**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 | All user input will be validated from untrusted data sources. Remain suspicious of the majority of external data sources (including command line arguments, network interfaces, environmental variables, and user-controlled files). Proper input validates eliminates a lot of software vulnerabilities. |
| 1. Heed Compiler Warnings | Only compile code using the highest warning level available for the compiler you are using and eliminate warnings by modifying the code. Also, using static and dynamic analysis tools early on to detect and eliminate any additional security flaws that we can. |
| 1. Architect and Design for Security Policies | Create a software architecture and design for your software to implements and enforce security policies. |
| 1. Keep It Simple | Keep the design as simple and small as possible so that the likelihood of errors through implementation, configuration, and use are reduced. |
| 1. Default Deny | At base level, make decisions based on permission rather than exclusion. Basically, access is denied, and the protection scheme identifies conditions under which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Every single process should be able to execute with the least set of privileges necessary to run the program and complete the job. Elevated permission should be access for the least about of time required. This reduces the opportunities an attacker has to execute code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data that is passed to complex subsystems (such as command shells, relational databases, and commercial off-the-shelf components. |
| 1. Practice Defense in Depth | Manage risks with using multiple defensive strategies, having multiple layers of defense. |
| 1. Use Effective Quality Assurance Techniques | Using effective quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Such techniques include fuzz testing, penetration testing, and source code audits. |
| 1. Adopt a Secure Coding Standard | Applying a secure coding standard to your target development language and platform. |

### 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 | Do not cast to any out-of-range enumeration values |

| **Noncompliant Code** |
| --- |
| A check to determine whether a given value is within range of the acceptable enumeration values. Upon casting the type, it might not be able to represent the given integer value. |
| enum EnumType {  First;  Second;  Third;  };  void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  if (enumVar < First || enumVar > Third) {  }  } |

| **Compliant Code** |
| --- |
| This solution checks to see if the value represented by the enumeration type before performing the conversion to garuntee that the conversion doesn’t result in an unspecified value. It restartics the converted value to one enumerator type. |
| enum EnumType {  First;  Second;  Third;  };  void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  if (enumVar < First || enumVar > Third) {  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** An overflow of the buffer can be created with any unspecified values. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS - Studio | 7.07 | V1016 | \_\_\_\_\_\_\_ |
| Helix QAC | 2021.1 | \_\_\_\_\_\_\_ | \_\_\_\_\_\_\_ |
| PRQA QA C++ | 2020.2 | CERT\_CPP – INT50-A | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration. |
| Axivion Bauhaus Suite | 6.9.0 | CertC++ - INT50 | \_\_\_\_\_\_\_ |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | The use of the correct type of references, pointers, and iterations that reference elements of a container. |

| **Noncompliant Code** |
| --- |
| POS is deemed as invalid after the first call to insert() and the loop iterations have 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] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| POS is assigned to a valid iterator on each insertion, preventing the initial 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) {  pos = d.instert(pos, items[i] + 41.0);  }  } |

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

| **Principles(s):** Unidentified behavior can be a result of using the wrong pointers and references. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Overflow\_upon\_dereference | \_\_\_\_\_\_\_\_\_ |
| Helix QAC | 2021.1 | \_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_ |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-CTR51-a | Do not modify the container while iterating over it and avoid null pointer dereferencing |
| PVS – Studio | 7.07 | V783 | \_\_\_\_\_\_\_\_\_ |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Do not attempt to create a std::string from a null pointer. |

| **Noncompliant Code** |
| --- |
| A std:string object is create from the results to std:getenv(), but because std::getenv() returns a null pointer failure, there is undefined behavior when the environment variable does not exist. |
| #include <cstdlib>  #include <string>  void f() {  std::string tmp(std::getenv(“TMP”));    if (!tmp.empty()) {    }  } |

| **Compliant Code** |
| --- |
| The results from std::getenv() are checked for null before the std::string is constructed. |
| #include <cstdlib>  #include <string>  void f() {  const char \*tmpPtrVal = std::getenv(“TMP”);  std::string tmp(tmpPtrVal ? tmpPtrVal : “”);    if (!tmp.empty()) {    }  } |

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

| **Principles(s):** Undefined behavior can be caused by dereferencing a null pointer which can cause a program to terminate at random. This can be used by an attacker to exploit to execute their code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Assert\_failure | \_\_\_\_\_\_ |
| Helix QAC | 2021.1 | \_\_\_\_\_\_ | \_\_\_\_\_\_ |
| ParasoftC/C++ | 2020.2 | CERT\_CPP-STR51-a | Avoid any null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-0040CPP | Do not store already owned pointer values into an unrelated smart pointer. |

| **Noncompliant Code** |
| --- |
| Two unrelated smart pointers are constructed from the same underlying pointer value. When the local variable P2 is destroyed, the in turn deleted the pointer value that it manages. Then when the local automatic variable P1 is destroyed, it deletes the same pointer value, resulting in a double-free vulnerability. |
| #include <memory>  void f() {  int \*I = new int;  std::shared\_ptr<int> P1(i);  std::shared\_ptr<int> P2(i);  } |

| **Compliant Code** |
| --- |
| The std:shared\_ptr objects are related to each other through copy construction. When automatic variable P2 is destroyed, the shared pointer value is decremented and still nonzero and when the automatic variable P1 is destroyed, the use count for the pointer is decremented to zero, and the managed pointer is destroyed. It also calls std:make\_shared() instead of allocating a raw pointer and storing its value in a local variable. |
| #include <memory>  void f() {  int \*I = new int;  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):** Undefined behavior can be caused by passing a pointer value to a deallocation function that was not obtained previously by the matching allocation function, this can lead to exploitable vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Dangling\_pointer\_use | \_\_\_\_\_\_ |
| Helix QAC | 2021.1 | \_\_\_\_\_ | \_\_\_\_\_\_ |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-MEM56-a | Do not store already owned pointer values in an unrelated smart pointer |
| PVS – Studio | 7.01 | V1006 | \_\_\_\_\_\_ |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| The local variable is passed as the expression to the placement new operator. This results in the pointer to be passed to ::operator delete() and then throwing an undefined behavior due to ::operator delete() in an attempt to free memory that was not originally returned by ::operator new(). |
| #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** |
| --- |
| The call to ::operator delete() is removed and instead explicitly calls s1’s destructor. |
| #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;  } |

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

| **Principles(s):** Undefined behavior can be caused by passing a pointer value to a deallocation function that was not obtained previously by the matching allocation function, this can lead to exploitable vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Invalid\_dynamic\_memory\_alocation\_dangling\_pointer\_use | \_\_\_\_\_\_ |
| Axivion Bauhaus Suite | 6.9.0 | CERTC++MEM51 | \_\_\_\_\_\_ |
| Clang | 3.9 | Clang-analyzer-cplusplus.NewDelete.Leaks-Wminsmiatched-new-deleteclang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations |
| CodeSonar | 6.0p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM | Double free  Free non-heap variable  Type Mistmatch |
| Helix QAC | 2021.1 | \_\_\_\_\_\_ | \_\_\_\_\_\_ |
| Klocwork | 2021.1 | CL.FFM.ASSIGNFM  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 | \_\_\_\_\_\_ |
| LDRA Tool Suite | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D | Partially Implemented |
| Parasoft C/C++ | 2020.2 | CERT\_CPP-MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d | Using the same form in corresponding calls to new and delete. Always provide empty brackets for when deallocating arrays. |
| Polyspace Bug Finder | R2020a | CERT C++: MEM51-CPP | Checks for invalid deletion and free of pointer, deallocation of previously deallocated pointer. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Use static assertions to test the value of a constant expression. |

| **Noncompliant Code** |
| --- |
| Displaying the use of the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. |
| #include <assert.h>  Struct time {  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** |
| --- |
| Adding a preprocessor conditional statement for assertions that only involve constant expressions. |
| #include <assert.h>  Struct time {  Unsigned char MODE;  Unsigned int DATA;  Unsigned int COUNT;  }  #if(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  #error “Structure must not have any padding  #endif |

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

| **Principles(s):** Static assertion is a valuable diagnostic tool that is used for finding and eliminating defects in the software. However, not having static assertions does not mean that code is incorrect. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | \_\_\_\_\_\_ | Users can create a customer check that uses the assert() function. |
| ÉCLAIR | 1.2 | CC2.DCL03 | \_\_\_\_\_\_\_ |
| Clang | 3.9 | Misc-static.assert | Checked by clang-tidy |
| Axivion Bauhaus Suite | 6.9.0 | CERTC-DCL03 | \_\_\_\_\_\_ |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handles all exceptions thrown before main () begins executing. |

| **Noncompliant Code** |
| --- |
| The constructor for S may throw an exception that is not caught when globalS is constructed during program startup. |
| Struct S {  S() noexcept (false);  };  Static S globalS; |

| **Compliant Code** |
| --- |
| This solution makes globalS into a local variable with static storage duration, allowing any exceptions trhwon during object construction to be caught. The constructor for S will be executed for the first time the function globalS() is called rather than at program startup. It does not require source code to be modified so that the previous uses of globalS are replaced by a function (globalS). |
| Struct S {  S() noexcept(false);  };  S &globalS() {  Try {  Static S s;  Return s;  } catch (…) {  //Handle error  }  //Unreachable  } |

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

| **Principles(s):** Abnormal program termination can be a result of throwing an exception that cannot be caught, which can lead to attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Potentially-throwing-static-initialization | Patially Checked |
| Axivion Bauhaus Suite | 6.9.0 | CERTC++-ERR58 | \_\_\_\_\_\_ |
| Clang | 3.9 | CERT-EER58-CPP | Checked by clang-tidy |
| PRQA QA-C++ | 4.4 | 4634, 4636, 4637, 4639 | \_\_\_\_\_\_ |
| Rule Checker | 20.10 | Potentially-throwing-static-initialization | Partially Checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Object Oriented Programming** | STD-009-CPP | Do not invoke virtual functions from constructors or destructors. |

| **Noncompliant Code** |
| --- |
| The base class attempts to seize and release an object’s resources through calls to virtual functions from the constructor and destructor. The B::B() constructor calls B::sieze() rather than D::seize(). |
| Struct B {  B() {. Suize(); }  Virtual ~B() { release(); }  Protected:  Virtual void seize();  Virutal void release();  };  Struct D :B {  Virtual ~D() = default;  Protected:  Void seize() override {  B::Seize();  //Det derived resources  }  Void release() override {  //release derived resources  B::release()();  }  }; |

| **Compliant Code** |
| --- |
| The constructors and destructors call a nonvirtual, private member function instead of a virtual function. This means that each class is responsible for seizing and releasing its own resources. |
| class B {  B() {. Seize\_this(); }  Virtual ~B() { release\_this(); }  Protected:  Virtual void seize\_this();  Virutal void release\_this();  };  class D : public B {  void seize\_this();  void release\_this();  Protected:  Void seize() override {  B::Seize();  //Det derived resources  }  Void release() override {  //release derived resources  B::release()();  }  }; |

**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** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Virtual-call-in-constructor-invalid\_function\_pointer | Fully Checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-OOP50 | \_\_\_\_\_\_ |
| Clang | 3.9 | Clang-analyzer-alpha.cplusplus.VirtualCall | Checked by clangy-tidy |
| PRQA QA-C++ | 4.4 | 4260, 4261, 4273, 4274, 4275, 4276, 4277, 4278, 4279, 4280, 4281, 4282 | \_\_\_\_\_\_\_ |
| SonarQube C/C++ Plugin | 4.10 | S1699 | \_\_\_\_\_\_ |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Input Output** | STD-008-CPP | Do not alternate input and output file from a file stream without an intervening positioning call. |

| **Noncompliant Code** |
| --- |
| Shows that appends data to the end of a file and then reads from the same file. Because there is no intervening positioning call between the formatted output and input calls, the behavior is deemed as undefined. |
| #include <fstream>  #include <string>  Void f(const std::string &filename) {  Std::fstream file(filename);    If (!file.is\_open()) {  //Handle Error  Return;  }  File << “Output some data”;  Std::string str;  File >> str;  } |

| **Compliant Code** |
| --- |
| Uses the std::basic\_istream<T>::seekg() function is called between the output and input, elemeinating the original undefined behavior. |
| #include <fstream>  #include <string>  Void f(const std::string &filename) {  Std::fstream file(filename);    If (!file.is\_open()) {  //Handle Error  Return;  }  File << “Output some data”;  Std::string str;  File.seekg(0, std::ios::beg);  File >> str;  } |

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

| **Principles(s):** Alternatively inputting and outputting from a stream without an intervening flush or a positioning call will result in undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.1 | \_\_\_\_\_\_ | \_\_\_\_\_\_ |
| Parasoft C/C++ Test | 2020.2 | CERT\_CPP\_FIO50-a | Do not alternate input and output from a stream without intervening flush or a postioning call |
| Polyspace Bug Finder | R2020a | ECRT C++: FIO50-Cpp | Checks for alternating input and output from a stream.  Fully covered |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous | STD-010-CPP | Value returning functions must return a value from all exit paths. |

| **Noncompliant Code** |
| --- |
| There is no return for the input value for a positive input. |
| Int absolute\_value(int a) {  If (a < 0) {  Return -a;  }  } |

| **Compliant Code** |
| --- |
| The code paths now return an actual value. |
| 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):** Undefined behavior is caused by failing to return a value from a code path in a value-returning function, which may be exploited to cause data integrity violations. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Return-implicit | Fully Checked |
| Axivion | 6.9.0 | CertC++-MSC52 | \_\_\_\_\_\_\_ |
| Clang | 3.9 | Wreutnr-type | Does not catch all instances of the rule |
| CodeSonar | 6.0p0 | LANG.STRUCT.MRS | Missing return statement |
| LDRA tool suite | 9.7.1 | 2 D, 36 S | Fully Implemented |
| Parasoft C/C++ Test | 2020.2 | CERT\_CPP-MSC52-a | Everything needs a return statement |
| RuleChecker | 20.10 | Return-implicit | Fully Checked |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

Being mindful of Defense in Depth and making sure that tests are being ran early in the SDLC will ensure that any flaws and/or vulnerabilities are caught early on. Verify and test is the top priority in the pre-production phase, as it ensures that all problems are solved before production starts. This makes sure that all security flaws are exposed and addressed early on. Once production starts, it is important that transition and health check are being ran often. We see in the above diagram that in the pre-production phase, security faults are checked for and test to see what the vulnerabilities are. Then we see in the production phase that checks are ran to make sure that security flaws do not happen. This is an ongoing cycle to ensure that security is a top priority.

### 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 | Unlikely | Medium | High | 3 |
| STD-002-CPP | High | Probable | High | High | 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 | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Low | Likely | Low | High | 2 |
| STD-008-CPP | Low | Likely | Medium | High | 2 |
| STD-009-CPP | Low | Unlikely | Medium | Low | 3 |
| STD-010-CPP | Medium | Probable | Medium | High | 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 | Encryption in rest is used to prevent an attacker from being able to access any unencrypted data. This is done by ensuring that the data on the disk is encrypted. This means that if the attacker only obtains the hard drive and not the encryption keys, the attacker must be able to defeat the encryption to access and read the data on the hard drive. |
| Encryption at flight | Encryption at flight means to encrypt data while it is in the process of being transmitted. Some data is not stored on hard drives as encrypted but becomes encrypted while it is being transmitted. This adds a layer of protection to the data. |
| Encryption in use | Encryption in used means to enable access to encrypted data both at rest and data in motion. In other words, if somebody has an encryption key, data at rest that is encrypted can be decrypted. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of authenticating who someone is to be able to access the system. Such authentication can include user logins like a username and password combination for the user to be able to access the system. |
| Authorization | Authorization is the users level of access within the system. This can include admins or users, in which permissions differ based off the access level. Some users can make changes to the database or includes addition of new users. |
| Accounting | Accounting is the process of monitoring what a specific user is doing with their level of authorization within the system, such as keeping track of what files are accessed by users or what is being done in the database when accessed. |

**\***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 | 04/25/2022 | First Revision | Hanah Deering |  |
| 1.2 | 05/11/2022 | Second Revision | Hanah Deering |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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