**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 | Validating input data helps reduce the risk for most vulnerabilities. We must be aware of how input data is used to properly validate this requirement. |
| 1. Heed Compiler Warnings | Compiler errors and warnings are important. These issues need to be solved to properly execute the code. Security flaws will continue to exist if these errors are not corrected. |
| 1. Architect and Design for Security Policies | Security policies must be addressed in the design of the application. It is important to do so otherwise will leave security gaps in the development. |
| 1. Keep It Simple | Complexity can introduce errors into the code. It is best practice to keep the code as simple as possible. |
| 1. Default Deny | Denying access by default is a way of standardizing access for specific events. If events cannot be handled in a specific way, they should be denied by default. |
| 1. Adhere to the Principle of Least Privilege | Allowing process only the exact amount of time to execute as needed to complete to task. Reducing the amount of time processes run reduces the overall risk to vulnerability. |
| 1. Sanitize Data Sent to Other Systems | Injection attacks can be filtered out using sanitization techniques. This ensures that data sent between the two systems remains untainted. |
| 1. Practice Defense in Depth | Use of multiple layers of security is important. This ensures that no two systems are reliant on each other for security purposes. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance should be an ongoing process of testing, reviewing and suggesting changes to improve existing code. |
| 1. Adopt a Secure Coding Standard | Always adhere to a coding standard such as CERT C++ to ensure that code is developed with the best practices in mind. This will observe the most current information on how to minimize the risk of vulnerabilities. |

### 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-nnn-LLL] | Casting to an out of range enumeration is not recommended. |

| **Noncompliant Code** |
| --- |
| Checks to see if enumeration value is within range. No error check. |
| enum EnumType {  One,  Two,  Three  };  void func(int intVariable) {  EnumType enumVar = static\_cast<EnumType>(intVariable);  if (enumVar < One || enumVar > Three) {  }  } |

| **Compliant Code** |
| --- |
| Checks to see if enumeration is within range with error checking. |
| enum EnumType {  One,  Two,  Three  };  void func(int intVariable) {  if (intVariable < One || intVariable > Three) {  }  EnumType enumVar = static\_cast<EnumType>(intVariable);  } |

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

| **Principles(s):** Unspecified values can result in a buffer overflow. This can also lead to the execution of arbitrary code by a threat actor. Alternatively, because enumerators are hardly used for indexing into arrays or other forms of pointer arithmetic it is more likely that a data integrity issue would occur rather than an 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] | Use valid references, pointers and iterators to reference elements of a container. |

| **Noncompliant Code** |
| --- |
| In this example, POS is invalidated after first call to insert() and there is undefined behavior in the following loop. |
| #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** |
| --- |
| In the compliant example there is an assigned validator on each insertion of POS. This 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):** Use of invalid references, pointers, or iterators to reference elements of a container results 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 example on noncompliant code, there is a std:string object from the results of the call to std:getenv(). This leads to undefined behavior because std::getenv() returns a null pointer failure. This can lead to undefined behavior since the variable does not exist. |
| #include <cstdlib>  #include <string>  void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...    }  } |

| **Compliant Code** |
| --- |
| In this solution, the compliant code is checking the results from std::getenv() before 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):** A dereferenced null pointer is considered undefined behavior which will typically result in abnormal program termination. Some cases this may cause arbitrary code to execute. This is considered a severe case but on platforms where it is not possible to exploit a null pointer to execute arbitrary code the severity is 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 an already owned pointer value in an unrelated smart pointer. |

| **Noncompliant Code** |
| --- |
| In this code example we have a double free vulnerability because the variable P2 destroys the value it manages then when P1 is destroyed it also deletes the same pointer value. |
| #include <memory>  void f() {  int \*i = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i);  } |

| **Compliant Code** |
| --- |
| In a compliant solution example we use copy construction for the std::shared\_ptr objects. Once the P2 value is destroyed this will decrement the number to a non-zero number. Finally, when P1 is destroyed the use count is decremented to zero. This solution calls on std::make\_shared() to store its local variable instead of a null pointer. |
| #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): Using a deallocation function to pass a pointer value that weas not previously obtained by the matching allocation function results in undefined behavior which can lead to high severity 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 an already-owned pointer 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 this code example the local variable is passed to an expression to the placement of a new operator. Undefined behavior results from when ::operator delete() is passed to the pointer. This undefined behavior causes ::operator delete() to attempt to free memory that is not returned. |
| #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** |
| --- |
| In the compliant example we remove the call to ::operator delete(). Instead of the previous method this is changed to explicitly call the s1 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;  // ...  s1->~S();  } |

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

| **Principles(s):** Passing pointer values to a deallocation function not previously obtained by the matching allocation function results in undefined behavior. 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.NewDeleteLeaks  -Wmismatched-new-delete  clang-analyzer-  unix.MismatchedDeallocator | Checked by clang-tidy, but this does not catch every violation 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 | 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, V77 |  |
| SonarQube  C/C++ Plugin | 4.10 | S1232 |  |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| This code uses the assert() macro to assert a property concerning a memory-mapped structure which is essential to the application 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** |
| --- |
| A preprocessor conditional statement may be used for insertions involving only constant expressions. |
| #include <assert.h>  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 assertion is a valuable tool for diagnostics. It is very useful for finding and eliminating software defects that may generate vulnerabilities at runtime. A lack of static assertions does not mean that the code is 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 with 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 is executed. |

| **Noncompliant Code** |
| --- |
| The constructor S may throw an exception that might not be caught when globalS is constructed during the program startup. |
| struct S {  S() noexcept(false);  };  static S globalS; |

| **Compliant Code** |
| --- |
| For the complaint solution globalS is made a local variable with static storage duration. This allows for all exceptions to be thrown to be caught because S will initiate once globalS initiates. |
| 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 which cannot be caught results in abnormal program termination. This can result in a denial-of-service-attack. |
| --- |

**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 | Partially Checked |
| Axivion Bauhaus Suite | 6.9.0 | CERTC++-ERR58 |  |
| Clang | 3.9 | Cert-eer58-cpp | Checked with clang-tidy |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-ERR58-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** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Do not alternate input and output from a file stream without an intervening positioning call. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example data is appended to the file and read then read from the same file. The behavior is undefined because of no intervening positioning call. |
| #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** |
| --- |
| For the compliant solution std::basic\_istream<T>::seekg() function is called between the output and input eliminating 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";  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):** Alternately inputting and outputting 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  ca |
| 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** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | Do not invoke virtual functions from constructors or destructors |

| **Noncompliant Code** |
| --- |
| In the noncompliant example the base class attempts to seize and release an objects resource through calls to virtual functions from the constructor and destructor. In this case B::B() calls to B::sieze instead of D::sieze. |
| 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** |
| --- |
| For the solution the constructors and destructors call a nonvirtual, private member function instead of calling a virtual function. |
| 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): Invoking a class’s virtual function from a constructor can create a vulnerability and abnormal program behavior.** |
| --- |

**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 with 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 | 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 destructo |
| PRQA QA-C++ | 4.4 | 4260, 4261, 4273, 4274,  4275, 4276, 4277, 4278,  4279, 4280, 4281, 428 |  |
| 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** |
| --- | --- | --- |
| Misc | [STD-010-CPP] | Value returning functions must return a value from all exit paths. |

| **Noncompliant Code** |
| --- |
| In this case the code does not return the input value for positive input. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| In the solution all input returns 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 function will result in undefined behavior that could be exploited to cause data integrity 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 checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC++MSC52 |  |
| CodeSonar | 6.0p0 | LANG.STRUCT.MR5 | Does not catch all instances of this rule, such as function-try-blocks |
| 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.

One consideration here is defense in depth. It is important to test early and discover flaws using unit testing. This will help save time and reduce cost in the long run.

### 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 | 2 |
| 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 | P6 | L2 |
| STD-010-CPP | Medium | Probable | Medium | P8 | L2 |

### 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 designed to prevent an attacker from accessing unencrypted data that might be stored digitally. Even if a hard drive is obtained the data is useless to the attacker if encryption is in place. |
| Encryption at flight | This is for example TLS/SSL encryption that protects data as it is moving across the network. This ensures that an attacker cannot intercept the data and be able to read the contents. |
| Encryption in use | By encrypting data in use for example encrypting memory we prevent anyone that might try to read that memory from being able to recover keys that may help them exploit the encryption in motion or in rest. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is confirming the user is someone who should have access to the system. This is typically a login and password. More sophisticated methods involve two factor authentication. |
| Authorization | Authorization is the level of access within the system the user is allowed to have. Some users may only need to read files while others may need to modify files in order to do their tasks. |
| Accounting | Accounting is being able to monitor access to the system and keep record of the logs for usage. |

**\***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 | 05/21/2022 | Initial Draft | Nathaniel Madore |  |
| 1.3 | 05/21/2022 | Revision | Nathaniel Madore |  |
| 1.6 | 06/12/2022 | Project One Revision | Nathaniel Madore |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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