**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 is the proper testing of any input supplied by another user or application. This input must be verified before use because it could have been created by a malicious user. If the new input is malicious, it could open your program up to injection attacks or other security breaches. |
| 1. Heed Compiler Warnings | Compiler warnings are intended to warn designers of bugs in the code. These warnings are important because some bugs cannot be picked up by testing. Warnings should be addressed, and the code corrected. Ignoring the warning or decreasing the warning level will lead to buggy, non-secure code. |
| 1. Architect and Design for Security Policies | Software architecture involves designing your software in such a way that security policies are implemented and enforced. This design may include requiring credentials to access certain documents or applications, or only allowing IT admin to install software. |
| 1. Keep It Simple | Ideally, the design for software is as small and as simple as possible. The more complex the design is, the more likely it is that errors were made. Also, the more complex a program is, the harder it is to properly test and examine for bugs. |
| 1. Default Deny | Default deny means that access to a system or program is based on permission. Thus, by default, users are denied access. Certain conditions must be met in order to use the system or program. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege refers to the concept of giving users the minimum amount of access to a system or program that they need to do their jobs. You can also limit the amount of time a user has to access a system. These restrictions reduce an attacker’s chance to gain access to important applications or data. |
| 1. Sanitize Data Sent to Other Systems | Data sent to complex systems (command processors, external programs, relational databases, etc.) should be cleansed of any special characters that may trigger actions or commands. If this event occurs, it may result in a software vulnerability that hackers can exploit. |
| 1. Practice Defense in Depth | Defense in depth is a way to manage risk by layering multiple defensive strategies together. If one layer of security fails, another is there to keep the line of defense going. Without defense in depth, a system has a single point of failure, and the risk of a security breach is high. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques are meant to identify and eliminate security vulnerabilities. A company may try fuzz testing on its systems, which is the process of feeding invalid or unexpected inputs into a program to find coding errors or security loopholes. They may also implement penetration testing, which is a simulated attack against your systems to expose vulnerabilities. |
| 1. Adopt a Secure Coding Standard | A secure coding standard should be established and applied for your chosen programming language and the platform in which you create your code. This ensures that all developers are on the same page and following the same guidelines, eliminating the chance that someone is using non-secure coding practices. |

### 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** | **Use correct integer precisions** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | The precision of an integer type cannot be determined by its size. Integer precision is important in code. The sizeof operator will return the number of bytes used by an object, but the precision will not be accurate. |

| **Noncompliant Code** |
| --- |
| This code example shows a function that produces 2 raised to the power of the function argument. This code will result in undefined behavior. An expression changed by a negative number of bits, or by <= the number of bits that exist in the operand, will ensure that the argument is less than the number of bits used to store the value of the unsigned int. |
| unsigned int pow2(unsigned int exp) {  if (exp >= sizeof(unsigned int) \* CHAR\_BIT) {  /\* Handle error \*/  }  return 1 << exp;  } |

| **Compliant Code** |
| --- |
| Below is the same function rewritten to be compliant. It uses a popcount function, which counts the number of bits set on any unsigned integer. This allows the code to determine the precision of any integer type, signed or unsigned. |
| size\_t popcount(uintmax\_t num) {  size\_t precision = 0;  while (num != 0) {  if (num % 2 == 1) {  precision++;  }  num >>= 1;  }  return precision;  } |

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

| **Principle 9:** Use Effective Quality Assurance Techniques. Quality assurance testing will find errors showing that integers have lost precision and are therefore displaying inaccurate results. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 |  | Supported: Astrée reports overflows due to insufficient precision. |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2021.2 | C0582  C++3115 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.1 | CERT\_C-INT35-a | Use correct integer precisions when checking the right-hand operand of the shift operator |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | CERT C: Rule INT35-C | Checks for situations when integer precisions are exceeded (rule fully covered) |
| [PRQA QA-C](https://wiki.sei.cmu.edu/confluence/display/c/PRQA+QA-C) | 9.7 | 0582 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that unsigned integer operations do not wrap** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Unsigned integer wrapping occurs when a number is too large to fit the 1-byte range. The number trying to be stored is divided by 256 and the remainder is stored instead of the correct number. |

| **Noncompliant Code** |
| --- |
| The addition of these unsigned integers may result in wrapping. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This code tests first that the addition will not result in wrapping before it allows the program to move forwards. |
| 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.**

| **Principle 10:** Adopt a Secure Coding Standard. All developers on the team should follow the same coding policies. This policy may be to avoid unsigned integers since they are insecure and use signed integers instead. |
| --- |

**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) | 20.10 | integer-overflow | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | CertC-INT30 | Implemented |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | INTEGER\_OVERFLOW | Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.1 | CERT\_C-INT30-a  CERT\_C-INT30-b  CERT\_C-INT30-c | Avoid integer overflows  Integer overflow or underflow in constant expression in '+', '-', '\*' operator  Integer overflow or underflow in constant expression in '<<' operator |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do not attempt to create a std::string from a null pointer** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | CPP has policy-based implementation details for the basic string type. One of these details is the character length function, which can be used to determine the number of characters in a null-terminated string. If a null pointer is passed to this function, it will dereference the null pointer and result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| The below noncompliant code example shows that a string object is created by calling to std::getenv. Because the variable does not exist, std::getenv returns a null pointer. This will lead the code to act in unexpected ways. |
| void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| We can make the above code compliant by telling the computer to check for null pointers before it creates the string object. |
| 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.**

| **Principle 2:** Heed Compiler Warnings. The compiler should recognize when a pointer has been dereferenced and produce a warning. Developers should not ignore this warning. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | assert\_failure |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | C++4770, C++4771, C++4772, C++4773, C++4774 |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2021.1 | 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 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Exclude user input from format strings** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Do not allow input functions with strings that may contain a changed value. If a hacker can control the input put into a string, they can cause system crashes, view sensitive data, or inject other harmful commands into the system. |

| **Noncompliant Code** |
| --- |
| The code below is meant for authentication, but it will accept the name of the user from the input string. The data cannot be trusted. The format-string vulnerability comes about because msg includes the untrusted user input, which is passed as the format-string argument in the call to fprintf. |
| void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {  /\* Handle error \*/  } else if (ret >= len) {  /\* Handle truncated output \*/  }  fprintf(stderr, msg); |

| **Compliant Code** |
| --- |
| Here is a safer way to do the same code. Instead of using fprintf, use fputs, which outputs msg directly to stderr without evaluating its contents. |
| void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {  /\* Handle error \*/  } else if (ret >= len) {  /\* Handle truncated output \*/  }  fputs(msg, stderr); |

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

| **Principle 1:** Validate Input Data. Any data being input from an outside user must be tested before it is accepted. Untested data may result in injection attacks or other security breaches. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.1p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | TAINTED\_STRING | Implemented |
| [GCC](https://wiki.sei.cmu.edu/confluence/display/c/GCC) | 4.3.5 |  | Can detect violations of this rule when the -Wformat-security flag is used |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.1 | 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** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | A pointer that referencing memory that has been deallocated becomes a dangling pointer. A hacker can access a dangling pointer and exploit its vulnerabilities. |

| **Noncompliant Code** |
| --- |
| This code shows that p is freed before p->next is executed, so p->next is reading memory that has already been freed. |
| struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  for (struct node \*p = head; p != NULL; p = p->next) {  free(p);  } |

| **Compliant Code** |
| --- |
| This error can be quickly corrected by storing a reference to p->next in q before p is freed. |
| struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  struct node \*q;  for (struct node \*p = head; p != NULL; p = q) {  q = p->next;  free(p);  } |

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

| **Principle 9:** Use Effective Quality Assurance Techniques. A quality assurance test can help identify and eliminate security vulnerabilities like this one. In a case like this, they may find it through penetration testing, which is a simulated attack against your systems to expose vulnerabilities like a dangling pointer. |
| --- |

**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) | 20.10 | dangling\_pointer\_use | Supported  Astrée reports all accesses to freed allocated memory. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-MEM30 | Detects memory accesses after its deallocation and double memory deallocations |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.1p0 | ALLOC.UAF | Use after free |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Assertion Handling** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | An assertion is a statement that checks if a condition is true. If the condition is true, the program will proceed normally. If it is false, the program will terminate, and an error message will be displayed. |

| **Noncompliant Code** |
| --- |
| Assertions are disabled when a program is released for general use. The below code attempts to disable assert by commenting out the NDEBUG macro; however, this actually causes assert to keep working. This program will end with an error. |
| //#define NDEBUG  int main()  {  assert(1+1==3-1);  cout << "Expression valid, program continues.\n";  assert(3+2==5+1);  cout << "Assert is disabled, so execution continues with invalid expression\n";  } |

| **Compliant Code** |
| --- |
| This code shows that the NDEBUG macro needs to stay active to disable assert. |
| #define NDEBUG  int main()  {  assert(1+1==3-1);  cout << "Expression valid, program continues.\n";  assert(3+2==5+1);  cout << "Assert is disabled, so execution continues with invalid expression\n";  } |

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

| **Principle 2:** Heed Compiler Warnings. Compilers and debuggers can catch assertion errors. Developers should keep warnings turned on and heed the warnings when thrown. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Microsoft Visual Studio](https://visualstudio.microsoft.com/vs/features/cplusplus/) | 2019 | Debug assertion failed | Debugger will break at assertion point. |
| Google C++ Testing Framework | 1.11.0 | Assertion failed | Test failed -error message. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Detect and handle standard library errors** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Programmers cannot assume that all calls to standard library functions will work at all times. If errors are not detected, the system will experience undefined behavior. Errors need to be detected and handled according to the guidance provided by the error-handling policy. |

| **Noncompliant Code** |
| --- |
| This code has a failure that has not been checked for and will likely result in unexpected behavior. Setlocale will return a null pointe if locale is not installed. Additionally, the call to mbstowcs may experience failure, or it may store an unexpected sequence of characters in wcs. |
| int utf8\_to\_wcs(wchar\_t \*wcs, size\_t n, const char \*utf8,  size\_t \*size) {  if (NULL == size) {  return -1;  }  setlocale(LC\_CTYPE, "en\_US.UTF-8");  \*size = mbstowcs(wcs, utf8, n);  return 0;  } |

| **Compliant Code** |
| --- |
| To be compliant, the code should check the value returned by setlocale, and avoid calling to mbstowcs if the function fails. If the function fails, locale is restored to its initial setting. |
| int utf8\_to\_wcs(wchar\_t \*wcs, size\_t n, const char \*utf8,  size\_t \*size) {  if (NULL == size) {  return -1;  }  const char \*save = setlocale(LC\_CTYPE, "en\_US.UTF-8");  if (NULL == save) {  return -1;  }    \*size = mbstowcs(wcs, utf8, n);  if (NULL == setlocale(LC\_CTYPE, save)) {  return -1;  } |

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

| **Principle 10:** Adopt a Secure Coding Standard. All developers should work under the same set of policies and procedures. Errors need to be detected and handled according to the guidance provided by the error-handling policy. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this recommendation when checking for violations of EXP12-C. Do not ignore values returned by functions and EXP34-C. Do not dereference null pointers |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | MISRA C 2012 Rule 22.8  MISRA C 2012 Rule 22.9  MISRA C 2012 Rule 22.10 | Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.1 | CERT\_C-ERR33-a  CERT\_C-ERR33-b  CERT\_C-ERR33-c  CERT\_C-ERR33-d | \*The value returned by a function having non-void return type shall be used  \*The value returned by a function having non-void return type shall be used  \*Avoid null pointer dereferencing  \*Always check the returned value of non-void function |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | pointer arithmetic | Exhaustively verified. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not use floating-point variables as loop counters** |
| --- | --- | --- |
| Floating-Point | STD-008-CPP | Floating-point cannot be trusted to represent any fraction with perfect preciseness. They have the same representational limitations that integers do. They have a fixed number of precision bits, so the number of significant digits that they can contain is limited. |

| **Noncompliant Code** |
| --- |
| This code is attempting to use a floating-point variable as a loop counter. 0.1 is a repeating fraction in binary, and it cannot be precisely represented here. This loop is going to iterate an unexpected number of times. |
| void func(void) {  for (float x = 0.1f; x <= 1.0f; x += 0.1f) {  /\* Loop may iterate 9 or 10 times \*/  } |

| **Compliant Code** |
| --- |
| The loop counter should use an integer and derive the floating-point value from that. |
| 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.**

| **Principle 9:** Use Effective Quality Assurance Techniques. Quality assurance testing will find errors showing that floating point variables inside loops have lost precision and are therefore displaying inaccurate results. |
| --- |

**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) | 20.10 | for-loop-float | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-FLP30 | Fully implemented |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | cert-flp30-c | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.1p0 | LANG.STRUCT.LOOP.FPC | Float-typed loop counter |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do not subtract or compare two pointers that do not refer to the same array** |
| --- | --- | --- |
| Arrays | STD-009-CPP | To subtract two pointers, they both must point to elements in the same array or one past the last element of the array. If this is not done, it will result in unexpected behavior. Comparing pointers must be done in the same way, or it will also result in unexpected behavior. |

| **Noncompliant Code** |
| --- |
| This example is attempting to use pointer subtraction to determine how many free elements are left in the nums array. It is assuming that the nums array is adjacent to the end variable in memory, but the compiler may have put padding between them or even reordered them altogether. |
| enum { SIZE = 32 };    void func(void) {  int nums[SIZE];  int end;  int \*next\_num\_ptr = nums;  size\_t free\_elements;    /\* Increment next\_num\_ptr as array fills \*/    free\_elements = &end - next\_num\_ptr;  } |

| **Compliant Code** |
| --- |
| In this example, the number of free elements is computed by subtracting next\_num\_ptr from the address of the pointer past the nums array. The pointer address is stated instead of being assumed. |
| enum { SIZE = 32 };    void func(void) {  int nums[SIZE];  int \*next\_num\_ptr = nums;  size\_t free\_elements;    /\* Increment next\_num\_ptr as array fills \*/    free\_elements = &(nums[SIZE]) - next\_num\_ptr;  } |

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

| **Principles 4:** Keep It Simple. The design for software should be as small and simple as possible. The more complex the design is, like when pointers from different arrays are attempting to be compared, the more likely it is that errors were made. With this level of complexity, it may hard to detect bugs. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-ARR36 | Can detect operations on pointers that are unrelated |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | MISRA C 2004 17.2  MISRA C 2004 17.3  MISRA C 2012 18.2  MISRA C 2012 18.3 | Implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 437 S, 438 S | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.1 | CERT\_C-ARR36-a  CERT\_C-ARR36-b | \*Do not subtract two pointers that do not address elements of the same array  \*Do not compare two unrelated pointers |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Ensure that control never reaches the end of a non-void function** |
| --- | --- | --- |
| Misc. | STD-010-CPP | If control reaches the closing curly brace of a non-void function without evaluating a return statement, using the return value of the function call will result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| Here we see that control reaches the end of the checkpass function when the two strings passed to strcmp are not equal. This results in undefined behavior. Compilers may generate code for the checkpass function, returning various values while executing where no return statement is defined. |
| int checkpass(const char \*password) {  if (strcmp(password, "pass") == 0) {  return 1;  }  }    void func(const char \*userinput) {  if (checkpass(userinput)) {  printf("Success\n");  } |

| **Compliant Code** |
| --- |
| The fix for this is to ensure that checkpass always returns a value, as shown below. |
| int checkpass(const char \*password) {  if (strcmp(password, "pass") == 0) {  return 1;  }  return 0;  }    void func(const char \*userinput) {  if (checkpass(userinput)) {  printf("Success!\n");  } |

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

| **Principle 10:** Adopt a Secure Coding Standard. All developers should be following the same secure coding standard. This ensures that they are all on the same page and following the same guidelines, eliminating the chance that someone is using non-secure coding practices. In the above code, the best practice is to ensure checkpass returns a value. All developers on the team must follow this rule. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | return-implicit | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.1p0 | LANG.STRUCT.MRS | Missing return statement |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | MISSING\_RETURN | Implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 2 D, 36 S, 66 S | Fully implemented |

### 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.*

Every stage of development and operation should include a plan for security. There are five primary stages to consider: planning/development, code creation and commitment, building and testing, production, and operation. In the DevSecOps design above, planning and development encompasses the “Assess and plan” and “Design” sections. Code creation and commitment and building and testing are covered in “Build” and “Verify and test”. Production is handled in “Transition and health check”. Operation covers “Monitor and detect”, “Respond”, and “Maintain and stabilize”. Planning and development should include threat modelling and the use of the IDE security plugins. In the code creation and commitment phase, you can perform static security testing and security unit testing. For building and testing, you can move to dynamic security testing and security acceptance testing. Once the project goes to production, you can perform security smoke tests and penetration testing. Lastly, once the project is operating, there should be continuous monitoring, threat intelligence should be kept up to date, and further penetration testing can be applied. To make this process more comprehensive and efficient, the automated tools listed in the coding standards tables can be used for testing for security vulnerabilities. The tools provided in this document are not an exhaustive list. There are many other security tools available that are useful in doing automated security checks.

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

### 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 in rest (data not being used) is stored in a secure location and should be protected by a firewall. As an additional layer of security, it can be encrypted. This is called encryption in rest. |
| Encryption at flight | Data in transit is considered at flight. Encryption at flight will protect this data while in transit. Methods of encryption at flight include third-party certification, always-ON encryption, independent keys, and fast key rotation intervals. |
| Encryption in use | Data in use uses encryption in use. The XOR Encryption Algorithm will encrypt the data you are using by applying a mask, and it create a cipher text. Reapplying the same mask will un-encrypt the data and revert the file back to plain text. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication refers to the process of identifying the user attempting to access the system. To gain access, a user must provide their username and password, and complete multifactor authentication (providing the security code texted to their phone or emailed to them). |
| Authorization | Authorization determines what a user can access once they have entered the system. The principle of least privilege states that a user should have access only to what they need to perform their duties, and no more than that. A user’s level of access determines what files they can perform CRUD operations on, what drives they can access, and what programs they can open. |
| Accounting | Accounting is the act of monitoring and logging what a user does inside the system. The system must track what files the user opens, when they enter programs and how long they use it, and any changes they make, such as updating or deleting files. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 10/10/2021 | Completed Policy | Abby Farnsworth |  |
| 3.0 |  |  |  |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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