**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 should never be trusted by default. Always ensure the data is valid, is as expected, and will not cause overflows or injections that can expose sensitive information or disrupt the functioning of the system or program. |
| 1. Heed Compiler Warnings | Warnings should never be ignored, as deprecated code and non-standard practices can cause problems or become a risk later even if the program functions with them. |
| 1. Architect and Design for Security Policies | Security Policies should be followed during the design process so that safety is in mind from the very concept of a software. Policies are developed to give specific rules that prevent threats. |
| 1. Keep It Simple | Simple methods are easier to fix, detect bugs in, and monitor. Choose simple ideas and solutions over complex ones. More complexity not only uses more time and memory but offers more to exploit and more difficulty when tracing bugs. |
| 1. Default Deny | Always deny access by default unless credentials and privilege level are passed. Do not write conditionals set to pass by default unless a check fails – do the opposite and deny until it passes. Use whitelisting to grant access and do not trust any device or user. |
| 1. Adhere to the Principle of Least Privilege | Only allow the necessary access to data and parts of the program. If it does not need to be public, make it private. If a user does not need to access it, make it admin-level only. If an admin does not need it to do their job, do not make it accessible to the admin. |
| 1. Sanitize Data Sent to Other Systems | Do not leave copies of data behind that are unnecessary when sending it to other systems. Whether physical or digital, ensure that data is destroyed completely, encrypted or masked - not just deleted and sent to a recycle bin – so that it does not fall into the wrong hands. |
| 1. Practice Defense in Depth | Do not rely on one layer of security alone. The system should be set up so that if one method fails, another is behind it to protect the system from an attack or disruption. Include protection on the physical, network, application and data levels. |
| 1. Use Effective Quality Assurance Techniques | Use techniques that allow users to communicate freely about problems with the software and do not ignore them. Have a plan in place to handle reports and complaints as well as fix bugs in a timely fashion. |
| 1. Adopt a Secure Coding Standard | Follow a set of standards in all development that are proven to prevent or safely mitigate against damage or disruption to the system and its users and stakeholders. Ensure that they are followed during the process and before releasing software. |

### 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]  SEI Ref STR-50-CPP | [Guarantee that storage for strings has sufficient space for character data and the null terminator.](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator) Sending information into a buffer that is not large enough results in overflow, which can be exploited to access private data or crash the program. Ensure that c-strings and basic\_strings have enough room for data that may be entered. |

| **Noncompliant Code** |
| --- |
| Dumping directly from input into a 12-character array could easily overflow. |
| void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The best way to prevent buffer overflow is to dump any potential overflow into another string. |
| #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 & 3– Follow a security standard to validate all input and not trust it by default. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | LIKELY | MEDIUM | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 2.10 | **stream-input-char-array** | Partially checked + soundly supported |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.0p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO** **LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.3 | **NNTS.MIGHT** **NNTS.TAINTED** **NNTS.MUST** **SV.UNBOUND\_STRING\_INPUT.CIN** | n/a |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] SEI ref EXP-33-CPP | Do not read uninitialized memory.  These have undetermined or unexpected values and can disrupt the program’s operation or cause unexpected behavior. |

| **Noncompliant Code** |
| --- |
| The string msg may not have a value if nothing is entered or input is skipped or removed. |
| std::string msg; std::cin >> msg;  std::cout << msg; |

| **Compliant Code** |
| --- |
| The string is initialized as an empty string as a fallback. |
| std::string msg = “ “;  std::cin >> msg;  std::cout << msg; |

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

| **Principles(s):** 2 & 3: Follow good coding standards, heed compiler warnings about uninitialized memory |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | PROBABLE | MEDIUM | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **uninitialized-local-read**  **uninitialized-variable-use** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.0p0 | **LANG.MEM.UVAR** | Uninitialized variable |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **UNINIT** | Implemented |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | **uninitvar** **uninitdata** **uninitstring** **uninitMemberVar** **uninitStructMember** | Detects uninitialized variables, uninitialized pointers, uninitialized struct members, and uninitialized array elements (However, if one element is initialized, then cppcheck assumes the array is initialized.) There are FN compared to some other tools because Cppcheck tries to avoid FP in impossible paths. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] SEI ref STR-51-CPP | Do not attempt to create a ‘string’ from a null pointer.  This can cause undefined or unexpected behavior. Check that strings are never created from null pointers. |

| **Noncompliant Code** |
| --- |
| Getenv() can return a null pointer, so it should not be used to create string tmp. |
| #include <cstdlib>  #include <string>    **void** f() {  std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| The results of getenv() are checked for null value before a string is created from it. |
| #include <cstdlib>  #include <string>    **void** f() {  **const** **char** \*tmpPtrVal = std::**getenv**("TMP");  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  **if** (!tmp.empty()) {  // ...  }  } |

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

| **Principles(s):** 3 & 10: Adopt a security policy and follow secure coding standards to prevent future issues. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | high | medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **assert\_failure** | N/A |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.0p0 | **LANG.MEM.NPD** | Null Pointer Dereference |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.3 | **DF4770, DF4771, DF4772, DF4773, DF4774** | N/A |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2023.3 | **NPD.CHECK.CALL.MIGHT** **NPD.CHECK.CALL.MUST** **NPD.CHECK.MIGHT** **NPD.CHECK.MUST** **NPD.CONST.CALL** **NPD.CONST.DEREF** **NPD.FUNC.CALL.MIGHT** **NPD.FUNC.CALL.MUST** **NPD.FUNC.MIGHT** **NPD.FUNC.MUST** **NPD.GEN.CALL.MIGHT** **NPD.GEN.CALL.MUST** **NPD.GEN.MIGHT** **NPD.GEN.MUST** **RNPD.CALL** **RNPD.DEREF** | N/A |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] SEI Ref STR-02-C | Sanitize data passed into other systems.  Validate and format data that is passed to other systems in order to prevent code injections and other undesired dangerous behavior. |

| **Noncompliant Code** |
| --- |
| This code takes user input and accepts any characters as a string, which is used to log in via database. It is vulnerable to SQL code injections that can destroy or expose sensitive data. |
| #include <iostream> #include <string> using namespace std;  char[12] input; cout << “Enter Username: ”;  cin.getline(input, 12); |

| **Compliant Code** |
| --- |
| Before storing the input, it is checked to ensure characters are alphanumeric. Whitespace, ‘, and = will not be accepted. This avoids SQL injection as well as other code injections. |
| #include <iostream> #include <string> using namespace std;  char[12] input;  char check; cout << “Enter Username: ”; for (i = 0; i < 12; i++){  cin.getline(check);  if (isalnum(check)){  input[i] = check;  }  else {  Cout << “Invalid username.” << endl;  } } |
|  |

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

| **Principles(s):** 7 – Sanitize data sent to other systems. 1 – Validate all input. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | LIKELY | MEDIUM | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | N/A | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.0p0 | **IO.INJ.COMMAND** **IO.INJ.FMT** **IO.INJ.LDAP** **IO.INJ.LIB** **IO.INJ.SQL** **IO.UT.LIB** **IO.UT.PROC** | Command injection Format string injection LDAP injection Library injection SQL injection Untrusted Library Load Untrusted Process Creation |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | **TAINTED\_STRING** | Fully implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2023.4 | **NNTS.TAINTED** **SV.TAINTED.INJECTION** | N/A |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] SEI Ref MEM11-C | Do not assume infinite heap space.  Memory is finite and can be exhausted. Do not allow users to infinitely enter data that is stored in the heap. |

| **Noncompliant Code** |
| --- |
| This code allows users to create an infinite number of ‘Cat’ objects/structs in the heap. |
| #include <iostream> #include Cat.h  #include <vector>  vector<Cat> cats;  using namespace std;  char[20] name;  cout >> “Enter cat name: “;  cin.getline(name, 20);  cats.push\_back(new Cat(name)); |

| **Compliant Code** |
| --- |
| The size of the dynamic vector is checked before new cats are created. |
| #include <iostream> #include Cat.h  #include <vector>  vector<Cat> cats;  using namespace std;  char[20] name;  cout >> “Enter cat name: “;  cin.getline(name, 20);  if (cats.size() < 100) {  cats.push\_back(new Cat(name));  }  else {  Cout << “Sorry, you don’t have room for any more cats.” << endl;  } |

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

| **Principles(s):** 3 & 10 – Follow policies and secure coding standards that prevent later issues. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| LOW | PROBABLE | HIGH | P3 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.0p0 | **ALLOC.LEAK** **IO.TAINT.SIZE** **MISC.MEM.SIZE.BAD** **(general)** | Leak Tainted allocation size Unreasonable size argument Library models account for allocator failure cases |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **26 S, 140 S, 6 D, 28 D, 5 C, 1 U** | Partially implemented |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **413, 613** | Assistance provided: reports use of null pointers including those which could be returned when a call to an allocation function fails |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023b | [CERT C: Rec. MEM11-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.mem11c.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] SEI Ref MSC11-C | Use Assertions for Diagnostic Testing.  Always use assert() macro to log and ensure that data is valid. It will return information about failures during testing and debug process. |

| **Noncompliant Code** |
| --- |
| This assertion is used in the right place (before accessing data), but is made on memory that could be potentially exhausted, so it could be useless. This is not a proper use for an assertion – instead the memory must be prevented from exhausting. |
| #include <iostream> #include <string> using namespace std;  string str;  cin << str; assert(str);  cout >> str; |

| **Compliant Code** |
| --- |
| This assertion is made on data that has been limited. It can be used to detect if you are accessing data that exists. |
| #include <iostream> #include <string> using namespace std;  char[10] str;  cin.getline(str, 10); assert(str);  cout >> str; |

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

| **Principles(s):** 3 & 10 – Follow policies and secure coding standards that prevent issues later on. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| LOW | UNLIKELY | HIGH | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.0p0 | **LANG.FUNCS.ASSERTS** | Not enough assertions |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **ASSERT\_SIDE\_EFFECT** | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-MSC11-a** | Assert liberally to document internal assumptions and invariants |
| N/A | N/A | N/A | N/A |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] SEI Ref ERR05-C | Detect errors without dictating the handling.  Instead of guiding a program to abort or close, catch errors in a way that logs the results and continues operation of the program. |

| **Noncompliant Code** |
| --- |
| This code handles an error by calling abort instead of continuing program operation and running the rest of the function. |
| **void** g(**void**) {  /\* ... \*/  **if** (something\_really\_bad\_happens) {  **fprintf**(stderr, "Something really bad happened!\n");  **abort**();  }  /\* ... \*/  }    **void** f(**void**) {  g();  /\* ... Do the rest of f ... \*/  } |

| **Compliant Code** |
| --- |
| This code uses a value as an error code when a problem is found, and uses that to determine if the rest of the code should run. This value can also be used for logging and bug checking in other parts of the code. |
| **const** errno\_t ESOMETHINGREALLYBAD = 1;    errno\_t g(**void**) {  /\* ... \*/  **if** (something\_really\_bad\_happens) {  **return** ESOMETHINGREALLYBAD;  }  /\* ... \*/  **return** 0;  }    errno\_t f(**void**) {  errno\_t status = g();  **if** (status != 0) {  **return** status;  }    /\* ... Do the rest of f ... \*/    **return** 0;  } |

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

| **Principles(s):** 8 & 10 – Follow secure coding practices and log errors as part of defense-in-depth layers |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) | N/A | N/A | Could detect violations of this rule merely by reporting functions that call abort(), exit(), or \_Exit() inside an if or switch statement. This would also catch many false positives, as ROSE could not distinguish a library function from an application function |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-ERR05-a** **CERT\_C-ERR05-b** **CERT\_C-ERR05-c** | The 'abort()' function from the 'stdlib.h' or 'cstdlib' library shall not be used The 'exit()' function from the 'stdlib.h' or 'cstdlib' library shall not be used The 'quick\_exit()' and '\_Exit()' functions from the 'stdlib.h' or 'cstdlib' library shall not be u |
| N/A | N/A | N/A | N/A |
| N/A | N/A | N/A | N/A |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers | [STD-008-CPP] INT30-C | Ensure that unsigned integer values do not wrap.  Overflow and underflow with unsigned integers can result in unexpected behavior that is difficult to trace. Use conditionals to prevent this. |

| **Noncompliant Code** |
| --- |
| This code performs addition on two unsigned integers without checking for overflow. The sum will not be correct. |
| **void** func(unsigned **int** ui\_a, unsigned **int** ui\_b) {  unsigned **int** usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This code uses a pre-condition check to make sure that the unsigned value will not overflow. |
| #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;  }  /\* ... \*/  } |

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

| **Principles(s):** 1 & 10 – Validate all input and follow secure coding standards. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | LIKELY | HIGH | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **integer-overflow** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | **CertC-INT30** | Implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.0p0 | **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](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **INTEGER\_OVERFLOW** | Implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Floats | [STD-009-CPP] SEI Ref FLP-30C | Never use floats as counters.  Floats are not precise whole numbers and can behave unpredictably when used as counters. Always use integers for iteration and counting. |

| **Noncompliant Code** |
| --- |
| Float used as a counter in a for loop. This may not iterate the number of times expected. |
| **void** func(**void**) {  **for** (**float** x = 0.1f; x <= 1.0f; x += 0.1f) {  /\* Loop may iterate 9 or 10 times \*/  }  } |

| **Compliant Code** |
| --- |
| This loop uses an integer for a counter to ensure that it always iterates exactly 10 times. |
| #include <stddef.h>    **void** func(**void**) {  **for** (**size\_t** count = 1; count <= 10; ++count) {  **float** x = count / 10.0f;  /\* Loop iterates exactly 10 times \*/  }  } |

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

| **Principles(s):** 10 – Use secure coding standards to prevent issues later on. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| LOW | PROBABLE | LOW | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **for-loop-float** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.0p0 | **LANG.STRUCT.LOOP.FPC** | Float-typed loop counter |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **MISRA C 2004 Rule 13.4**  **MISRA C 2012 Rule 14.1** | Implemented |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 23.04 | **for-loop-float** | Fully checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Identifiers | [STD-010-CPP] SEI Ref DCL51-CPP | Do not use reserved identifiers.  Avoid using any reserved identifier by the C++ libraries or files. Anything starting with a dunder or single underscore followed by a letter could be reserved. Do not ignore compiler warnings about reserved keywords. |

| **Noncompliant Code** |
| --- |
| This filename uses a reserved identifier syntax and is not safe. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| This corrected code uses a safer syntax that begins with an alphanumeric character. |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    // Contents of <my\_header.h>    #endif // MY\_HEADER\_H |

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

| **Principles(s):** 3 & 10 –Create & follow secure coding standards to avoid issues later. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| LOW | UNLIKELY | LOW | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **reserved-identifier** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.0p0 | **LANG.ID.NU.MK**  **LANG.STRUCT.DECL.RESERVED** | Macro name is C keyword  Declaration of reserved name |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **86 S, 218 S, 219 S, 580 S** | Fully implemented |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **reserved-identifier** | Partially checked |

### 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.

Automated testing shall be used while coding in the form of unit tests during the build phase of Pre-Production. These tests will run automatically during the build and verify/test phase to ensure every part of the program is working properly. They should be made for every function of the program, including the construction of classes. Assertions should be included in the code during this phase for all variables, during all functions.

Automated tools such as Astree or CppCheck should be run on code during the Verify and Test step to catch problems that are not apparent during coding or caught by the IDE.

All functions should use error handling and logging, added in during the build phase. These will help track down bugs and problems in the ‘monitor and detect’ phase of Production so that they can be repaired and responded to.

### 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 | Medium | High | 2 |
| STD-002-CPP | High | Probable | Medium | High | 1 |
| STD-003-CPP | High | High | Medium | High | 1 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | Low | Probable | High | Low | 3 |
| STD-006-CPP | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Medium | Probable | High | Low | 3 |
| STD-008-CPP | High | Likely | High | Medium | 2 |
| STD-009-CPP | Low | Probable | Low | Medium | 2 |
| STD-010-CPP | Low | Unlikely | Low | Low | 3 |

### 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 | Data should be encrypted during storage. Passwords in the database should not be human-readable, nor should sensitive data such as credit card and PIN numbers or full addresses and phone numbers.  This prevents harm to customers if data is leaked. |
| Encryption at flight | Data should never be sent unencrypted. All data travelling between devices should be fully encrypted, as this can be intercepted by attackers and revealed and is a common target. This is when data is most vulnerable. |
| Encryption in use | While in use, only decrypt when necessary for functioning of the application, such as when a user wishes to see and reset their password after proving their identity, and when it needs to be verified by the application using the decryption key. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Use unique usernames and encrypted passwords up to current legal standards consisting of at least 8 characters and containing only alphanumeric characters and !@#$%^&\*  Require this authentication for access to all areas of the application.  Limits unauthorized access by confirming the identity of anyone using the system. Standards make passwords difficult to guess through brute force and unreadable in case of an information leak from a database. Limiting characters prevents code injections. |
| Authorization | Create user and administrative roles and grant them access only to specific parts of the application that are of interest. Verify the role and permissions of the user whenever data is in transit or functions are performed.  This prevents unauthorized access and follows the Principle of Least Privilege to limit attacks and leaks of data. |
| Accounting | Log all the activity of users and keep this in a database or somewhere accessible only to the highest-privilege users. Changes to the database, functions performed, and sending data should all be tracked with the user credentials and ISP.  This helps track down an attack when it happens and verify activity. |

**\***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.

*The existence of these policies alone is part of principle 3: Create a set of standards for the development team to follow for consistency and prevention of issues and vulnerabilities.*

**STD-001-CPP:** Policies 1 & 10 – Follow a security standard to validate all input and not trust it by default.

**STD-002-CPP:** 2 & 3: Follow good coding standards, heed compiler warnings about uninitialized memory.

**STD-003-CPP:** 10: Follow secure coding standards to prevent future issues.

**STD-004-CPP:** 1 & 7: Sanitize data sent to other systems. Validate all input.

**STD-005-CPP:** 10 – Follow policies and secure coding standards that prevent later issues.

**STD-006-CPP:** 10 – Follow policies and secure coding standards that prevent issues later on.

**STD-007-CPP:** 10 – Follow secure coding practices and log errors as part of defense-in-depth layers

**STD-008-CPP:** 1 & 10 – Validate all input and follow secure coding standards.

**STD-009-CPP:** 10 – Use secure coding standards to prevent issues later on.

**STD-010-CPP:** 10 – Follow secure coding standards to avoid issues later.

**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 | 1/24/2024 | Standards and Principles | Vincent Snow |  |
| 1.2 | 2/13/2024 | Revisions, Explanations, Tables | Vincent Snow |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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