**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

# By Joel Meza

# 

## 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 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 12](#_Toc52464063)

[Coding Standard 6 14](#_Toc52464064)

[Coding Standard 7 16](#_Toc52464065)

[Coding Standard 8 18](#_Toc52464066)

[Coding Standard 9 20](#_Toc52464067)

[Coding Standard 10 22](#_Toc52464068)

[Defense-in-Depth Illustration 24](#_Toc52464069)

[Project One 24](#_Toc52464070)

[1. Revise the C/C++ Standards 24](#_Toc52464071)

[2. Risk Assessment 24](#_Toc52464072)

[3. Automated Detection 24](#_Toc52464073)

[4. Automation 24](#_Toc52464074)

[5. Summary of Risk Assessments 25](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 25](#_Toc52464076)

[7. Map the Principles 26](#_Toc52464077)

[Audit Controls and Management 27](#_Toc52464078)

[Enforcement 27](#_Toc52464079)

[Exceptions Process 27](#_Toc52464080)

[Distribution 28](#_Toc52464081)

[Policy Change Control 28](#_Toc52464082)

[Policy Version History 28](#_Toc52464083)

[Appendix A Lookups 28](#_Toc52464084)

[Approved C/C++ Language Acronyms 28](#_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 | Data that is inserted into the program that is verified and validated. |
| 1. Heed Compiler Warnings | Set of instructions with warnings messages called error exceptions where their meant to prevent when code is compiled to help a developer see a possible exploit attack within the script of code. |
| 1. Architect and Design for Security Policies | Build a development and operation within an architecture design including applied security policies into a program. |
| 1. Keep It Simple | Short and sweet is a motto for keeping it simple when executing new code for a program to eliminate any future complex designed code that may lead into deeper errors to be compiled and be harder to debug later. |
| 1. Default Deny | Create data accessible for a user to enable access into the data released with security permission. Otherwise, deny by default to any user unless they provide proof of key or values required to grant access to the data. |
| 1. Adhere to the Principle of Least Privilege | Any project process will need the least privileges required to finish the task. Otherwise, any other elevated permission will only be accessible for a minimum amount of time as necessary to finish the privilege executable task. |
| 1. Sanitize Data Sent to Other Systems | Data that is sanitized and passed through complex subsystems; for example, relational databases SQL, terminal command prompts, etc. |
| 1. Practice Defense in Depth | Diverse layers of defense that are implemented are used for the program’s safety. Multiple layers of defense enable a program to stay secured and shielded by different layers of defense which helps in example to one layer getting compromised, then the other layers will still safeguard the program. |
| 1. Use Effective Quality Assurance Techniques | Considering effective quality assurance techniques are required for verifying and validating any vulnerability that can cause bugs to be malicious to the source code. Determining what techniques to use could be security testing, performance testing, and use case testing by penetration. |
| 1. Adopt a Secure Coding Standard | Encourage a secure coding standard by determining what programming language and software project infrastructure will be used for the security and performance of project/program. |

### 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] | Don’t emit any isolated enumerated value. |

| **Noncompliant Code** |
| --- |
| Execute code whether a given value is within scale of approved enumerated values. Although, by emitting to enumeration kind, whether would not be applicable to be signified by EnumList as one, two, and three. Therefore, if a value is external from the scale passed by f(), then the emitted EnumList outcome will not have a determined value and by using the value undetermined within the if conditional statement, its outcome will be undetermined behavior. |
| enum EnumList { // Data Type  first,  second,  third  };  Void f(int intVar) {  EnumList enumVar = static\_cast<EnumList>(intVar);    if (enumVar < first || enumVar > third) {  // Solution to Error  }  } |

| **Compliant Code** |
| --- |
| Verifies the value given to be signified by the enumerated data type. This is done previously until executing the conversion to validate that its outcome value is not left undetermined. |
| enum EnumList { // Data Type  first,  second,  third  };  Void f(int intVar) {  if (intVar < first || intVar > third) {  // Solution to Error  }  EnumList enumVar = static\_cast<EnumList>(intVar);  } |

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

| **Principles(s):** [Authenticate Data Input – Throwing to a not specified outcome doesn’t enable the data input to authenticate or invalidate access to the program/system. ] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Medium] | [Not Likeable] | [Medium] | [P4] | [L3] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [PVS Studio] | [7.15] | [V1016] |  |
| [Parasoft C/C++ Test] | [2021.2] | [Cert Cpp-INT50-A] | [Enum is an expression underlined type that only has values agreed to the enumerators.] |
| [PRQA QA-C++] | [4.4] | [3013] |  |
| [Axivion Bauhaus Suite] | [7.2.0] | [CertC++INT50] |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Using validity pointer, iterators, and references to components in containers. |

| **Noncompliant Code** |
| --- |
| The ‘pos’ is not validated after the first callback to place the following loop iterations that got undefined behavior. |
| #include <deque>  Void f(const double \*items, std::size\_t count) {  Std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  d.insert(pos, items[i] + 40.0);  }  } |

| **Compliant Code** |
| --- |
| The ‘pos’ is assigned to a validated iterator for each placement which will avoid not defined behavior. |
| #include  Void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  pos = d.insert(pos, items[i] + 40.0);  }  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [High] | [Likeable] | [High] | [P6] | [L2] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [PVS-Studi0] | [7.15] | [V783] |  |
| [Parasoft in C/C++ Test] | [2021.2] | [Cert CPP-CTR51-A] | [While being iterated, don’t change container settings] |
| [Astree] | [20.10] | [Prior Dereferenced Overflow] |  |
| [Helix QAC] | [2021.2] | [C++ 4746; C++ 4747,; C++ 4748; C++ 4749] |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Contains sufficient storage for the scale or range of data characters including a null determinator. |

| **Noncompliant Code** |
| --- |
| The input has absolute value where the following code can be directed to an exploit buffer overflow. |
| #include <iostream>  void f() {  Char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| Verify and validate that the data input is not reduced and the ‘for’ is protected among exploits buffer overflows which in this case, it’s used as std::string rather than a restricted array. |
| #include <iostream>  #include <string>  void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne, >> stringtTwo;  } |

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

| **Principles(s):** [Security Policy: Architecture and Design – Avoids any database architecture issues prior knowing if there is enough storage available.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [High] | [Likeable] | [Medium] | [P18] | [L1] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA Tool Suite] | [9.7.1] | [489 S; 66 X; 70 X; 71 X] | [Partial Implemented] |
| [SonarQube C/C++ Plugin] | [4.10] | [S3519] |  |
| [Code Sonar] | [6.1P-0] | [Lang.Mem.B0]  [Lang.Mem.T0] | [Buffer Overrun]  [Type Overrun]  [No Space for Null Terminator] |
| [Klocwork] | [2021.3] | [NNTS.MIGHT]  [NNTS.TAINTED] |  |
| [Parasoft C/C++ Test] | [2021.2] | [CERT-CPP-STR50-B; CERT-CPP-STR50-C; CERT-CPP-STR50-E; CERT-CPP-STR50-F; CERT-CPP-STR50-G] | [Prevent overflow prior reading a non-zero string] |
| [Helix QAC] | [2021.2] | [C++ 2835; C++ 2836; C++ 2839; C++ 5216] |  |
| [PolySpace Bug Finder] | [R2021B] | [CERT C++ - STR50-CPP] | 1. [Examine dangerous misbehavior standard functions.]  2. [Buffer overflow due to improper string specific formation.]  3. [Destination in buffer overflow by string manipulation.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Never initiate a value from an existing owned pointer that has an irrelevant smart pointer. |

| **Noncompliant Code** |
| --- |
| There are two irrelevant smart pointers which are built from the same constructor pointer value. If the variable ‘p2’ is deleted, then it removes the pointer value which is administrated. Therefore, when the variable ‘p1’ is deleted, then it removes the identical pointer value where the outcome becomes double the vulnerability. |
| #include <memory>  void f() {  int \*i = new int;  std::shared ptr<int> p1 (i);  std::shared ptr<int> p2 (i);  } |

| **Compliant Code** |
| --- |
| The constructor created two relevant objects for std::shared\_ptr to work with one another. So, the variable ‘p2’ is deleted, then the count for the shared\_pointer value is decreased. Therefore, if the variable ‘p1’ is deleted, then the count for the shared\_pointer is decrease towards zero, so its administrated pointer is removed. The compliance from std::make\_shared() was made alternatively by not allocating a raw\_ pointer and stowing its value into a constant variable. |
| #include <memory>  void f() {  std::shared\_ptr<int> p1 = std::make\_shared<int>();  std::shared\_ptr<int> p2(p1);  } |

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

| **Principles(s):** [Security Policy of Architecture and Design – Avoid errors from any buffer variable being modified and eliminated within the given code. This problem abandons ownership of the given pointer’s value.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [High] | [Likeable] | [Medium] | [P18] | [L1] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [PVS-Studio] | [7.15] | [V1006] |  |
| [Helix QAC] | [2021.2] | [C++ 4721; C++ 4722; C++ 4723] |  |
| [PolySpace Bug Finder] | [R2021B] | [CERT C++: MEM56-CPP] | [Checks the use of Pointer ownership] |
| [ParaSoft C/C++ Test] | [2021.2] | [CERT-CPP-MEM56-A] | [Never allocate an existing pointer value into an irrelevant smart pointer.] |
| [Astree] | [20.10] | [Use Pointers Dangling] |  |
| [Axivion Bauhaus Suite] | [7.2.0] | [Cert C++ MEM56] |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Dynamic Memory Allocation |

| **Noncompliant Code** |
| --- |
| Constant variable space is sent towards the next placement according to the pointer. From that call, the outcome pointer is sent towards the delete operator, that finishes with an undetermined behavior because of the delete operator trying to deliberate memory that wasn’t returned from the new() method operator. |
| #include <iostream>  struct S {  S() { std::cout << “S::S()” << std::endl; }  S() { std::cout<< “S::~S()” << std::endl; }  };  void f() {  alignas(struct S) char space[sizeof(struct S) ];  S \*s1 = new (&space) S;  delete s1;  } |

| **Compliant Code** |
| --- |
| Destroys the Delete operator [delete()], rather than an explicit remove callback method of ‘s1’. This is one of the examples where an explicit delete operator is used for a reasonable removal execution. |
| #include  struct S {  S() { std::cout << “S::S()” << std::endl; }  ~S() {std::cout << “S::~S()” << std::endl; }  };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;  S1->~S();  } |

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

| **Principles(s):** [Defense in Depth Practice – May cause a null pointer to have undefine behavior while output given is a new expression to the new operator in command.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [High] | [Likeable] | [Medium] | [P18] | [L1] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar] | [6.1P0] | [ALLOC.FNH; ALLOC.DF; ALLOC.TM] | [Free of not a heap variable]  [Double free]  [Mismatched Type] |
| [Klocwork] | [2021.3] | [CL.FFM.Assign;  CL.FFM.COPY;  CL.FMM;  FMM.MIGHT;  FMM.MUST;  FNH.MIGHT;  FNH.MUST;  FUM.GEN.MIGHT;  FUM.GEN.MUST;  UNINIT.CTOR.MIGHT;  UNINIT.CTOR.MUST;  UNINIT.HEAP.MIGHT;  UNINIT.HEAP.MUST;  UNINIT.STACK.ARRAY.MIGHT;  UNINIT.STACK.ARRAY.PARTIAL.MUST;  UNINIT.STACK.ARRAY.MUST;  UNINIT.STACK.MIGHT;  UNINIT.STACK.MUST] |  |
| [Clang] | [3.9] | [Clang\_Analyzer\_cplusplus.NewDeleteLeaks\_Wmismatched\_New\_DeleteClang\_Analyzer\_Unix.MismatchedDeallocator] | [Examine the clang tidy, it doesn’t catch all exceptions from given rule.] |
| [Astree] | [20.10] | [Invalid-dynamic-memory-allocation]  [Dangling Pointer Use] |  |
| [Helix QAC] | [2021.2] | [C++2110; C++2111; C++ 2112; C++2113; C++ 2118; C++ 3337; C++ 3339; C++ 4262; C++ 4263; C++ 4264] |  |
| [Axivion Bauhaus Suite] | [7.2.0] | [Cert C++ MEM51] |  |
| [PolySpace Bug Finder] | [R2021B] | [CERT C++ - MEM51-CPP] | [Invalid Deletion of Pointer]  [Deallocate any previous unassigned pointer.] |
| [PVS-Studio] | [7.15] | [V515; V554; V611; V701; V748; V773; V1066] |  |
| [SonarQube C/C++ Plugin] | [4.10] | [S1232] |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Testing the value of a constant argument, will require the need of static assertion. |

| **Noncompliant Code** |
| --- |
| When non-compliant code exits, we alter it by using the assert() method to a struct parameter that has memory which will test to comply with its proper behavior. |
| #include <assert.h>  struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  int func(void) {  Assert(sizeOf(struct timer) == sizeOf(unsigned char) + sizeOf(unsigned int) + sizeOf(unsigned int));  } |

| **Compliant Code** |
| --- |
| This static assertion detects runtime errors before being compiled which helps to avoid a new creation for a bug to be the outcome of the source code. |
| #include <assert.h>  struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  static\_assert(sizeOf(struct timer == sizeOf(unsigned char) + sizeOf(unsigned int) + sizeOf(unsigned int), “Structure must not have any padding”); |

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

| **Principles(s):** [Effective Quality Assurance Techniques – Enables great coding practices and tactics to use when testing sections of compiled errors in code.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Low] | [Likeable] | [High] | [P1] | [L3] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar] | [6.1P0] | [Customizable] | [Implementation of CPP check that accounted for the use in assert functions in macro.] |
| [LDRA Tool Suite] | [9.7.1] | [44 S] | [Full Implementation] |
| [Axivion Bauhaus Suite] | [7.2.0] | [CertC-DCL03] |  |
| [ECLAIR] | [1.2] | [CC2.DCL03] | [Full Implementation] |
| [Clang] | [3.9] | [misc-static-assert] | [Examined by Clang Tidy] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Exceptions are made to pledge for the safety of the given compiled code. |

| **Noncompliant Code** |
| --- |
| The class has variable members in an array format regards to its validity pointer where the numElem values that are stored (number of elements) are inside an array to aim to the next array. The method of this functionality is to unassign an array and place the element encounter “numElem”, before unassigning a new unit of memory for the copy. |
| #include <cstring>  Class intArray {  int \*array;  std::size\_t numElem;  public:  ~intArray() {  delete[] array;  }  intArray(const intArray& that); // significant copy constructor  intArray& operator =(const intArray &rhs) {  if(this != &rhs) {  delete[] array;  array = nullptr;  numElem = rhs.numElem;  if(numElem) {  array = new int[numElem];  std::memcpy(array, rhs.array, numElem \* sizeOf(\*array));  }  }  }; |

| **Compliant Code** |
| --- |
| Copy assignment operator assures the pledge for exception safety. This functionality assigns a new space (storage) for the copy given before the state of the object becomes modified. |
| #include <cstring>  Class intArray {  int \*array;  std::size\_t numElem;  public:  ~intArray() {  delete[] array;  }  ~intArray(const intArray& that); // significant copy constructor  intArray& operator =(const inArray &rhs) {  int \*tmp = nullptr;  if (rhs.numElem) {  tmp = new int [rhs.numElem];  std::memcpy(tmp, rhs.array, rhs.numElem, \* sizeOf(\*array));  }  delete[] array;  array = tmp;  numElem = rhs.numElem;  return \*this;  }  }; |

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

| **Principles(s):** [Embrace a Secure Coding Standard – Enables software engineers to be accountable in a high expectation standard when it secures the safety and security measurement protocols within the development project to consider new exception data to be promptly used, when assigning new data into a buffer/storage.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [High] | [Likeable] | [High] | [P9] | [L2] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [ParaSoft C/C++ Test] | [2021.2] | [CERT-CPP-MEM56-A] | [Enables Catch Exceptions without permitting ‘catch’ blocks to be left empty.] |
| [PVS-Studio] | [7.15] | [V565; V1023; V5002] |  |
| [Helix QAC] | [2021.2] | [C++ 4075; C++ 4076] |  |
| [PRQA QA-C++] | [4.4] | [4075; 4076] |  |
| [LDRA Tool Suite] | [9.7.1] | [527S; 56 D; 71 D] | [Partial Implementation] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| File Handling | [STD-008-CPP] | Goal is to reduce any amount of risk taken in consideration to proper handling files. |

| **Noncompliant Code** |
| --- |
| User’s or attackers can drain the system which may lead to future vulnerabilities which in this code it doesn’t call for destructors. |
| std::fstream file(fileName);  if (!file.is\_open()) {  // Handle Error  return;  }  std::terminate(); |

| **Compliant Code** |
| --- |
| Guarantees all compiled code and the system to be used properly. |
| 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):** [Standard is relevant to encryption in flight due to the communication from two software tools that are given. Once the file handling is done improperly then it could lead to a range of vulnerabilities where the program resources are drained and future exploits are created.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [High] | [Likeable] | [High] | [P1] | [L1] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC] | [2021.1] | [C++ 4786; C++ 4787; C++ 4788] |  |
| [ParaSoft C/C++ Test] | [2021.1] | [CERT-CPP-FIO51-A] | [Secure resources are freed] |
| [CodeSonar] | [6.0p0] | [ALLOC.LEAK] | [Leak] |
| [Klocwork] | [2021.1] | [RH.LEAK] |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Return Method Function | [STD-009-CPP] | Secure a returned value by all possible paths of a method functionality which escapes internal vulnerabilities and considerable future exploitations. |

| **Noncompliant Code** |
| --- |
| Will not return a value of any character or data type, if the conditional statement fails prior to the Boolean check. |
| int absolute\_value(int a) {  if (a < 0 ) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| Returns a compliant value each time, once after the code path is ran by the Boolean check. |
| int absolute\_value(int a) {  if (a<0) {  return -a;  }  return a;  } |

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

| **Principles(s):** [Encryption is relevant to this standard where it’s implemented, because it pertains protection to data traffic. Where its highly protected over vulnerabilities when executing possible paths from its functionality.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Low] | [Not Likeable] | [Low] | [P6] | [L5] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang] | [3.9] | [-WReturn-Type] | [Methods like ‘Try’ blocks are not caught by this rule.] |
| [Astree] | [20.10] | [Implicit Return] | [Fully Examined] |
| [CodeSonar] | [6.0p0] | [LANG.STRUCT.MRS] | [Undefined Return Statement] |
| [Axivion Bauhaus Suite] | [7.2.0] | [CertC ++ -MSC52] |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data Containers | [STD-010-CPP] | Considering the validity of a scale between the iterators that are secured by handling containers. |

| **Noncompliant Code** |
| --- |
| Suspicious data from source can go through the “int pos” argument/parameter as a negative value which the outcome is not valid in range. |
| void insert\_in\_table(int \*table, std::size\_t tableSize, int pos, int value) {  if (pos >= tableSize) {  // Handle Error  return;  }  table[pos] = value;  } |

| **Compliant Code** |
| --- |
| Whether pos is hashed as size\_t, it won’t be hashed in as a negative integer. |
| void insert\_in\_table(int \*table, std::size\_t tableSize, std::size\_t pos, int value) {  if (pos >= tableSize) {  // Handle Error  return;  }  table[pos] = value;  } |

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

| **Principles(s):** [Encryption is relevant to this standard due to the data protection when it’s implemented. In example can be a buffer overflow because it can direct this potential threat to the compromised data and/or expose undefined behavior when is implement.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [High] | [Not Likeable] | [Medium] | [P9] | [L3] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar] | [6.0p0] | [LANG.MEM.B0;  LANG.MEM.BU;  LANG.MEM.TO;  LANG.MEM.TU;  LANG.MEM.TBA;  LANG.STRUCT.PBB;  LANG.STRUCT.PPE] | [Used for Buffer Overflow, Underflow, Types of Overflow and Underflow, Corrupted buffer access, Pointer placed before and after of an object.] |
| [Klocwork] | [2021.1] | [ABV.ANY-SIZE-ARRAY;  ABV.GENERAL;  ABV.STACK;  ABV.TAINTED;  SV.TAINTED.ALLOC-SIZE;  SV.TAINTED.CALL.INDEX-ACCESS;  SV.TAINTED.CALL.LOOP-BOUND;  SV.TAINTED.INDEX-ACCESS] |  |
| [Astree] | [20.10] | [Overflow-Upon-Dereference] |  |
| [Helix QAC] | [2021.1] | [C++ 2891; C++ 3139; C++ 3140] |  |

### 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.

[Prior to automating enforcement of the standard’s policy, it’s very intuitive and smart to begin with security protocols first. Where code is developed, and unit testing are being implemented according to this standard policy. Any new function in code will not be worthless or even time will not be wasted if security features are used. Although it can make some slight changes when security features are adjusted in the code, but it will count as being secure, safe, and reliable to be efficient when tested. Before trying to commit any code to the local repository, any software developer will need to enforce the security feature policy first. Therefore, it’s very important to go over the security policy standards first, before trying to build new code or modify existing code which helps to make a program or system production way smoother when trying to catch errors before being compiled.]

### 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 | [Medium] | [Unlikely] | [Medium] | [Low] | [2] |
| STD-002-CPP | [High] | [Likely] | [High] | [Medium] | [2] |
| STD-003-CPP | [High] | [Likely] | [Medium] | [High] | [5] |
| STD-004-CPP | [High] | [Likely] | [Medium] | [High] | [5] |
| STD-005-CPP | [High] | [Likely] | [Medium] | [High] | [5] |
| STD-006-CPP | [Low] | [Unlikely] | [High] | [Low] | [1] |
| STD-007-CPP | [High] | [Likely] | [High] | [Medium] | [3] |
| STD-008-CPP | [High] | [Likely] | [High] | [High] | [1] |
| STD-009-CPP | [Low] | [Unlikely] | [Low] | [High] | [5] |
| STD-010-CPP | [High] | [Unlikely] | [Medium] | [Medium] | [3] |

### Create Policies for Encryption and Triple A

Include all three types of encryptions (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 protection from a disk that includes files for example of a database encryption. Implemented when data is allocated in storage and guarantee protection when data is compromised.] |
| Encryption at flight | [Data protection in transit, where it’s implemented by the data transfer from one location to the next by securing the channel path from global influence in such tools like a VPN. |
| Encryption in use | [Data protection when is implemented, where its applicable to confidential credentials or data in such as passwords, social profiles, bank accounts, and personal identity. Ensures that data is protected while it is in use.] |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | [Authentication is used when first validating a person’s identity either he/she is approved to be granted with access into a program/system.] |
| Authorization | [Authorization is a function/method used to verify once it is validated from a person’s identity to grant access to a level of depth within a software program where for many others it can be restricted.] |
| Accounting | [Accounting is the security measure of where or when it does take a person either new or existing to gain access into the program/system. How often or amount of data is being inserted while attempting to gain access. The number of attempts where software engineers have allowed a new or existing user to open the layers of defense in depth, which accountability to software security measures are approved to a new person prior of his/her privileges to a level of access in a program/system.] |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 01/20/2022 | Coding Standards | Joel Meza |  |
| 1.2 | 02/07/2022 | Project One | Joel Meza |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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