**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 | When taking input from users and other untrusted sources there needs to be check and balances to ensure the data is secure and safe to use. If we don’t check the input, we may be setting ourselves up for a data breach and attack. |
| 1. Heed Compiler Warnings | Use the highest warning level in the compiler and eliminate the warnings by compiling the code. These give us warnings at a base level that we can build off. By doing this we can ensure that the ability of an intrusion form someone with malintent is minimized. |
| 1. Architect and Design for Security Policies | If our system has different levels of clearance needed at different times, we should make sure to have several layers of defense and not keep all the data in the same location. That way if there is a breach, we are only exposing little pieces of the data at once and not all of it. |
| 1. Keep It Simple | Keeping the code and architecture simple is going to maximize security by making code understandable to current and future employees. It will also ensure that when we go to update and make sure we are secure that we can easily see what we did later. |
| 1. Default Deny | By default, we need to deny everyone from having access to certain data and areas of our enterprise. What this means is that no one has access until they are given access. We do this to ensure that we can see who is getting access at all times, to ensure security needs are being met. |
| 1. Adhere to the Principle of Least Privilege | Each task should only sue the minimal number of privileges to complete a task. We need to also minimize the time that we are accessing higher privileged data. This will limit the number of opportunities a hacker has to breach our system. |
| 1. Sanitize Data Sent to Other Systems | We need to ensure that we Sanitize the data that is being sent to other systems. This means that we are assuring that the data being sent is validated to ensure that it is code we want the system to take in and not malicious users trying to break in. |
| 1. Practice Defense in Depth | We need to ensure that we are using things like secure coding, firewalls and using safe runtime environments to ensure that we are staying as protected as possible. Several layers of defense will help us ensure that we are keeping ourselves protected against malicious users. |
| 1. Use Effective Quality Assurance Techniques | We should be testing the software ourselves as well as having external reviews to ensure we get a non-biased look at our code. We can use techniques like fuzz testing, penetration testing and source code audits. |
| 1. Adopt a Secure Coding Standard | We need to ensure that we are on the same page as far as the standards we are using to ensure secure code. Ensuring that we pick the proper languages and platforms for the job at hand. |

### 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** | **Ensure that operations on signed integers do not result in overflow** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Make sure that we use the correct operations and best practices to ensure that we don’t have overflow. |

| **Noncompliant Code** |
| --- |
| Integer overflow can happen during the addition here because of the signed operands si\_a and si\_b. |
| void func(signed int si\_a, signed int si\_b) {  signed int sum = si\_a + si\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| ensures that the addition operation cannot overflow, since we have an if else addressing the possibility of overflow. |
| #include <limits.h>    void f(signed int si\_a, signed int si\_b) {  signed int sum;  if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||  ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {  /\* Handle error \*/  } else {  sum = si\_a + si\_b;  }  /\* ... \*/  } |

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

| **Principles(s): Adopt a secure coding standard ties in because we need to choose the right languages for the job and understand how to use those languages safely.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | integer-overflow | Fully checked |
| CodeSonar | 6.2p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  MISC.MEM.SIZE.ADDOFLOW  MISC.MEM.SIZE.BAD  MISC.MEM.SIZE.MULOFLOW  MISC.MEM.SIZE.SUBUFLOW | Addition overflow of allocation size  Integer overflow of allocation size  Multiplication overflow of allocation size  Subtraction underflow of allocation size  Addition overflow of size  Unreasonable size argument  Multiplication overflow of size  Subtraction underflow of size |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| Helix QAC | 2022.1 | C2800, C2801, C2802, C2803, C2860, C2861, C2862, C2863  C++2800, C++2801, C++2802, C++2803, C++2860, C++2861, C++2862, C++2863 |  |
| Klocwork | 2022.1 | CWARN.NOEFFECT.OUTOFRANGE  NUM.OVERFLOW |  |
| LDRA tool suite | 9.7.1 | 493 S, 494 S | Partially implemented |
| Parasoft C/C++test | 2021.2 | CERT\_C-INT32-a  CERT\_C-INT32-b  CERT\_C-INT32-c | Avoid integer overflows  Integer overflow or underflow in constant expression in '+', '-', '\*' operator  Integer overflow or underflow in constant expression in '<<' operator |
| Parasoft Insure++ |  | CERT C: Rule INT32-C | Runtime analysis |
| Polyspace Bug Finder | R2021a | CERT C: Rule INT32-C | Checks for:  Integer overflow  Tainted division operand  Tainted modulo operand  Rule partially covered. |
| PRQA QA-C | 9.7 | 2800, 2801, 2802, 2803,  2860, 2861, 2862, 2863 | Fully implemented |
| PRQA QA-C++ | 4.4 | 2800, 2801, 2802, 2803,  2860, 2861, 2862, 2863 |  |
| PVS-Studio | 7.17 | V1026, V1070, V5010 |  |
| TrustInSoft Analyzer | 1.38 | signed\_overflow |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Use valid iterator ranges** |
| --- | --- | --- |
| **Data Value** | [STD-003-CPP] | When iterating over elements of a container the iterators need to iterate over the valid range. The range is a pair of or iterators that are based of the first and past-the-end elements. |

| **Noncompliant Code** |
| --- |
| Std::for-each() will compare the first iterator after comparing it, with c.end() before c.begin(). It will not iterate. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.end(), c.begin(), [](int i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| Since c.begin() is before c.end() it will iterate through the function properly. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.begin(), c.end(), [](int i) { std::cout << i; });  } |

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

| **Principles(s):** Validate input data and adopting secure coding standards tie to this because we always need to ensure that we are using valid iterator ranges. These both allow us to do so. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | overflow\_upon\_dereference |  |
| CodeSonar | 6.2p0 | LANG.MEM.BO | Buffer Overrun |
| Helix QAC | 2022.1 | C++3802 |  |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-CTR53-a  CERT\_CPP-CTR53-b | Do not use an iterator range that isn't really a range  Do not compare iterators from different containers |
| PRQA QA-C++ | 4.4 | 3802 |  |
| PVS-Studio | 7.17 | V539, V662, V789 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do not attempt to create a std::string from a null pointer** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Passing a null pointer to this function is undefined behavior because it would result in dereferencing a null pointer. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist (or some other error occurs). |
| void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| the results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| 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): Head compiler warnings and using effective quality assurance techniques should catch the issue listed here. The compiler should throw a warning your way that you need to pay attention to and fix. And by testing the code we can make sure it works right away.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Assert\_failure |  |
| Helix QAC | 2022.1 | C++4770, C++4771, C++4772, C++4773, C++4774 |  |
| Klocwork | 2022.1 | 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 | 2021.2 | [Insert text.] | Avoid null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Range check element access** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | If index operators are unchecked (no exceptions are thrown for range errors), as well as trying to modify the resulting out of range object results in undefined behavior. |

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

| **Compliant Code** |
| --- |
| The std::basic\_string::at() function, which behaves in a similar fashion to the index operator[] but throws a std::out\_of\_range exception if pos >= size(). |
| extern std::size\_t get\_index();    void f() {  std::string s("01234567");  try {  s.at(get\_index()) = '1';  } catch (std::out\_of\_range &) {  // Handle error  }  } |

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

| **Principles(s):** Keep it simple and head compiler warnings is important when looking at this coding standard. We need to ensure that we see warning messages and keep the code clean and simple so we understand what is going on when we look back on it later. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | assert\_failure |  |
| CodeSonar | 6.2p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Helix QAC | 2022.1 | C++3162, C++3163, C++3164, C++3165 |  |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-STR53-a | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2021b | CERT C++: STR53-CPP | Checks for:  Array access out of bounds  Array access with tainted index  Pointer dereference with tainted offset  Rule partially covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Allocate sufficient memory for an object** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CLG] | The types of integer expressions used as size arguments to malloc(), calloc(), realloc(), or aligned\_alloc() must have sufficient range to represent the size of the objects to be stored. If size arguments are incorrect or can be manipulated by an attacker, then a buffer overflow may occur. Incorrect size arguments, inadequate range checking, integer overflow, or truncation can result in the allocation of an inadequately sized buffer. |

| **Noncompliant Code** |
| --- |
| inadequate space is allocated for a struct tm object because the size of the pointer is being used to determine the size of the pointed-to object |
| struct tm \*make\_tm(int year, int mon, int day, int hour,  int min, int sec) {  struct tm \*tmb;  tmb = (struct tm \*)malloc(sizeof(tmb));  if (tmb == NULL) {  return NULL;  }  \*tmb = (struct tm) {  .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,  .tm\_mday = day, .tm\_mon = mon, .tm\_year = year  };  return tmb;  } |

| **Compliant Code** |
| --- |
| The correct amount of memory is allocated for the struct tm object. |
| struct tm \*make\_tm(int year, int mon, int day, int hour,  int min, int sec) {  struct tm \*tmb;  tmb = (struct tm \*)malloc(sizeof(\*tmb));  if (tmb == NULL) {  return NULL;  }  \*tmb = (struct tm) {  .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,  .tm\_mday = day, .tm\_mon = mon, .tm\_year = year  };  return tmb;  } |

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

| **Principles(s):** Validate input data and adopting a secure coding standard are crucial. We need to ensure that the data we are entering in and looking for fits the desired input and output. Secure coding standards allow us to make sure that we are using the proper syntax and languages to get a job done. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | malloc-size-insufficient | Partially checked  Besides direct rule violations, all undefined behaviour resulting from invalid memory accesses is reported by Astrée. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM35 |  |
| CodeSonar | 6.2p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  ALLOC.SIZE.TRUNC  IO.TAINT.SIZE  MISC.MEM.SIZE.BAD | Addition overflow of allocation size  Addition overflow of allocation size  Multiplication overflow of allocation size  Subtraction underflow of allocation size  Truncation of allocation size  Tainted allocation size  Unreasonable size argument |
| Compass/ROSE |  |  | Could check violations of this rule by examining the size expression to malloc() or memcpy() functions. Specifically, the size argument should be bounded by 0, SIZE\_MAX, and, unless it is a variable of type size\_t or rsize\_t, it should be bounds-checked before the malloc() call. If the argument is of the expression a\*b, then an appropriate check is  if (a < SIZE\_MAX / b && a > 0) ... |
| Coverity | 2017.07 | BAD\_ALLOC\_STRLEN  SIZECHECK (deprecated) | Partially implemented  Can find instances where string length is miscalculated (length calculated may be one less than intended) for memory allocation purposes. Coverity Prevent cannot discover all violations of this rule, so further verification is necessary  Finds memory allocations that are assigned to a pointer that reference objects larger than the allocated block |
| Helix QAC | 2022.1 | C0696, C0701, C1069, C1071, C1073, C2840, C2841, C2842, C2843, C2935, C2936, C2937, C2938  C++2840, C++2841, C++2842, C++2843, C++2935, C++2936, C++2937, C++2938 |  |
| Klocwork | 2022.1 | INCORRECT.ALLOC\_SIZE  SV.TAINTED.ALLOC\_SIZE |  |
| LDRA tool suite | 9.7.1 | 400 S, 487 S, 115 D | Enhanced enforcement |
| Splint | 3.1.1 |  |  |
| Parasoft C/C++test | 2021.2 | CERT\_C-MEM35-a | Do not use sizeof operator on pointer type to specify the size of the memory to be allocated via 'malloc', 'calloc' or 'realloc' function |
| PC-lint Plus | 433, 826 | 433, 826 | Partially supported |
| Polyspace Bug Finder | CERT C: Rule MEM35-C | CERT C: Rule MEM35-C | Checks for:  Pointer access out of bounds  Memory allocation with tainted size  Rule fully covered. |
| PRQA QA-C | 9.7 | 0696, 0701, 1069, 1071, 1073, 2840, 2841, 2842, 2843, 2935, 2936, 2937, 2938 |  |
| PRQA QA-C++ | 4.4 | 2840, 2841, 2842, 2843, 2935, 2936, 2937, 2938 |  |
| PVS-Studio | 7.17 | V531, V635, V781 |  |
| RuleChecker | 20.10 | malloc-size-insufficient | Partially checked |
| TrustInSoft Analyzer | 1.38 | mem\_access | Exhaustively detects undefined behavior (see one compliant and one non-compliant example). |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Incorporate diagnostic tests using assertions** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | Incorporate diagnostic tests into your program. Like the assert() macro. |

| **Noncompliant Code** |
| --- |
| Because memory availability depends on the overall state of the system and can become exhausted at any point during a process lifetime, a robust program must be prepared to gracefully handle and recover from its exhaustion. |
| char \*dupstring(const char \*c\_str) {  size\_t len;  char \*dup;    len = strlen(c\_str);  dup = (char \*)malloc(len + 1);  assert(NULL != dup);    memcpy(dup, c\_str, len + 1);  return dup;  } |

| **Compliant Code** |
| --- |
| This code shows how to detect and handle possible memory exhaustion: |
| char \*dupstring(const char \*c\_str) {  size\_t len;  char \*dup;    len = strlen(c\_str);  dup = (char\*)malloc(len + 1);  /\* Detect and handle memory allocation error \*/  if (NULL == dup) {  return NULL;  }    memcpy(dup, c\_str, len + 1);  return dup;  } |

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

| **Principles(s):** Using effective quality assurance techniques ties into incorporating diagnostic tests using assertions. We want to ensure that we are always testing out code so as we write more code we can see if we broke our code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| 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 | 2021.2 | CERT\_C-MSC11-a | Assert liberally to document internal assumptions and invariants |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Guarantee exception safety** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Proper handling of errors and exceptional situations is essential for the continued correct operation of software. The preferred mechanism for reporting errors in a C++ program is exceptions rather than error codes. Because exceptions introduce code paths into a program, it is important to consider the effects of code taking such paths and to avoid any undesirable effects that might arise otherwise. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows a flawed copy assignment operator. The implicit invariants of the class are that the array member is a valid (possibly null) pointer and that the nElems member stores the number of elements in the array pointed to by array. The function deallocates array and assigns the element counter, nElems, before allocating a new block of memory for the copy. |
| class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }      IntArray(const IntArray& that); // nontrivial copy constructor  IntArray& operator=(const IntArray &rhs) {  if (this != &rhs) {  delete[] array;  array = nullptr;  nElems = rhs.nElems;  if (nElems) {  array = new int[nElems];  std::memcpy(array, rhs.array, nElems \* sizeof(\*array));  }  }  return \*this;  }    // ...  }; |

| **Compliant Code** |
| --- |
| In this compliant solution, the copy assignment operator provides the strong exception safety guarantee. The function allocates new storage for the copy before changing the state of the object. Only after the allocation succeeds does the function proceed to change the state of the object. |
| class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }    IntArray(const IntArray& that); // nontrivial copy constructor    IntArray& operator=(const IntArray &rhs) {  int \*tmp = nullptr;  if (rhs.nElems) {  tmp = new int[rhs.nElems];  std::memcpy(tmp, rhs.array, rhs.nElems \* sizeof(\*array));  }  delete[] array;  array = tmp;  nElems = rhs.nElems;  return \*this;  }    // ...  }; |

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

| **Principles(s):** Practice in defense in depth as well as using quality assurance techniques is what programmers can use to ensure that we are catching bugs as we go. We do not want errors in C++ we want exceptions because they introduce code paths into the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2022.1 | C++4075, C++4076 |  |
| LDRA tool suite | 9.7.1 | 527 S, 56 D, 71 D | Partially implemented |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-ERR56-a  CERT\_CPP-ERR56-b | Always catch exceptions  Do not leave 'catch' blocks empty |
| PRQA QA-C++ | 4.4 | 4075, 4076 |  |
| PVS-Studio | 7.17 | V565, V1023, V5002 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not add or subtract an integer to a pointer to a non-array object** |
| --- | --- | --- |
| Arrays | [STD-008-CPP] | Pointer arithmetic must be performed only on pointers that reference elements of array objects. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to access structure members using pointer arithmetic. This practice is dangerous because structure members are not guaranteed to be contiguous. |
| struct numbers {  short num\_a, num\_b, num\_c;  };    int sum\_numbers(const struct numbers \*numb){  int total = 0;  const short \*numb\_ptr;    for (numb\_ptr = &numb->num\_a;  numb\_ptr <= &numb->num\_c;  numb\_ptr++) {  total += \*(numb\_ptr);  }    return total;  }    int main(void) {  struct numbers my\_numbers = { 1, 2, 3 };  sum\_numbers(&my\_numbers);  return 0;  } |

| **Compliant Code** |
| --- |
| A better solution is to define the structure to contain an array member to store the numbers in an array rather than a structure. |
| struct numbers {  short a[3];  };    int sum\_numbers(const short \*numb, size\_t dim) {  int total = 0;  for (size\_t i = 0; i < dim; ++i) {  total += numb[i];  }    return total;  }    int main(void) {  struct numbers my\_numbers = { .a[0]= 1, .a[1]= 2, .a[2]= 3};  sum\_numbers(  my\_numbers.a,  sizeof(my\_numbers.a)/sizeof(my\_numbers.a[0])  );  return 0;  } |

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

| **Principles(s):** Adopting a secure coding standard will tell us how to use pointers in the program and code we are working with. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 |  | Supported indirectly via MISRA C:2004 Rule 17.4. |
| Axivion Bauhaus Suite |  | CertC-ARR37 | Supported indirectly via MISRA C:2004 Rule 17.4. |
| Compass/ROSE |  |  | Fully implemented |
| Coverity |  | ARRAY\_VS\_SINGLETON | Implemented |
| Helix QAC |  | C2930, C2931, C2932, C2933  C++2930, C++2931, C++2932, C++2933, C++3705, C++3706, C++3707 |  |
| Klocwork |  | MISRA.PTR.ARITH.2012 |  |
| LDRA tool suite |  | 567 S | Partially implemented |
| Parasoft C/C++test |  | CERT\_C-ARR37-a | Pointer arithmetic shall not be applied to pointers that address variables of non-array type |
| PC-lint Plus |  | 2662 | Partially supported |
| Polyspace Bug Finder |  | CERT C: Rule ARR37-C | Checks for invalid assumptions about memory organization (rule partially covered) |
| PRQA QA-C |  | 2930, 2931, 2932, 2933 |  |
| PRQA QA-C++ |  | 2930, 2931, 2932, 2933,  3705, 3706, 3707 |  |
| RuleChecker | 20.10 |  | Supported indirectly via MISRA C:2004 Rule 17.4. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Close files when they are no longer needed** |
| --- | --- | --- |
| Input Output | [STD-009-CPP] | 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** |
| --- |
| The constructor for std::fstream calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. Consequently, the underlying std::basic\_filebuf<T> object maintained by the object is not properly closed. |
| void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  std::terminate();  } |

| **Compliant Code** |
| --- |
| std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed. |
| 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();  } |

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

| **Principles(s):** practice defense in depth, by closing files when you are done with them you limit the ability for hackers to get into your system and code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 |  | Supported, but no explicit checker |
| CodeSonar | 6.2p0 | ALLOC.LEAK | Leak |
| Compass/ROSE |  |  |  |
| Coverity | 2017.07 | RESOURCE\_LEAK (partial) | Partially implemented |
| Helix QAC | 2022.1 | C2701, C2702, C2703  C++2701, C++2702, C++2703 |  |
| Klocwork | 2022.1 | RH.LEAK |  |
| LDRA tool suite | 9.7.1 | 49 D | Partially implemented |
| Parasoft C/C++test | 2021.2 | CERT\_C-FIO42-a | Ensure resources are freed |
| PC-lint Plus | 1.4 | 429 | Partially supported |
| Polyspace Bug Finder | R2021a | CERT C: Rule FIO42-C | Checks for resource leak (rule partially covered) |
| PRQA QA-C | 9.7 | 2701, 2702, 2703 |  |
| PRQA QA-C++ | 4.4 | 2701, 2702, 2703 |  |
| SonarQube C/C++ Plugin | 3.11 | S2095 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Guarantee that library functions do not overflow** |
| --- | --- | --- |
| Containers | [STD-010-CPP] | Copying data into a container that is not large enough to hold that data results in a buffer overflow. To prevent such errors, data copied to the destination container must be restricted on the basis of the destination container's size, or preferably, the destination container must be guaranteed to be large enough to hold the data to be copied. |

| **Noncompliant Code** |
| --- |
| A vector of integers is copied from src to dest using std::copy(). Because std::copy() does nothing to expand the dest vector, the program will overflow the buffer on copying the first element. |
| void f(const std::vector<int> &src) {  std::vector<int> dest;  std::copy(src.begin(), src.end(), dest.begin());  // ...  } |

| **Compliant Code** |
| --- |
| The proper way to use std::copy() is to ensure the destination container can hold all the elements being copied to it. This compliant solution enlarges the capacity of the vector prior to the copy operation. |
| void f(const std::vector<int> &src) {  // Initialize dest with src.size() default-inserted elements  std::vector<int> dest(src.size());  std::copy(src.begin(), src.end(), dest.begin());  // ...  } |

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

| **Principles(s):** Validate input data and adopting secure coding standards will allow you to make sure that you do not overflow your values. Follow the rules laid out and follow the rules for the data being input and you should be able to eliminate most errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | invalid\_pointer\_dereference |  |
| CodeSonar | 6.2p0 | BADFUNC.BO.\*  LANG.MEM.BO | A collection of warning classes that report uses of library functions prone to internal buffer overflows.  Buffer Overrun |
| Helix QAC | 2022.1 | C++3526, C++3527, C++3528, C++3529, C++3530, C++3531, C++3532, C++3533, C++3534 |  |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-CTR52-a | Do not pass empty container iterators to std algorithms as destinations |

### 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.

There are a few places we can use automation to help us enforce the standards in the DevOps process. Verifying and testing will allow us to scan for vulnerabilities before they arise. When we write tests that automatically run when we write new code, we enable ourselves to see mistakes and prevent attacks in the future. Assess and planning will let us know how and where we may be vulnerable to attacks. By monitoring and detecting we can log and alert events of intrusion or attempted intrusion. It is important that we are constantly on the look out for how we can use automation to defend our system against hackers.

### 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 | Probable | High | P6 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-005-CPP | High | Probable | High | P6 | L2 |
| STD-006-CLG | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | High | Likely | High | P9 | L2 |
| STD-008-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | High | Likely | Medium | P18 | L1 |

### 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 | This is designed to prevent attackers from accessing data that is unencrypted by ensuring that data is encrypted when on disk. This is inactive data stored physically like in databases, Datawarehouse, spreadsheets, archives, etc. Encryption is used here to ensure the data is secure from hackers, the only way they will get the information is if they hack the encryption protecting the data which would be hard. |
| Encryption at flight | This is data that is being encrypted when it’s being transmitted over a network. Amazon web services offers something like this. They take the data you send encrypt it, then once data is delivered to the proper person, they decrypt it. |
| Encryption in use | Encryption in use is done by never leaving data unsecured no matter Lifecyle stage the data is in. You are able to do this by having a triple-A framework and make sure that only the data the person needs is available to them. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication identifies a user and ensures they are who they say they are. This is typically enforced with some type of userID and password combination. This is to ensure that the data is only being seen by the desired users. |
| Authorization | Authorization is when you enforce policies such as, determining the activities, resources, or services a user can use. This policy is to make sure that the company can control what you see as what you are able to do with the data you are seeing. |
| Accounting | Accounting records the resources users consume during access to different things such as a network or application. This is used so the administrator can look back and see what data was looked at and by who if a breach or anything malicious happened to the data. |

**\***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 | 03/25/2022 | Milestone Three- Coding Standards | Michael Gontarek |  |
| [Insert text.] | 04/11/2022 | Project One: Security Policy | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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