**Green Pace Developer: Security Policy Guide**

**6-2 Project One: Security Policy, Ryan McFarland**



# Green Pace Secure Development Policy

## Contents

[Green Pace Secure Development Policy 0](#_Toc95651626)

[Contents 1](#_Toc95651627)

[Overview 2](#_Toc95651628)

[Purpose 2](#_Toc95651629)

[Scope 2](#_Toc95651630)

[Module Three Milestone 2](#_Toc95651631)

[Ten Core Security Principles 2](#_Toc95651632)

[C/C++ Ten Coding Standards 3](#_Toc95651633)

[Coding Standard 1 4](#_Toc95651634)

[Coding Standard 2 6](#_Toc95651635)

[Coding Standard 3 8](#_Toc95651636)

[Coding Standard 4 10](#_Toc95651637)

[Coding Standard 5 12](#_Toc95651638)

[Coding Standard 6 14](#_Toc95651639)

[Coding Standard 7 16](#_Toc95651640)

[Coding Standard 8 18](#_Toc95651641)

[Coding Standard 9 20](#_Toc95651642)

[Coding Standard 10 22](#_Toc95651643)

[Defense-in-Depth Illustration 24](#_Toc95651644)

[Summary of Risk Assessments 25](#_Toc95651645)

[Create Policies for Encryption and Triple A 25](#_Toc95651646)

[Audit Controls and Management 26](#_Toc95651647)

[Enforcement 26](#_Toc95651648)

[Exceptions Process 26](#_Toc95651649)

[Distribution 27](#_Toc95651650)

[Policy Change Control 27](#_Toc95651651)

[Policy Version History 27](#_Toc95651652)

[Appendix A Lookups 27](#_Toc95651653)

[Approved C/C++ Language Acronyms 27](#_Toc95651654)

## 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 | Validation begins with identifying all input sources. The goal is preventing injection or over/under flow. Data should be correlated with prepared statements if possible and the prepared statements should be used. Exception handling should be used to handle improper inputs into alphanumeric fields. |
| 1. Heed Compiler Warnings | Compiler warnings provide useful information about where problematic code can be found. This information should be acted on, rather than ignored. The compiler is likely correct while the programmer is likely wrong, as far as compiler warnings are concerned. |
| 1. Architect and Design for Security Policies | Designs must include security considerations into every decision. A design without concern for security will inevitably need to be redesigned. A modular design is best so that security is easier to assess. |
| 1. Keep It Simple | The least amount of code is the easiest to use. |
| 1. Default Deny | When access modeling occurs, default deny will deny access to any element not specifically granted access. |
| 1. Adhere to the Principle of Least Privilege | Every function will be granted the lowest level of access necessary to perform the intended purpose. Access is not granted unless necessary. |
| 1. Sanitize Data Sent to Other Systems | Related to input validation. Ensuring data conforms to the requirements of the system it is passed to and is not leaking data. |
| 1. Practice Defense in Depth | Defense in depth is creating layers of security of different types so that it decreases the likelihood of a successful attack. Defense in depth can also be useful in reducing the impact of an attack. Detecting abnormal memory behavior used with secure coding is one example of defense in depth. If one fails, there are other attempts at defending the vulnerability. |
| 1. Use Effective Quality Assurance Techniques | Use assertions to check expected behavior. Address compiler warnings. Use analysis tools to get further information about problems. |
| 1. Adopt a Secure Coding Standard | A secure coding standard makes it easier for programmers to understand exactly what is requested to write code that is of the quality expected in the project. The quality of the code produced should be high if the standards are developed properly. |

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Ensure that unsigned integer operations do not wrap. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example can result in an unsigned integer wrap during the addition of the unsigned operands ui\_a and ui\_b. If this behavior is unexpected, the resulting value may be used to allocate insufficient memory for a subsequent operation or in some other manner that can lead to an exploitable vulnerability. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution performs a precondition test of the operands of the addition to guarantee there is no possibility of unsigned wrap: |
| #include <limits.h>    void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum;  if (UINT\_MAX - ui\_a < ui\_b) {  /\* Handle error \*/  } else {  usum = ui\_a + ui\_b;  }  /\* ... \*/  } |

| **Principles(s): 1:** ValidateInput Data, this maps directly as it concerns input validation. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | integer-overflow | Fully checked |
| Axivivion Bauhaus Suite | 7.2.0 | CertC-INT30 | Implemented |
| CodeSonar | 6.2p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  MISC.MEM.SIZE.ADDOFLOW  MISC.MEM.SIZE.BAD  MISC.MEM.SIZE.MULOFLOW  MISC.MEM.SIZE.SUBUFLOW | Addition overflow of allocation size  Integer overflow of allocation size  Multiplication overflow of allocation size  Subtraction underflow of allocation size  Addition overflow of size  Unreasonable size argument  Multiplication overflow of size  Subtraction underflow of size |
| Compass/ROSE | -- | -- | Can detect violations of this rule by ensuring that operations are checked for overflow before being performed (Be mindful of exception INT30-EX2 because it excuses many operations from requiring validation, including all the operations that would validate a potentially dangerous operation. For instance, adding two unsigned ints together requires validation involving subtracting one of the numbers from UINT\_MAX, which itself requires no validation because it cannot wrap.) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Ensure that operations on signed integers do not result in overflow. |

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

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

| **Principles 1&2:** Practice Defense in Depth & ValidateInput Data, assumptions shouldn’t be made about data. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Integer-overflow | Fully Checked |
| CodeSonar | 6.2p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  MISC.MEM.SIZE.ADDOFLOW  MISC.MEM.SIZE.BAD  MISC.MEM.SIZE.MULOFLOW  MISC.MEM.SIZE.SUBUFLOW | Addition overflow of allocation size  Integer overflow of allocation size  Multiplication overflow of allocation size  Subtraction underflow of allocation size  Addition overflow of size  Unreasonable size argument  Multiplication overflow of size  Subtraction underflow of size |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| Helix QAC | 2021.3 | C2800, C2801, C2802, C2803, C2860, C2861, C2862, C2863  C++2800, C++2801, C++2802, C++2803, C++2860, C++2861, C++2862, C++2863 | -- |

#### Coding Standard 3

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

| **Noncompliant Code** |
| --- |
| Because the input is unbounded, the following code could lead to a buffer overflow. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

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

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

| **Principles(s):** 1, 2, 3, 4. These principles apply due to the emphasis on validation, simplicity, and designing with security in mind. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Rule 7 STR31-C  High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | -- | Supported  Astrée reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data. |
| 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 | 6.2p0 | 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 |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Utilize parameters over raw string queries to access data sources. |

| **Noncompliant Code** |
| --- |
| This code is non-compliant. The code injects raw strings into the query engine and can allow users to enter additional items in the string and cause it to fail. |
| #include <iostream>  #include <sqlite3.h>  void func()  {  sqlite3\* DB;  int exit = 0;  exit = sqlite3\_open("example.db", &DB);  string sql("INSERT INTO PERSON VALUES(1, 'STEVE', 'GATES', 30, 'PALO ALTO', 1000.0);"  exit = sqlite3\_exec(DB, sql.c\_str(), NULL, 0, &messaggeError);    if (exit != SQLITE\_OK) {  std::cerr << "Error Insert" << std::endl;  sqlite3\_free(messaggeError);  }  } |
|  |

| **Compliant Code** |
| --- |
| This code is compliant as it uses parameters instead of raw strings to insert data into the query. Its much harder to succumb to SQL injection style attacks. |
| #include <iostream>  #include <sqlite3.h>  void func(){  sqlite3\* DB;  int exit = 0;  exit = sqlite3\_open("example.db", &DB);  string sql("INSERT INTO PERSON VALUES(1, 'STEVE', 'GATES', 30, 'PALO ALTO', 1000.0);"  exit = sqlite3\_exec(DB, sql.c\_str(), NULL, 0, &messaggeError);  if (exit != SQLITE\_OK) {  std::cerr << "Error Insert" << std::endl;  sqlite3\_free(messaggeError);  }  } |
|  |

| **Principles(s):** 1, 4, 7, 8. These principles apply in this scenario because it concerns input validation, defense in depth, simplicity, and data sanitation. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| FIO30-C  High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO30 | Partially implemented |
| CodeSonar | 6.2p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| GCC | 4.3.5 | -- | Can detect violations of this rule when the -Wformat-security flag is used |
| Parasoft C/C++test | 2021.2 | CERT\_C-FIO30-a  CERT\_C-FIO30-b  CERT\_C-FIO30-c | Avoid calling functions printf/wprintf with only one argument other than string constant  Avoid using functions fprintf/fwprintf with only two parameters, when second parameter is a variable  Never use unfiltered data from an untrusted user as the format parameter |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. 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** |
| --- |
| 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):** 4, 8. These principles are related to simplicity and defense in depth and apply in this example. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High  MEM30-C | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2021.2 | CERT\_C-MEM30-a | Do not use resources that have been freed |
| Parasoft Insure++ | -- | -- | Runtime analysis |
| TrustInSoft Analyzer | 1.38 | dangling\_pointer | Exhaustively verified |
| Compass/ROSE | -- | -- | -- |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Do not use assertions in production code. |

| **Noncompliant Code** |
| --- |
| Use assertions only for development code, this code would not be accepted for deployment. |
| void doStuff(){  std::cin >> stuff;  assert(stuff != 0);  std::cout << (stuff -1);  } |

| **Compliant Code** |
| --- |
| This compliant code has the unnecessary assertion removed. |
| void doStuff(){  std::cin >> stuff;  std::cout << (stuff -1);  } |

| **Principles(s):** 3, 4, 6, 8 all apply due to the design, simplicity, least privilege, and defense in depth implications. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Not Likely | Low | Low | Low |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| N/A | -- | -- | -- |
| -- | -- | -- | -- |
| -- | -- | -- | -- |
| -- | -- | -- | -- |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

| **Principles(s):** 1, 3, 4, 8. These principles apply because of the relation to input validation, designing for security, simplicity, and defense in depth. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Klocwork | 2021.4 | MISRA.CATCH.ALL | -- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | **C++4035, C++4036, C++4037** | -- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-008-CPP] | Use valid iterator ranges |

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

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

| **Principles(s):** 1, 2, 3, 4, 5, 8. These principles apply because of the effects that undefined behavior can have. Simplicity is likely the most important of these in this case. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | **C++3802** | -- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-CTR53-a** **CERT\_CPP-CTR53-b** | Do not use an iterator range that isn't really a range Do not compare iterators from different containers |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.17 | [**V539**](https://pvs-studio.com/en/docs/warnings/v539/), [**V662**](https://pvs-studio.com/en/docs/warnings/v662/), [**V789**](https://pvs-studio.com/en/docs/warnings/v789/) | -- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **overflow\_upon\_dereference** | -- |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Environment | [STD-009-CPP] | Do not call system() |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the system() function is used to execute any\_cmd in the host environment. |
| #include <string.h>  #include <stdlib.h>  #include <stdio.h>  enum { BUFFERSIZE = 512 };  void func(const char \*input){  char cmdbuf[BUFFERSIZE];  int len\_wanted = snprintf(cmdbuf, BUFFERSIZE,  "any\_cmd '%s'", input);  if (len\_wanted >= BUFFERSIZE) {  /\* Handle error \*/  } else if (len\_wanted < 0) {  /\* Handle error \*/  } else if (system(cmdbuf) == -1) {  /\* Handle error \*/  }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses the Microsoft Windows CreateProcess() API: |
| #include <Windows.h>    void func(TCHAR \*input) {  STARTUPINFO si = { 0 };  PROCESS\_INFORMATION pi;  si.cb = sizeof(si);  if (!CreateProcess(TEXT("any\_cmd.exe"), input, NULL, NULL, FALSE,  0, 0, 0, &si, &pi)) {  /\* Handle error \*/  }  CloseHandle(pi.hThread);  CloseHandle(pi.hProcess);  } |

| **Principles(s):** 2, 3, 4, 6, 8, 9, 10. These principles apply to this standard directly as it concerns heeding compiler warnings, secure design, simplicity, privilege management, defense in depth, quality assurance, and security standardization. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | **CERT\_C-ENV33-a** | Do not call the 'system()' function from the 'stdlib.h' or 'cstdlib' library with an argument other than '0' (null pointer) |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151949) | 3.11 | [**S990**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-c.html#RSPEC-990) | Detects uses of "abort", "exit", "getenv" and "system" from <stdlib.h> |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | cert-env33-c | Checked by clang-tidy |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **stdlib-use-system** | Fully checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Pseudo randomization | [STD-010-CPP] | Do not use std::rand() for generating pseudorandom numbers |

| **Noncompliant Code** |
| --- |
| The following noncompliant code generates an ID with a numeric part produced by calling the rand() function. The IDs produced are predictable and have limited randomness. Further, depending on the value of RAND\_MAX, the resulting value can have modulo bias. |
| #include <cstdlib>  #include <string>    void f() {  std::string id("ID");  // Holds the ID, starting with the characters "ID" followed  // by a random integer in the range [0-10000].  id += std::to\_string(std::rand() % 10000);  // ...  } |

| **Compliant Code** |
| --- |
| The C++ standard library provides mechanisms for fine-grained control over pseudorandom number generation. It breaks random number generation into two parts: one is the algorithm responsible for providing random values (the engine), and the other is responsible for distribution of the random values via a density function (the distribution). The distribution object is not strictly required, but it works to ensure that values are properly distributed within a given range instead of improperly distributed due to bias issues. This compliant solution uses the Mersenne Twister algorithm as the engine for generating random values and a uniform distribution to negate the modulo bias from the noncompliant code example |
| #include <random>  #include <string>  void f() {  std::string id("ID"); // Holds the ID, starting with the characters "ID" followed  // by a random integer in the range [0-10000].  std::uniform\_int\_distribution<int> distribution(0, 10000);  std::random\_device rd;  std::mt19937 engine(rd());  id += std::to\_string(distribution(engine));  // ...  } |

| **Principles(s):** 1, 3, 4, 6, 7, 9, 10. These principles apply to this standard due to the related nature of input validation, designing securely, simplicity, managing privileges, data sanitation, quality assurance, and standard implementation. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | **bad-function (AUTOSAR.26.5.1A)** | Fully checked |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-MSC50-a** | Do not use the rand() function for generating pseudorandom numbers |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2021.4 | [**CERT.MSC.STD\_RAND\_CALL**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) | -- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 4.0 (prerelease) | cert-msc50-cpp | Checked by clang-tidy |

### Defense-in-Depth Illustration

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





As it exists currently, the DevSecOps process involves assessing and planning, designing, building, verifying, transition testing, monitoring, responding, and maintaining and stabilizing. My suggestion would be to include an automated detection tool into the workflow. This would fit best in the building and verification and testing phases. Additionally, it would make sense to periodically retest production code with automated detection tools such as clang or cppcheck. The tools could be integrated into the workflow by simply including them during the verification and testing phase along with vulnerability scanning with products like Metasploit. This would fall in line with the stated goals of the verification and trust phase of functional, compliancy, and security testing.

### Summary of Risk Assessments

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | High | Likely | High | High | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Not Likely | Low | Low | 1 |
| STD-007-CPP | Low | Probable | Medium | Low | 3 |
| STD-008-CPP | High | Probable | High | Medium | 2 |
| STD-009-CPP | High | Probable | Medium | Medium | 1 |
| STD-010-CPP | Medium | Unlikely | Low | Low | 2 |

### Create Policies for Encryption and Triple A

| 1. **Encryption** | **Definition and application** |
| --- | --- |
| Encryption in rest | Stored passwords and similar must be stored in a secure manner. Data must be encrypted so that unauthorized access is less likely to result in useable data for the perpetrators**. Encryption schemes will change as technology changes.** Schemes that result in significant delays to access data are appropriate if data will not be accessed routinely. |
| Encryption at flight | Communications must be conducted in a secure manner starting with methods to confirm the recipient identity. Information sent must be obfuscated sufficiently to prevent successful interception of data in transit by those not privy. |
| Encryption in use | Data in use will be obfuscated whenever feasible. Data used heavily will be encrypted while at rest and unencrypted during use with an eye toward using schemes that allow for faster retrieval. Speed will not be achieved at the expense of effective encryption. |

| 1. **Triple-A Framework** | **Definition and application** |
| --- | --- |
| Authentication | Identifying a user. This applies because it is necessary to identify users in order to administer an effective security policy. This typically occurs during a login process of some sort which could be as simple as remembering which devices are registered by the user. |
| Authorization | After being identified the user identity will determine what authorization is conferred upon the user. When appropriate this may be determined by attributes such as the user location, device, or some similar data. Authorization may also be determined on a case-by-case basis. |
| Accounting | Accounting is the act of logging every transaction. These transactions may need to be scrutinized further to establish whether malicious activity has occurred within a system. If poor accounting has taken place this will make it difficult to determine root cause for security events. Accounting must log all system activity to be effective. This logging should include information such as files accessed by users. |

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum | -- |
| 2.0 | 02/13/2022 | Revised | Ryan McFarland | -- |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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