of pointer failed");  std::exit(1); // Fail, but still call destructors  }  } |

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

| **Principles(s):** Adopt a Secure Coding Standard – Functions that are responsible for allocation or construction and deallocation or destruction should fulfill those features upon exit from the function. Following coding standards ensure that the appropriate actions are executed or handled gracefully (i.e. validating that file.close() is true once a file function that uses file.open() is called). |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | **P6** | **L2** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **destructor-without-noexcept delete-without-noexcept** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL57** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.STRUCT.EXCP.CATCH**  **LANG.STRUCT.EXCP.THROW** | Use of catch  Use of throw |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++2045, C++2047, C++4032, C++4631** |  |

### 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.

DevSecOps is a model that integrates security into the development and testing process of the software development lifecycle (SDLC). The standard SDLC approach begins with the planning and design phases, migrates to the development and testing phases, and finishes with the deployment and maintenance phases.

Incorporating the use of security automated tools during the testing phase allows issues to be detected early, when compared to a final security audit after development has concluded. Using an automation tool requires significantly less time than a manual review and it provides an opportunity to implement solutions as the problems arise, reducing the risk that defects will accumulate and compound into more sophisticated problems.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| DCL51-CPP | Low | Unlikely | Low | **P3** | **L3** |
| INT33-C | Low | Likely | Medium | **P6** | **L2** |
| STR51-CPP | High | Likely | Medium | **P18** | **L1** |
| FIO30-C | High | Likely | Medium | **P18** | **L1** |
| MEM51-CPP | High | Likely | Medium | **P18** | **L1** |
| MSC11-C | Low | Unlikely | High | **P1** | **L3** |
| ERR50-CPP | Low | Probable | Medium | **P4** | **L3** |
| INT50-CPP | Medium | Unlikely | Medium | **P4** | **L3** |
| STR53-CPP | High | Unlikely | Medium | **P6** | **L2** |
| DCL57-CPP | Low | Likely | Medium | **P6** | **L2** |

### 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 | Data stored on disk should be encrypted. This ensures that if the data is accessed without authorization the breacher will not be able to access the information within without the correct encryption keys. User generated files containing account credentials or information are commonly encrypted so attackers can not retrieve them and use them on another system as each encryption requires a specific key. |
| Encryption in flight | Data should be encrypted when transferred between systems. This ensures that if the information is intercepted or redirected then the information within cannot be accessed with the correct encryption keys. Communication protocols should employ encryption to protect transferred data, especially for services with sensitive data such as banking services, healthcare and government records, and for personal data such as user email or cloud storage services. |
| Encryption in use | Data should be encrypted when in use by applications and programs. This ensures that memory scanning will not reveal the contents of sensitive information. Common tactics include scanning memory after a read operation of user-generated files. User data that is temporarily stored within a system’s memory is often encrypted so memory dumps will not reveal user information such as full name, address, and so on. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication verifies that an accessing user is correct through a unique combination of account credentials. In addition to restricting access to known individuals, this policy also provides the basis in which accounting can record detailed information. Authentication is necessary to provide a secure system that provides detailed logging. |
| Authorization | Authorization provides permissions to user accounts. This policy further restricts the data that a user can create, access, modify and delete in databases. It forms another layer of defense by allowing access only to those that need it, helping prevent unnecessary damage. |
| Accounting | Accounting records information about user actions. This policy helps with the establishment of dangerous user patterns to rectify before it progresses to more severe actions, and it also provides a final log for analysis for unauthorized access to systems or files. |

**\***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 | 08/11/2024 | Completed Template | Marcus Spruill |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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