**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 | Input data should be assumed to be malicious until it is proven otherwise. No matter where the input data comes from, whether a service, a file share, a user, or a database, the input data should be validated. Proper input validation can eliminate a vast majority of security vulnerabilities. |
| 1. Heed Compiler Warnings | Code should be compiled to use the highest warning level available for the compiler and to eliminate warnings. Static and dynamic analysis can be performed to find and eliminate security flaws. |
| 1. Architect and Design for Security Policies | Software architecture should be created, and software should be designed to implement and enforce security policies. Distinct intercommunicating systems can be used with an appropriate privilege set if the system requires different privileges at different times. |
| 1. Keep It Simple | Designs should be kept as simple as possible as complex designs increase the likelihood that errors will be made during implementation, as well as during configuration and use. There is more effort involved in assuring complex designs are secure and this effort will increase as the design becomes more complex. |
| 1. Default Deny | Access should be denied on default, and access should be based on permission rather than exclusions. The protections should identify the conditions on which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | The least set of privileges necessary to do the job should be executed for every process. Elevated permission should only be allowed for the limited amount of time required to do a task. Using this approach will reduce the opportunity of an attacker to execute code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Data sent to complex subsystems should be sanitized. This includes data such as command shells, relational databases, and commercial off-the-shelf products. Attackers may be able to invoke SQL commands or other injection attacks into these components as the application understands the context upon which the external application is receiving data. |
| 1. Practice Defense in Depth | Risks should be managed with multiple defensive strategies so if one one layer of defense is not successful, another layer can prevent a security flaw. Combining secure programming techniques with secure runtime environments should reduce the potential for vulnerabilities remaining in the code at runtime to be exploited during operational development. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques are often effective at identifying and eliminating security vulnerabilities. Penetration testing, fuzz auditing, and source code audits should be included as a part of an effective quality assurance plan. In addition, independent security reviews can also help in creating more secure systems. |
| 1. Adopt a Secure Coding Standard | A secure coding standard should be developed that is specifically designed for the programming language that is chosen for the application and platform. |

### C/C++ Ten Coding Standards

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Never qualify a reference type with const or volatile  C++ does not allow you to change the value of a reference type, effectively treating all references as being const qualified. The C++ Standard, [dcl.ref], paragraph 1 [ISO/IEC 14882-2014], states the following:  Cv-qualified references are ill-formed except when the cv-qualifiers are introduced through the use of a typedef-name (7.1.3, 14.1) or decltype-specifier (7.1.6.2), in which case the cv-qualifiers are ignored.  Thus, C++ prohibits or ignores the cv-qualification of a reference type. Only a value of non-reference type may be cv-qualified. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a const-qualified reference to a char is formed instead of a reference to a const-qualified char. This results in undefined behavior. |
| #include <iostream>    void f(char c) {  char &const p = c;  p = 'p';  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| This compliant solution removes the const qualifier. |
| #include <iostream>    void f(char c) {  char &p = c;  p = 'p';  std::cout << c << std::endl;  } |

| **Principles(s): 1, 4, 2, 10** Designs should be kept as simple as possible and follow the programming language’s guidelines. Input data should be validated. Compiler errors should be adhered to and the code modified to comply to ensure that security vulnerabilities are not introduced. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL52** | |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | | **C++0014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | | **CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | | [CERT C++: DCL52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl52cpp.html) | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | |  | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options. |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | | [**S3708**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3708) |  |

#### 

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not cast to an out-of-range enumeration value  Enumerations in C++ come in two forms: scoped enumerations in which the underlying type is fixed and unscoped enumerations in which the underlying type may or may not be fixed. The range of values that can be represented by either form of enumeration may include enumerator values not specified by the enumeration itself. The range of valid enumeration values for an enumeration type is defined by the C++ Standard, [dcl.enum], in paragraph 8 [ISO/IEC 14882-2020]:  For an enumeration whose underlying type is fixed, the values of the enumeration are the values of the underlying type. Otherwise, the values of the enumeration are the values representable by a hypothetical integer type with minimal width M such that all enumerators can be represented. The width of the smallest bit-field large enough to hold all the values of the enumeration type is M. It is possible to define an enumeration that has values not defined by any of its enumerators. If the enumerator-list is empty, the values of the enumeration are as if the enumeration had a single enumerator with value 0.  The C++ Standard, [expr.static.cast], paragraph 10, states the following:  A value of integral or enumeration type can be explicitly converted to a complete enumeration type. If the enumeration type has a fixed underlying type, the value is first converted to that type by integral conversion, if necessary, and then to the enumeration type. If the enumeration type does not have a fixed underlying type, the value is unchanged if the original value is within the range of the enumeration values (9.7.1), and otherwise, the behavior is undefined. A value of floating-point type can also be explicitly converted to an enumeration type. The resulting value is the same as converting the original value to the underlying type of the enumeration (7.3.10), and subsequently to the enumeration type.  To avoid operating on unspecified values, the arithmetic value being cast must be within the range of values the enumeration can represent. When dynamically checking for out-of-range values, checking must be performed before the cast expression. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. On a two's complement system, the valid range of values that can be represented by EnumType are [0..3], so if a value outside of that range were passed to f(), the cast to EnumType would result in an unspecified value, and using that value within the if statement results in unspecified behavior. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);    if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| This compliant solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does this by restricting the converted value to one for which there is a specific enumerator value. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  if (intVar < First || intVar > Third) {  // Handle error  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

| **Principles(s):** 1, 2, 3, 4, 10 – designs should be kept as simple as possible and should adhere to the best practices of the programming language. Input data should be validated. The architecture should be created to ensure that secure standards are followed. |
| --- |

**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) | 7.3p0 | **LANG.CAST.COERCE**  **LANG.CAST.VALUE** | Coercion Alters Value  Cast Alters Value |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++3013** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-INT50-a** | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.25 | [**V1016**](https://pvs-studio.com/en/docs/warnings/v1016/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **cast-integer-to-enum** | Partially checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CLG] | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| Copying data to a buffer that is not large enough to hold that data results in a buffer overflow. Buffer overflows occur frequently when manipulating strings [Seacord 2013b]. To prevent such errors, either limit copies through truncation or, preferably, ensure that the destination is of sufficient size to hold the character data to be copied and the null-termination character. (See STR03-C. Do not inadvertently truncate a string.) This noncompliant code example demonstrates an off-by-one error [Dowd 2006]. The loop copies data from src to dest. However, because the loop does not account for the null-termination character, it may be incorrectly written 1 byte past the end of dest: |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {  size\_t i;    for (i = 0; src[i] && (i < n); ++i) {  dest[i] = src[i];  }  dest[i] = '\0';  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the loop termination condition is modified to account for the null-termination character that is appended to dest: |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {  size\_t i;    for (i = 0; src[i] && (i < n - 1); ++i) {  dest[i] = src[i];  }  dest[i] = '\0';  } |

| **Principles(s):** 1, 2, 3, 4, 10 – designs should be kept as simple as possible and should adhere to the best practices of the programming language. Input data should be validated. The architecture should be created to ensure that secure standards are followed in order to ensure that security vulnerabilities are not introduced. |
| --- |

**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-STR31** | Detects calls to unsafe string function that may cause buffer overflow Detects potential buffer overruns, including those caused by unsafe usage of fscanf() |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.MEM.BO LANG.MEM.TO MISC.MEM.NTERM BADFUNC.BO.\*** | Buffer overrun Type overrun No space for null terminator A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| [Compass/ROSE](https://www.securecoding.cert.org/confluence/display/seccode/Rose) |  |  | Can detect violations of the rule. However, it is unable to handle cases involving strcpy\_s() or manual string copies such as the one in the first example |
| [Coverity](https://www.securecoding.cert.org/confluence/display/seccode/Coverity) | 2017.07 | **STRING\_OVERFLOW**  **BUFFER\_SIZE**  **OVERRUN**  **STRING\_SIZE** | Fully implemented |
| [Fortify SCA](https://www.securecoding.cert.org/confluence/display/seccode/Fortify) | 5.0 |  |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **C2840,  C5009, C5038**  **C++0145, C++5009, C++5038**  **DF2840, DF2841, DF2842, DF2843, DF2845, DF2846, DF2847, DF2848, DF2930, DF2931, DF2932, DF2933, DF2935, DF2936, DF2937, DF2938** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2023.1 | **SV.FMT\_STR.BAD\_SCAN\_FORMAT** **SV.UNBOUND\_STRING\_INPUT.FUNC** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152309) | 9.7.1 | **489 S, 109 D, 66 X, 70 X, 71 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-STR31-a** **CERT\_C-STR31-b** **CERT\_C-STR31-c** **CERT\_C-STR31-d** **CERT\_C-STR31-e** | Avoid accessing arrays out of bounds Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Avoid using unsafe string functions which may cause buffer overflows |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **421, 498** | Partially supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rule STR31-C](https://www.mathworks.com/help/bugfinder/ref/certcrulestr31c.html) | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation * Insufficient destination buffer size   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.25 | [**V518**](https://pvs-studio.com/en/docs/warnings/v518/), [**V645**](https://pvs-studio.com/en/docs/warnings/v645/), [**V727**](https://pvs-studio.com/en/docs/warnings/v727/), [**V755**](https://pvs-studio.com/en/docs/warnings/v755/) |  |
| [Splint](https://www.securecoding.cert.org/confluence/display/seccode/Splint) | 3.1.1 |  |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **mem\_access** | Exhaustively verified (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/144ae03a)). |

#### 

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Range check element access  The std::string index operators const\_reference operator[](size\_type) const and reference operator[](size\_type) return the character stored at the specified position, pos. When pos >= size(), a reference to an object of type charT with value charT() is returned. The index operators are unchecked (no exceptions are thrown for range errors), and attempting to modify the resulting out-of-range object results in undefined behavior.  Similarly, the std::string::back() and std::string::front() functions are unchecked as they are defined to call through to the appropriate operator[]() without throwing.  Do not pass an out-of-range value as an argument to std::string::operator[](). Similarly, do not call std::string::back() |

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

| **Compliant Code** |
| --- |
| This compliant solution uses the std::basic\_string::at() function, which behaves in a similar fashion to the index operator[] but throws a std::out\_of\_range exception if pos >= size(). |
| #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  }  } |

| **Principles(s):** 2, 3, 4, 10 – designs should be kept as simple as possible and should adhere to the best practices of the programming language. The architecture should be created to ensure that secure standards are followed in order to ensure that security vulnerabilities are not introduced. |
| --- |

**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) | 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 |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++3162, C++3163, C++3164, C++3165** |  |
| [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 |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory  Evaluating a pointer—including dereferencing the pointer, using it as an operand of an arithmetic operation, type casting it, and using it as the right-hand side of an assignment—into memory that has been deallocated by a memory management function is undefined behavior. Pointers to memory that has been deallocated are called dangling pointers. Accessing a dangling pointer can result in exploitable vulnerabilities.  It is at the memory manager's discretion when to reallocate or recycle the freed memory. When memory is freed, all pointers into it become invalid, and its contents might either be returned to the operating system, making the freed space inaccessible, or remain intact and accessible. As a result, the data at the freed location can appear to be valid but change unexpectedly. Consequently, memory must not be written to or read from once it is freed. |

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

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

| **Principles(s):** 2, 3, 4, 10 – designs should be kept as simple as possible and should adhere to the best practices of the programming language. The architecture should be created to ensure that secure standards are followed in order to ensure that security vulnerabilities are not introduced. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **dangling\_pointer\_use** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MEM50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **ALLOC.UAF** | Use after free |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  |  |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | v7.5.0 | **USE\_AFTER\_FREE** | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++4303, C++4304** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2023.1 | **UFM.DEREF.MIGHT** **UFM.DEREF.MUST** **UFM.FFM.MIGHT** **UFM.FFM.MUST** **UFM.RETURN.MIGHT** **UFM.RETURN.MUST** **UFM.USE.MIGHT** **UFM.USE.MUST** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **483 S, 484 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: MEM50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem50cpp.html) | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.25 | [**V586**](https://pvs-studio.com/en/docs/warnings/v586/), [**V774**](https://pvs-studio.com/en/docs/warnings/v774/) |  |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Splint) | 5.0 |  |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | Use a static assertion to test the value of a constant expression  Assertions are a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities (see MSC11-C. Incorporate diagnostic tests using assertions). The runtime assert() macro has some limitations, however, in that it incurs a runtime overhead and because it calls abort(). Consequently, the runtime assert() macro is useful only for identifying incorrect assumptions and not for runtime error checking. As a result, runtime assertions are generally unsuitable for server programs or embedded systems. |

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly: |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution: |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

| **Principles(s):** 9 -Quality assurance techniques are often effective at identifying and eliminating security vulnerabilities. Assertions are valuable in finding software defects that may lead to security vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** |  |
|  |  |  |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Honor exception specifications  If a function throws an exception other than one allowed by its exception-specification, it can lead to an implementation-defined termination of the program.  If a function declared with a dynamic-exception-specification throws an exception of a type that would not match the exception-specification, the function std::unexpected() is called. The behavior of this function can be overridden but, by default, causes an exception of std::bad\_exception to be thrown. Unless std::bad\_exception is listed in the exception-specification, the function std::terminate() will be called.  Similarly, if a function declared with a noexcept-specification throws an exception of a type that would cause the noexcept-specification to evaluate to false, the function std::terminate() will be called.  Calling std::terminate() leads to implementation-defined termination of the program. To prevent abnormal termination of the program, any function that declares an exception-specification should restrict itself, as well as any functions it calls, to throwing only allowed exceptions. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a function is declared as nonthrowing, but it is possible for std::vector::resize() to throw an exception when the requested memory cannot be allocated. |
| #include <cstddef>  #include <vector>    void f(std::vector<int> &v, size\_t s) noexcept(true) {  v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the function's noexcept-specification is removed, signifying that the function allows all exceptions. |
| #include <cstddef>  #include <vector>    void f(std::vector<int> &v, size\_t s) {  v.resize(s); // May throw, but that is okay  } |

| **Principles(s):** 10,2 – Code should be compiled to heed compiler warnings and help to eliminate security flaws. Code should adhere to the best practices of the selected programming language. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **unhandled-throw-noexcept** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR55** |  |
| [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++4035, C++4036, C++4632** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **56 D** | Partially implemented |
| [Parasoft C/C++Test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-ERR55-a** | Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: ERR55-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr55cpp.html) | Checks for noexcept functions exiting with exception (rule fully covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **unhandled-throw-noexcept** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous | [STD-008-CPP] | Value-returning functions must return a value from all exit paths  The C++ Standard, [stmt.return], paragraph 2 [ISO/IEC 14882-2014], states the following:  Flowing off the end of a function is equivalent to a return with no value; this results in undefined behavior in a value-returning function.  A value-returning function must return a value from all code paths; otherwise, it will result in undefined behavior. This includes returning through less-common code paths, such as from a function-try-block, as explained in the C++ Standard, [except.handle], paragraph 15:  Flowing off the end of a function-try-block is equivalent to a return with no value; this results in undefined behavior in a value-returning function (6.6.3). |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the programmer forgot to return the input value for positive input, so not all code paths return a value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, all code paths now return a value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  return a;  } |

| **Principles(s):** 1, 10 – Input data should be validated in order to eliminate a vast majority of security vulnerabilities. Code should adhere to the best practices of the selected programming language. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **return-implicit** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MSC52** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | **-Wreturn-type** | Does not catch all instances of this rule, such as function-try-blocks |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.STRUCT.MRS** | Missing return statement |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **DF2888** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **FUNCRET.GEN**  **FUNCRET.IMPLICIT** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **2 D, 36 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-MSC52-a** | All exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: MSC52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc52cpp.html) | Checks for missing return statements (rule partially covered) |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S935**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-935) |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.25 | [**V591**](https://pvs-studio.com/en/docs/warnings/v591/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **return-implicit** | Fully checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions | [STD-009-CPP] | Do not throw an exception across execution boundaries  Throwing an exception requires collaboration between the execution of the throw expression and the passing of control to the appropriate catch statement, if one applies. This collaboration takes the form of runtime logic used to calculate the correct handler for the exception and is an implementation detail specific to the platform. For code compiled by a single C++ compiler, the details of how to throw and catch exceptions can be safely ignored. However, when throwing an exception across execution boundaries, care must be taken to ensure the runtime logic used is compatible between differing sides of the execution boundary.  An execution boundary is the delimitation between code compiled by differing compilers, including different versions of a compiler produced by the same vendor. For instance, a function may be declared in a header file but defined in a library that is loaded at runtime. The execution boundary is between the call site in the executable and the function implementation in the library. Such boundaries are also called ABI (application binary interface) boundaries because they relate to the interoperability of application binaries.  Throw an exception across an execution boundary only when both sides of the execution boundary use the same ABI for exception handling. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an exception is thrown from a library function to signal an error. Despite the exception being a scalar type (and thus implicitly conforming to EXP60-CPP. Do not pass a nonstandard-layout type object across execution boundaries), this code can still result in abnormal program execution if the library and application adhere to different ABIs. |
| // library.h  void func() noexcept(false); // Implemented by the library    // library.cpp  void func() noexcept(false) {  // ...  if (/\* ... \*/) {  throw 42;  }  }    // application.cpp  #include "library.h"    void f() {  try {  func();  } catch(int &e) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the error from the library function is indicated by a return value instead of an exception. Using Microsoft Visual Studio (or GCC) to compile both the library and the application would also be a compliant solution because the same exception-handling machinery and ABI would be used on both sides of the execution boundary. |
| // library.h  int func() noexcept(true); // Implemented by the library    // library.cpp  int func() noexcept(true) {  // ...  if (/\* ... \*/) {  return 42;  }  // ...  return 0;  }    // application.cpp  #include "library.h"    void f() {  if (int err = func()) {  // Handle error  }  } |

| **Principles(s):** 2, 3, 10 – code should be architected to adhere to the best practices of the suggested programming language. Code should be architected to ensure that secure standards are followed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++3809, C++3810** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-ERR59-a** | Do not throw an exception across execution boundaries |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C++: ERR59-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr59cpp.html) | Checks for exceptions raised from library interfaces (rule partially covered). |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-010-CPP] | Do not use pointer-to-member operators to access nonexistent members  The pointer-to-member operators .\* and ->\* are used to obtain an object or a function as though it were a member of an underlying object.  Do not use a pointer-to-member expression where the dynamic type of the first operand does not contain the member to which the second operand refers, including the use of a null pointer-to-member value as the second operand. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a pointer-to-member object is obtained from D::g but is then upcast to be a B::\*. When called on an object whose dynamic type is D, the pointer-to-member call is well defined. However, the dynamic type of the underlying object is B, which results in undefined behavior. |
| struct B {  virtual ~B() = default;  };    struct D : B {  virtual ~D() = default;  virtual void g() { /\* ... \*/ }  };    void f() {  B \*b = new B;    // ...    void (B::\*gptr)() = static\_cast<void(B::\*)()>(&D::g);  (b->\*gptr)();  delete b;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the upcast is removed, rendering the initial code ill-formed and emphasizing the underlying problem that B::g() does not exist. This compliant solution assumes that the programmer's intention was to use the correct dynamic type for the underlying object. |
| struct B {  virtual ~B() = default;  };    struct D : B {  virtual ~D() = default;  virtual void g() { /\* ... \*/ }  };    void f() {  B \*b = new D; // Corrected the dynamic object type.    // ...  void (D::\*gptr)() = &D::g; // Moved static\_cast to the next line.  (static\_cast<D \*>(b)->\*gptr)();  delete b;  } |

| **Principles(s):** 2, 3, 4, 10 -Designs should be kept as simple as possible and follow the programming language’s guidelines. Compiler errors should be adhered to and the code modified to comply to ensure that security vulnerabilities are not introduced. The code should adhere to the best practices of the selected programming language. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **overflow\_upon\_dereference invalid\_function\_pointer** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-OOP55** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.MEM.UVAR** | Uninitialized Variable |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **DF2810, DF2811, DF2812, DF2813, DF2814** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2023.1 | **CERT.OOP.PTR\_MEMBER.NO\_MEMBER** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-OOP55-a** | A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: OOP55-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop55cpp.html) | Checks for pointers to member accessing non-existent class members (rule fully covered). |

### Defense-in-Depth Illustration

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





The standards in this policy are enforced in all stages, however, special attention is given in the Assess and Plan stage, the Design Stage, and the Verify and test stage to ensure that the application is successful while it is running in production, and before it gets to production. The Design and Verify and Test stages are critical not only for the initial build of the application, but also for when the code is updated or modified as the existing plan will help to ensure that new security vulnerabilities are not introduced.

Automation is the key to ensuring that security is a part of the software development process, along with secure coding and unit testing in the development stage, and testing automation. Automated security scans can scan additional code that is added during the continuous development/integration flow. Automated scans can check for common flaws that are found in the Open Web Application Security Project’s list and alert to these issues before the code is deployed to production. Automation will help to find issues in the production code such vulnerability updates that may be necessary in order to keep third party framework software up to date.

### Summary of Risk Assessments

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-002-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-003-CLG | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-005-CPP | High | Likely | Medium | P18 | L4 |
| STD-006-CLG | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-009-CPP | High | Probable | Medium | P12 | L1 |
| STD-010-CPP | High | Probable | High | P6 | L2 |

### 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 the act of encrypting data that is being stored while not in use, and sensitive data must have a form of encryption. The purpose of this is to create a layer of defense that the attacker must breach to read the data if it is obtained after previous layers have been defeated. Various encryption methods are available depending on the use of the data, and encryption at rest secures data from those who are not authorized to read it. |
| Encryption at flight | Data in flight is data that is traveling or data that is being transmitted. Data that is being transmitted should be done with a VPN and behind a firewall. Data that is in flight should be encrypted while being transmitted as if a malicious attacker tries to intercept it, it will be inaccessible to them. Data in flight can be protected with the use of SSL certificates that have private and public key. HTTPS and SSL/TLS protocols should be used to protect data that is encrypted in transit. |
| Encryption in use | Data that is in use, such as sensitive data such as credit card numbers and passwords, should be encrypted the entire time that they are in use, ensuring that sensitive data is never left unsecured. Data in use should be protected by security tools that monitor for breach detection. Data that it is use should be protected by a role based access system that limits it’s access to only those who need it. |

| 1. **Triple-A Framework** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication will describe how to identify a user and ensure that that user is who they claim to be. A user must have a valid username and a password before being allowed access, whether when using a login, or accessing a database or files. Passwords can be used as well as single sign on systems, biometrics, digital certificates, and public key infrastructures. |
| Authorization | The next level of authentication is authorization, which describe what the user is allowed to do, or what they are authorized to perform. The user’s level of access should be role based, giving them only the permissions necessary to accomplish a task according to their role. Every user will not need total access, and Administrator access should be given only to those who are making changes to the database and adding new users. |
| Accounting | Accounting defines what resources were accessed, at what time, and who accessed them, as well as any commands that were issued. Accounting can measure access to the network, session statistics and duration, and the data that was sent and received. Accounting will assure that administrators will be able to view who performed what action, when it was performed, and what they performed at a specific time. |

## 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 | 05/21/2023 | Valerie Smith | [Insert text.] | [Insert text.] |
| 1.2 | 06/9/2023 | Valerie Smith | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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