**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 validation must happen because wrong inputs could crash the system due to wrong input type, or perhaps cause a buffer overflow. |
| 1. Heed Compiler Warnings | It is best to enable compiler warnings in our IDE so that we might be aware of problems within our code. |
| 1. Architect and Design for Security Policies | “Never trust, always verify” is an idea behind the Zero Trust Architecture. Designs can incorporate and be built around security policies during the software development lifecycle, which is a better idea than applying security designs after the fact. |
| 1. Keep It Simple | C++ can be a complex language and with that complexity adds it’s strength, but we want to cut out any unnecessary complexity in favor of making it easier to understand and work with. |
| 1. Default Deny | This is an idea that any action which is not specifically allowed will be denied. This works in conjunction with the principle of least privilege, where only those processes or individuals that specifically intended to receive the resource are given access.  An example of this can also be found in a firewall, where specific programs are allowed to communicate over the web and everything else is denied. |
| 1. Adhere to the Principle of Least Privilege | Only give access to those that need it, and only give the resources necessary for the individual or process to fulfill it’s function. |
| 1. Sanitize Data Sent to Other Systems | Data can be accepted or sent that causes unwanted processes to happen, like a string that includes characters that would be handled differently than intended (example: SQL injection). Sanitization is the process of removing this capability from the data or denying it from being processed. We want to validate and sanitize data anytime it is handled. |
| 1. Practice Defense in Depth | Defense in depth means that we stack as many protective layers against attacks as possible. If one layer fails, the attack must still get through others. |
| 1. Use Effective Quality Assurance Techniques | Integrated Development Environments will typically have tools to analyze code quality. We can utilize Visual Studio, cppcheck, and other resources to check for code quality issues. We can also utilize coding standards (like SEI CERT C++) to improve the overall function of our application. |
| 1. Adopt a Secure Coding Standard | We need to start our application development with secure coding in mind, from beginning to end of the development lifecycle. I believe we can also utilize SEI CERT C++ coding standard [here](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682). |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Use correct data type for necessary data precision. For example, with INT35-C we need to use correct integer precisions that offer the size and precision of the data we’ll calculate, but not use excessive memory. Also, we want to make sure that data type conversions do not result in lost or misinterpreted data (INT31-C), as with the example provided with unsigned to signed values. |

| **Noncompliant Code** |
| --- |
| Errors can occur when converting from unsigned to signed int. Sourced: https://wiki.sei.cmu.edu/confluence/display/c/INT31-C.+Ensure+that+integer+conversions+do+not+result+in+lost+or+misinterpreted+data |
| #include <limits.h>  **void** func(**void**) {    unsigned **long** **int** u\_a = ULONG\_MAX;  **signed** **char** sc;    sc = (**signed** **char**)u\_a; /\* Cast eliminates warning \*/    /\* ... \*/  } |

| **Compliant Code** |
| --- |
|  |
| #include <limits.h>    **void** func(**void**) {    unsigned **long** **int** u\_a = ULONG\_MAX;  **signed** **char** sc;  **if** (u\_a <= SCHAR\_MAX) {      sc = (**signed** **char**)u\_a;  /\* Cast eliminates warning \*/    } **else** {      /\* Handle error \*/    }  } |

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

| **Principles(s):**  Validate Input Data: we need to watch for input type validation. A assert() check could be an option to validate input type, and certainly a Gtest.  Heed Compiler Warnings: This, along with CppCheck are useful tools in finding errors including input type.  Keep it simple: if we adhere to a principle of keeping code simple and easy to read, it can be easier to spot simple data type problems. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Necessary data ranges, sizes, and required precision should be accounted for in picking the type of memory we plant to store it in. We want to avoid overflows but also not utilize excessive memory to store smaller values. INT32-C |

| **Noncompliant Code** |
| --- |
| This simple subtraction can result in an overflow and should probably be unsigned int. Source: https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow |
| **void** func(**signed** **int** si\_a, **signed** **int** si\_b) {  **signed** **int** diff = si\_a - si\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Error checking to make sure that overflow does not occur can be utilized. |
| #include <limits.h>    **void** func(**signed** **int** si\_a, **signed** **int** si\_b) {  **signed** **int** diff;  **if** ((si\_b > 0 && si\_a < INT\_MIN + si\_b) ||        (si\_b < 0 && si\_a > INT\_MAX + si\_b)) {      /\* Handle error \*/    } **else** {      diff = si\_a - si\_b;    }      /\* ... \*/  } |

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

| **Principles(s):**  Validate Input Data: similar to input type, we also need to avoid buffer overflows or other memory issues if the data is outside of allocated values.  Heed Compiler Warnings: Again, this, along with CppCheck are useful tools in finding errors including data value.  Keep it simple: And the same goes here, if we adhere to a principle of keeping code simple and easy to read, it can be easier to spot simple data type problems. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | With strings we need to be concerned about buffer overflows. In C we do not store string length. The end is designated with a null character. Functions like strcpy() will copy characters until it hits the null character but if it copies to a variable that doesn’t have enough room, we’ll buffer overflow. |

| **Noncompliant Code** |
| --- |
| In this code, the loop copies 1 byte past the available space for 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** |
| --- |
| The gets() function returns the size of the string so we account for the storage space needed for 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';  } |

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

| **Principles(s):** Validate Input Data, Heed Compiler Warnings, and Keep It Simple. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Strategies to avoid SQL injection might be prepared statements, properly constructed stored procedures, allow-list input validation, and escaping all user supplied input.  STR02-C describes how we should sanitize data passed to complex subsystems so that special characters do not trigger unwanted commands or actions. |

| **Noncompliant Code** |
| --- |
| This is an example of unsanitized data being transferred from the user to the login program. Source: https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152409 |
| (**void**) execl(LOGIN\_PROGRAM, "login",    "-p",    "-d", slavename,    "-h", host,    "-s", pam\_svc\_name,    (AuthenticatingUser != NULL ? AuthenticatingUser :  **getenv**("USER")),    0); |

| **Compliant Code** |
| --- |
| This compliant solution inserts the "--" (double dash) argument before the call to getenv("USER") in the call to execl(): |
| (**void**) execl(LOGIN\_PROGRAM, "login",    "-p",    "-d", slavename,    "-h", host,    "-s", pam\_svc\_name,    "--",    (AuthenticatingUser != NULL ? AuthenticatingUser :  **getenv**("USER")), 0); |

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

| **Principles(s):** with defense in-depth we can use a layered strategy of secure coding.  Validate Input Data: We can validate inputs to make sure they do not include certain inputs.  Sanitize Data Sent to Other Systems: We can convert characters to harmless encoded characters or avoid concatenated strings. There are many methods to sanitize data to stop SQL injection. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | It is good practice to avoid undefined behavior when considering memory. For example, deleting a pointer and trying to access it later can cause undefined behavior (called a dangling pointer). Utilizing smart pointers can be a great option beyond avoiding dangling pointers where they may exist only as the pointer is needed.  MEM50-CPP Do not access freed memory.  MEM51-CPP Properly deallocate dynamically allocated resources.  MEM52-CPP Detect and handle memory allocation errors.  MEM53-CPP Explicitly construct and destruct objects when manually managing object lifetime.  MEM54-CPP Provide placement new with properly aligned pointers to sufficient storage capacity.  MEM55-CPP Honor replacement dynamic storage management requirements  MEM56-CPP Do not store an already-owned pointer value in an unrelated smart pointer.  MEM57-CPP Avoid using default operator new for over-aligned types. |

| **Noncompliant Code** |
| --- |
| New and delete. This would be an example of memory that is not deallocated. |
| #include <new>  struct S {   void f();  };  void g() noexcept(false) {   S \*s = new S;   // ...   delete s;   // ...   s->f();  } |

| **Compliant Code** |
| --- |
| #include <new>  struct S {   void f();  };  void g() noexcept(false) {   S \*s = new S;   // ...   s->f();   delete s;  } |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

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

| **Principles(s):**  Heed Compiler Warnings: In this situation we have a coding error that should show up in CppCheck or as a compiler warning that it is missing a destructor. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | It is important to perform diagnostic testing using assertions. The assert() macro is a convenient way to perform testing but should not have side effects. If the result of the assert() macro is 0 it will call abort() |

| **Noncompliant Code** |
| --- |
| Assert() macro with side effect |
| #include <assert.h>  #include <stddef.h>  void process(size\_t index) {   assert(index++ > 0); /\* Side effect \*/   /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Assert() no longer contains side effect |
| #include <assert.h>  #include <stddef.h>  void process(size\_t index) {   assert(index > 0); /\* No side effect \*/   ++index;   /\* ... \*/  } |

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

| **Principles(s):**  Validate Input Data: The Assert() macro is used to validate data.  Heed Compiler Warnings: however, in this situation we might have an unintended side effect from the iterator. A compiler warning might catch this. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Exceptions and error handling should not abruptly terminate the program. Doing so may result in not calling destructors and leaving memory entangled. Abruptly terminating an application should only happen in direct response to a critical error.  ERR50-CPP Do not abruptly terminate the program.  ERR51-CPP Handle all exceptions.  ERR52-CPP Do not use setjmp() or longjmp().  ERR53-CPP Do not reference base classes or class data members in a constructive or destructor function-try-block handler.  ERR54-CPP Catch handlers should order their parameter types from most derived to least derived.  ERR55-CPP Honor exception specifications.  ERR56-CPP Guarantee exception safety.  ERR57-CPP Do not leak resources when handling exceptions.  ERR58-CPP Handle all exceptions thrown before main() begins executing.  ERR59-CPP Do not throw an exception across execution boundaries.  ERR60-CPP Exception objects must be nothrow copy constructible.  ERR61-CPP Catch exceptions by lvalue reference.  ERR62-CPP Detect errors when converting to string to a number. |

| **Noncompliant Code** |
| --- |
| f(), which was registered as an exit handler with std::at\_exit(), may result in a call to std::terminate() because  throwing\_func() may throw an exception. |
| #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** |
| --- |
| #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   }   // ...  } |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

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

| **Principles(s):** Use Effective Quality Assurance Techniques. Exceptions should be handled without abruptly terminating the application. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Input/Output** | STD-008-CPP | Specifically with file streams we need to close them once we are finished moving data.  FIO51-CPP Close files when they are no longer needed.  A call to the std::basic\_filebuf<T>::open() function must be matched with a call to  std::basic\_filebuf<T>::close() before the lifetime of the last pointer that stores the  return value of the call has ended or before normal program termination, whichever occurs first. |

| **Noncompliant Code** |
| --- |
| Instead of calling file.close() this noncompliant code calls terminate(). Destructors will not execute so the basic\_filebuf<T> object maintained by the object is not properly closed. |
| #include <exception>  #include <fstream>  #include <string>  void f(const std::string &fileName) {   std::fstream file(fileName);   if (!file.is\_open()) {   // Handle error   return;   }   // ...   std::terminate();  } |

| **Compliant Code** |
| --- |
| #include <exception>  #include <fstream>  #include <string>  void f(const std::string &fileName) {   std::fstream file(fileName);   if (!file.is\_open()) {   // Handle error   return;   }   // ...   file.close();   if (file.fail()) {   // Handle error   }   std::terminate();  } |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

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

| **Principles(s):**  Heed Compiler Warnings: Forgetting to write in code to close a file stream should be caught by the compiler. We might also see this issue in CppCheck. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Containers** | STD-009-CPP | Somewhat similar to a data type buffer overflow we also need to make sure that objects iterated within an array do not fall outside of the array table size.  CTR50-CPP Guarantee that containers indices and iterators are within the valid range. |

| **Noncompliant Code** |
| --- |
| In this example there are two values, pos and value. Pos is checked to make sure it does not exceed the upper bounds of tableSize, but not the lower. |
| #include <cstddef>  void insert\_in\_table(int \*table, std::size\_t tableSize, int pos,  int value) {   if (pos >= tableSize) {   // Handle error   return;   }   table[pos] = value;  } |

| **Compliant Code** |
| --- |
| #include <cstddef>  void insert\_in\_table(int \*table, std::size\_t tableSize, std::size\_t  pos, int value) {   if (pos >= tableSize) {   // Handle error   return;   }   table[pos] = value;  } |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

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

| **Principles(s):** Validate Input Data to make sure that it will fit in the object, vector, table, etc., that it intends to be saved in. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Floating Point** | STD-010-CPP | Floating-point numbers cannot represent all real numbers exactly. They are also represented by large data values and are not the best use of memory.  To keep code portable, floating-point variables must not be used as the loop induction variable.  FLP30-C Do not use floating-point variables as loop counters. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a floating-point variable is used as a loop counter. The decimal number 0.1 is a repeating fraction in binary and cannot be exactly represented as a binary floating-point number. Depending on the implementation, the loop may iterate 9 or 10 times. |
| void func(void) {   for (float x = 0.1f; x <= 1.0f; x += 0.1f) {   /\* Loop may iterate 9 or 10 times \*/   }  } |

| **Compliant Code** |
| --- |
| #include <stddef.h>  void func(void) {   for (size\_t count = 1; count <= 10; ++count) {   float x = count / 10.0f;   /\* Loop iterates exactly 10 times \*/   }  } |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

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

| **Principles(s):**  Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Defense-in-Depth Illustration

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



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Likely | High | P9 | L2 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-009-CPP | High | Likely | High | P9 | L2 |
| STD-010-CPP | Low | Probable | Low | 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 in rest | Default encryption at rest is the idea that that we protect data that is stored on disk (or SSDs). A typical encryption would be the Advanced Encryption Standard (AES) 256, however there are other methods.  This adds a protective layer to data at rest. AES encryption would take long enough to break that it would mostly render the data useless by the time they were successful (about one billion years times an order of magnitude). |
| Encryption at flight | Data that moves over a network is particularly vulnerable. Encryption is performed by encrypting packets independently. Transport encryption uses two symmetric AES keys, randomly generated. Dual-key encryption and decryption is done at the remote end. |
| Encryption in use | Encryption in use is the principle of protecting data when it is being utilized. So files that are currently open, databases, and RAM data needs to be decrypted, and then encrypted when saved, there needs to be a good authentication mechanism. Single Sign-On (SSO) and Multi-Factor Authentication (MFA) can be implemented to increase security.  It is important to have good access management of these resources to increase security. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is the process of making sure we can identify the user with valid credentials. Two-factor authentication might be an added security policy to ensure authentication of the user. |
| Authorization | Along with authentication we should attach an authorization to the user. With this we can match just the resources the user needs and restrict anything they do not. This goes along with the Default Deny principle, if they do not specifically have authentication and authorization, resources should be denied. |
| Accounting | Accounting of user actions can be a good way to look for suspicious use of the system. We might track actions, or even how fast actions are being performed to make sure that the user is human and performing normal activities. |

**\***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 | n/a | Initial Milestone | Dustin Haugh | [Insert text.] |
| 1.2 | n/a | Security Policy Completion | Dustin Haugh | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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