**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 all inputs from untrusted or unknown data sources are validated we can prevent many of the major vulnerabilities in software that we may encounter. One rule of thumb is that we should always be aware of any external data sources. This includes but is not limited to command line arguments, network interfaces and environmental variables. |
| 1. Heed Compiler Warnings | The highest warning levels available should always be used in compiling code. Any warnings that can be eliminated through the modification of the code should be done so. In order to aid in detecting and to ultimately eliminate any additional security flaws both static and dynamic testing tools should be utilized. |
| 1. Architect and Design for Security Policies | It is important to keep in mind that when designing the software architecture that proper security policies are enforced and implemented. |
| 1. Keep It Simple | In order to prevent having to cater to more complex systems that are more at risk for errors and failed security mechanisms we must keep our coding design both small and simple. |
| 1. Default Deny | All access decisions should be standardized based on permissions as opposed to exclusion. By default, access should be denied with specific conditions to permit access. |
| 1. Adhere to the Principle of Least Privilege | The process executions should have, at minimum, the least set of privileges necessary to complete the job. Elevated permissions should only be accessed for the time that it takes to complete the task at hand. This is necessary to aid in reducing the potential risk of attackers using the arbitrary code located within those elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | The data that passes through complex subsystems should be sanitized on a regular basis to prevent attackers from using injection attacks to try and manipulate the components of the subsystems. |
| 1. Practice Defense in Depth | Remember to always practice defense in depth with multiple layers of security. This way when or if one layer fails there are other layers in place to help prevent security flaws from becoming exploitable. |
| 1. Use Effective Quality Assurance Techniques | By implementing quality assurance techniques, we can increase chances of identifying and eliminating vulnerabilities. We must also use testing phases, independent security reviews and external security reviews to help lead to an overall more secure system. |
| 1. Adopt a Secure Coding Standard | We must ensure that we always have and maintain a secure coding standard for development regardless of any language and platform we are using. |

### 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 an out-of-range enumeration value. |

| **Noncompliant Code** |
| --- |
| This code portion checks whether or not a given value is within range of acceptable enumeration values. After casting the type, it may 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) {  // Handle error } } |

| **Compliant Code** |
| --- |
| The compliant solution will check the value which is represented by the enumeration type before it performs the conversion. This will ensure that the conversion does not result in an unspecified value. As a result, the converted value is restricted to only one specific enumerator type. |
| enum EnumType { First, Second, Third };   void f(int intVar) {  if (intVar < First || intVar > Third) {  // Handle error } EnumType enumVar = static\_cast<EnumType>(intVar); } |

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

| **Principles(s):** It is possible for unspecified values to result in a buffer overflow, which then leads to the execution of arbitrary code by an attacker. However, due to enumerators being rarely used for indexing into arrays or other pointer forms, it is more likely that this scenario would result in data integrity violations rather than the aforementioned arbitrary code execution. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion  Bauhaus Suite | 6.9.0 | CertC++ - INT50 |  |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++  test | 2020.2 | CERT\_CPP – INT50-a | An expression with enum  underlying type shall only have  values corresponding to the  enumerators of the  enumeration |
| PRQA QA- C++ | 4.4 | 3013 |  |
| PVS – Studio | 7.07 | V1016 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Using valid references, pointers, and iterators to reference the elements of a container. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code block below is an example where pos is invalidated after the first initial call to insert(), and also the subsequent 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** |
| --- |
| The compliant solution below shows pos being assigned to a valid iterator on each insertion. This then prevents the 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.insert(pos, items[i] + 41.0); } } |

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

| **Principles(s):** By using invalid references, pointers or iterators to reference the elements of a container will result in undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Overflow\_unpon\_dereference |  |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++  test | 2020.2 | CERT\_CPP-CTR51-a | Do not modify container while  iterating over it |
| 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** |
| --- |
| In this noncompliant example code block, a std::string object is created from the results of a call to std::getenv(). But, because std::getenv() returns a null pointer upon failure this code can ultimately lead to 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** |
| --- |
| This compliant example code block below shows the results from the call to std::getenv() being checked for null prior to the std::string object 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):** By dereferencing a null pointer we create undefined behavior. This behavior is typically abnormal program termination. However, in some situations the dereferencing a null pointer can lead to the execution of the arbitrary code. [Jack 2007, Van Sprundel 2006]. The severity indicated for this more sever case is on platforms where it is impossible to exploit a null pointer dereference to execute the arbitrary code. The actual severity is considered low. |
| --- |

**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++  test | 2020.2 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Do not store pointer values that are already owned in an unrelated smart pointer. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code block below, two unrelated smart pointers are being constructed from the same underlying pointer value. When the local, automatic variable, p2, is destroyed it then deletes the pointer value that it manages. Next, when the local automatic variable, p1 , is destroyed it then deletes the same pointer value which results 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** |
| --- |
| In this compliant code block, the std::shared\_ptr objects are related to one another through the copy construction. When the local automatic variable, p2, is destroyed the use count for the shared pointer value is decremented but remains nonzero. Next the local automatic variable, p1, is destroyed the use count for the shared pointer value is decremented to zero. The managed pointer is then destroyed. This compliant solution also calls std::make\_shared() as opposed to allocating a raw pointer and storing the value in a local 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):** Passing a pointer value to a deallocation function that was not previously obtained by the matching allocation function results in undefined behavior which can lead to another exploitable vulnerability. |
| --- |

**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 an already owned  pointer value 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** |
| --- |
| In the noncompliant code block below, the local variable space is passed as the expression to the placement new operator. This then results in the pointer of that call being passed to ::operator delete(). This action results in undefined behavior due to it attempting to free memory that was not 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 compliant code block below removes the call to ::operator delete() and instead explicitly calls the s1’s destructor. This is one of the few times when explicitly invoking a destructor is warranted. |
| #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;  // ...   s1->~S(); } |

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

| **Principles(s):** Passing a pointer value to a deallocation function that was not previously obtained by the matching allocation function results in undefined behavior, which then 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\_d  angling\_pointer\_use |  |
| Axivion  Bauhaus Suite | 6.9.0 | CERTC++-MEM51 |  |
| Clang | 3.9 | clang-analyzercplusplus.  NewDeleteLeaks  -Wmismatched-new-delete  clang-analyzerunix.  MismatchedDeallocator | Checked by clang-tidy, but  does not catch all violations of  this rule |
| CodeSonar | 6.0p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM | Free non-heap variable  Double free  Type mismatch |
| 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  UNINIT.STACK.ARRAY.MIGHT  UNINIT.STACK.ARRAY.MUST  UNINIT.STACK.MIGHT  UNINIT.STACK.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++  test | 2020.2 | CERT\_CPP-MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d | Use the same form in  corresponding calls to new/malloc  and delete/free  Always provide empty brackets ([])  for delete when deallocating  arrays  Both copy constructor and copy  assignment operator should be  declared for classes with a  nontrivial destructor  Properly deallocate dynamically  allocated resources |
| Parasoft Insure  ++ |  |  | Runtime Detection |
| Polyspace Bug  Finder | R2020a | CERT C++: MEM51-CPP | Checks for:  Invalid deletion of pointer  Invalid free of pointer  Deallocation of previously  deallocated pointer  Rule partially covered. |
| PRQA QA-C++ | 4.4 | 2110, 2111, 2112, 2113, 2118,  3337, 3339, 4262, 4263, 4264 |  |
| PVS – Studio | 7.07 | V515, V554, V611, V701, V748, V773 |  |
| SonarQube  C/C++ Plugin | 4.10 | S1232 |  |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| In the noncompliant code block below the code uses 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 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** |
| --- |
| In the compliant code block below, we see assertions involving only constant expressions a preprocessor conditional statement can be used. |
| struct timer {  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 assertions is a valuable diagnostic tool for finding and eliminating software that defects that may result in vulnerabilities at the time of the compile. The absence of static assertions does not mean that the code is then incorrect. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion  Bauhaus Suite | 6.9.0 | CERTC-DCL03 |  |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 6.0p0 | “customization” | Users can implement a custom  check that reports uses of the  assert() macro |
| Compass/Rose |  |  | Could detect violations of this rule  merely by looking for calls to  assert(), and if it can evaluate the  assertion (due to all values being  known at compile time), then the  code should use static-assert  instead; this assumes ROSE can  recognize macro invocation |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| In the noncompliant code block below, the constructor for S may throw an exception that is not caught when globalS is constructed during the startup of the program. |
| struct S {  S() noexcept(false); };  static S globalS; |

| **Compliant Code** |
| --- |
| The compliant code block below shows the solution is to make globalS into a local variable with static storage duration. This will then allow any exceptions thrown during the object construction to be caught because the constructor for S will be executed the first time the function globalS() is called as opposed to at the startup of the program. This solution does however require the programmer to modify source code so that it the previous uses of globalS are replaced by a function call to globalS(). |
| struct S {  S() noexcept(false); };  S &globalS() {  try {  static S s;  return s; } catch (...) {  // Handle error, perhaps by logging it and gracefully terminating the application. } // Unreachable. } |

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

| **Principles(s):** Throwing an exception that cannot be caught will result in abnormal program termination and can ultimately lead to denial of service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | [Insert text.] potentially-throwingstatic-  initialization | Partially Checked |
| Axivion  Bauhaus Suite | 6.9.0 | CERTC++-ERR58 |  |
| Clang | 3.9 | Cert-eer58-cpp | Checked by clang-tidy |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++  test | 2020.2  CERT\_CPPERR58-  a |  | Exceptions shall be raised only  after start-up and before  termination of the program |
| 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** |
| --- | --- | --- |
| Input/Output | [STD-008-CPP] | Do not alternate input and output from a file stream without an intervening positioning call. |

| **Noncompliant Code** |
| --- |
| In the noncompliant code block below the data is appended to the end of a file and then read from the same file. But, because there is no intervening position call between the formatted output and input calls the behavior is then considered 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** |
| --- |
| In the compliant code block below the solution is for std::basic\_istram<T>::seekg() function to be called between the output and the input. This will then eliminate the 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";  Compliant Code   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):** Alternating the input and output from a stream without an intervening flush or positioning call is 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 alternately input and  output from a stream without an  intervening flush or positioning  call |
| Polyspace Bug  Finder | R2020a | ECRT C++: FIO50-CPP | Checks for alternating input and  output from a stream without  flush or positioning call (rule fully  covered) |

#### Coding Standard 9

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

| **Noncompliant Code** |
| --- |
| In the noncompliant code block below, the base class attempts to seize and release and object’s resources through calling to virtual functions from the constructor and destructor. But, the B::B() constructor calls B::seize() rather than D::seize(). Alternatively, the B::~B() destructor calls B::release() rather than D::release(). |
| struct B {  B() { seize(); }  virtual ~B() { release(); }  protected:  virtual void seize();  virtual void release(); };  struct D : B {  virtual ~D() = default;  protected:  void seize() override {  B::seize();  // Get derived resources... }  void release() override {  // Release derived resources...  B::release(); } }; |

| **Compliant Code** |
| --- |
| In the compliant code block below the constructors and destructors call a nonvirtual, private member function instead of calling a virtual function. This results in each class being responsible for seizing and releasing its own resources. |
| class B {  void seize\_mine(); void release\_mine();  public: B() { seize\_mine(); } virtual ~B() { release\_mine(); }  protected: virtual void seize() { seize\_mine(); } virtual void release() { release\_mine(); } };  class D : public B { void seize\_mine(); void release\_mine();  public: D() { seize\_mine(); } virtual ~D() { release\_mine(); }  protected: void seize() override { B::seize(); seize\_mine(); }  void release() override { release\_mine(); B::release(); } } |

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

| **Principles(s):** |
| --- |

**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 clang-tidy |
| Helix QAC | 2021.1 |  |  |
| LDRA tool suite | 9.7.1 | 467S, 92D | Fully implemented |
| Parasoft C/C++  test | 2020.2 | CERT\_CPP-OOP50-a  CERT\_CPP-OOP50-b  CERT\_CPP-OOP50-c  CERT\_CPP-OOP50-d | Avoid calling virtual functions from  constructors  Avoid calling virtual functions from  destructors  Do not invoke class's virtual  functions from any of its  constructors  Do not invoke class's virtual  functions from its destructor |
| PRQA QA-C++ | 4.4 | 4260, 4261, 4273, 4274,  4275, 4276, 4277, 4278,  4279, 4280, 4281, 4282 |  |
| PVS-Studio | 20.10 | Virtual-call-in-customer | Fully checked |
| SonarQube  C/C++ Plugin | 4.10 | S1699 |  |

#### 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** |
| --- |
| In this noncompliant code block below, the programmer forgot to return the input value for positive input, so not all code paths return a value. |
| int absolute\_value(int a) {  if (a < 0)  {  return -a; } } |

| **Compliant Code** |
| --- |
| In the compliant code block below, all the code paths now are set to return a 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):** Failing to return a value from a code path in a value-returning functions results in undefined behavior that could then be exploited to cause data integrity vulnerabilities or violations. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Return-implicit | Fully chcecked |
| Axivion | 6.9.0 | CertC++-MSC52 |  |
| Bauhaus Suite |  |  |  |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this  rule, such as function-try-blocks |
| CodeSonar | 6.0p0 | LANG.STRUCT.MRS | Missing return statement |
| Helix QAC | 2021.1 |  |  |
| LDRA tool suite | 9.7.1 | 2 D, 36 S | Fully implemented |
| Parasoft C/C++  test | 2020.2 | CERT\_CPP-MSC52-a | All exit paths from a function with  non-void return type shall have an  explicit return statement with an  expression |
| Polyspace Bug  Finder | R2020a | Cert C++: MSC52-a | Checks for missing return  statements (rule partially covered) |
| SonarQube  C/C++ Plugin | 4.10 | S935 |  |
| PRQA QA- C++ | 4.4 | 1510 |  |
| PVS-Studio | 7.07 | V591 |  |
| 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.

DevOps transforms to DevSecOps

by integrating security measures into each step of the DevOps toolchain. Within the pre-production phases, threat modeling and security tool training and selections are added to the “Assess and Plan” segment. In the “Design” and “Build” phases, IDE security is addressed. Static application testing and automated security scans are added to the “Verify & Test” phase along with unit, integration, and other tests.

Once in production, the automated testing continues with prevention by using integrity checks and defense-in-depth measures. Network monitoring, penetration testing, network monitoring and performance logs are some methods of continuous threat detection. Just as with testing in the QA sense, testing in the security sense should also be performed early and often.

### 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 | P4 | L3 |
| STD-002-CPP | High | Probable | High | P6 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CPP | Low | Likely | Medium | P6 | L2 |
| STD-009-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-010-CPP | Medium | Probable | Medium | P8 | L2 |

### 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 | Protects data where it is stored (i.e.: on a computer/phone, a database, in cloud). This may include physical sources such as hard drives, or logical sources such as databases or cloud assets. There are several options for encryption tools, such as VeraCrypt, AxCrypt or InnoDB tablespace encryption. These tools are necessary to protect data from being physically stolen, logically stolen, or otherwise be breached. |
| Encryption at flight | Protects data as it is moved from one location to another (i.e.: sending an email, browsing the internet). Email encryption tools such as S/MIME or PGP shall be used for email transmission. Web traffic shall only be sent over a Secure Sockets Layer (SSL) such as Transport Layer Security (TLS) by obtaining an SSL/TLS HTTPS certificate from authorities such as GoDaddy or DigiCert. Refer to Table 1 below for suggested secure network protocols. Using these tools prevents network layer attacks, such as eavesdropping, and tampering-based attacks, such as third-party communication hijacking. |
| Encryption in use | Protects data as it is being created, edited, accessed, processed, or viewed. This state occurs in between the at-rest and at-flight states when, say for instance, accessing a website on a server, or whenever the CPU is in use processing applications. It is important to encrypt data-in-use because memory can be hacked, and encryption keys for data-at-rest may be exposed. CPU manufacturer AMD offers full memory encryption, called Secure Memory Encryption (SME), and Intel offers Total Memory Encryption (TME) in order to protect CPU based key storage. There are also cryptographic tools that can be used to protect data during computation |
|  |  |

Diagram

Description automatically generated

|  |  |  |
| --- | --- | --- |
| Transfer Type | What to avoid (insecure) | What to use (secure) |
| Web Access | HTTP | HTTPS |
| E-Mail Servers | POP3, SMTP, IMAP | POP3S, IMAPS, SMTPS |
| File Transfer | FTP, RCP | FTPS, SFTP, SCP, WebDAV over HTTPS |
| Remote Shell | telnet | SSH2 |
| Remote Desktop | VNC | radmin, RDP |

### Table 1. Secure Data-In-Transit Protocols

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Who are you? Authentication verifies a user’s identity credentials. This can be achieved in several ways, including username/password verification, single sign-on (SSO) systems, biometrics, I and/or digital certificates, for example. All users, new and existing, shall be verified. Identity theft and unauthorized system access can result if user credentials are not authenticated. |
| Authorization | What can you use? Authorization is set for each existing and new user and defines the level of access to files, directories and/or applications for each individual user. Each user or group shall be permitted to have read, write and/or execution permission, depending on the user’s role within an organization. For example, a data entry clerk may have read / write access to view or edit a particular file while a manager may warrant administrative permissions. |
| Accounting | What happened and when? Accounting refers to records, or log files, which detail things like user logins; new user profile creations; file access events; database updates; data transfers or access. All actions are date and timestamped as a record of occurrence. Usage information can track events such as authorization or resource utilization which can be used in turn for system wide planning. Accounting tracks who is doing what at all times. |

**\***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 | 11/14/2021 | Initial Revision | Justin Vallia |  |
| 1.2 | 12/01/2021 | Final Amendment(s) and Revision(s) | Justin Vallia |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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