**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 | Validate input from all untrusted data sources. Input validation can eliminate the majority of software vulnerabilities. Verify all external data sources, including command line arguments, network interfaces, environmental variables, and user-controlled files. |
| 1. Heed Compiler Warnings | Use the highest warning level available for your compiler and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and remove more security flaws. |
| 1. Architect and Design for Security Policies | Create software architecture that implements and enforces security policies. If your system requires different privileges, consider dividing it into distinct intercommunicating subsystems, each with an appropriate privilege set. |
| 1. Keep It Simple | Keep the design as simple and small as possible to reduce the amount of effort required to keep the system secure. |
| 1. Default Deny | Use permission to determine access rather than exclusion. Deny access by default and the protection scheme identifies conditions in which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Processes should be executed with the least amount of privileges necessary. Set maximum time allotted to complete privileged tasks that require elevated permission. This reduces the amount of opportunities an attacker has to execute attacks with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed through command shells, relational databases, and commercial off the shelf components. Attackers may exploit unused functionality in these components through the use of SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem does not understand the context in which the call is made. |
| 1. Practice Defense in Depth | Create multiple defense layers to prevent attacks on several levels. If 1 attack is successful the attacker will be required to breach additional layers before data can be compromised. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance can help identify and eliminate potential vulnerabilities. Fuzz testing, penetration testing, and source code audits can create more effective quality assurance strategies. External reviewers can help provide an outside perspective which can provide added insight into potential threats. |
| 1. Adopt a Secure Coding Standard | Utilize secure coding standards catering to your chosen 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 an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| Check range of the given value to make sure it is within acceptable values. Casting the type may not represent 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 solution to the code above is to make sure the value is correct before conversion to prevent an incorrect value from being represented. Restricts value after conversion to 1 enumerator type. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  if (intVar < First || intVar > Third) {  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** Unspecified values may cause buffer overflow which an attacker can exploit with arbitrary code. This issue is more likely to cause data integrity violations rather than arbitrary code execution since enumerators aren’t commonly used for indexing data into arrays. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree’ | 22.10 | Cast-integer-to-enum | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| CodeSonar | 8.1p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2024.2 | C++3013 |  |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| PVS-Studio | 7.33 | V1016 |  |
| RuleChecker | 22.10 | Cast-integer-to-enum | Partially checked |
| Polyspace Bug Finder | R2024a | CERT C++:INT50-CPP | Checks for casting to out of range enumeration value(rule fully covered) |

#### 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 of noncompliant code, pos is invalidated after insert() call causing the loop to have an unexpected outcome. |
| #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 this compliant version, pos is assigned a valid iterator after each insertion correcting unexpected 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):** Avoid undefined behavior by ensuring appropriate usage of pointers, references, or iterators for elements of a container. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.UAF | Use After Free |
| Helix QAC | 2024.2 | DF4746, DF4747, DF4748, DF4749 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR52-a | Use valid references, pointers, and iterators to reference elements of a basic\_string |
| Polyspace Big Finder | R2024a | CERT C++:STR52-CPP | Checks for use of invalid string iterator (rule partially covered) |

#### 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** |
| --- |
| This example shows a std::string created as a result of a call to std::getenv(). This returns a null pointer when fails so this can lead to unexpected behavior when no variable is recorded. |
| #include <cstdlib>  #include <string>  void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| In this correction the result of calling std::getenv() is checked for null before creating the std::string object. |
| #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):** Some attackers may use dereferencing of a null pointer to execute arbitrary code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree’ | 22.10 | Assert\_failure | [Insert text.] |
| CodeSonar | 8.1p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Helix QAC | 2024.2 | DF4770, DF4771, DF4772, DF4773, DF4774 | [Insert text.] |
| Klocwork | 2024.2 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF | [Insert text.] |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2024a | CERT C++:STR51-CPP | Checks for string operations on null pointer (rule partially covered) |

#### Coding Standard 4

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

| **Noncompliant Code** |
| --- |
| 2 unrelated pointers are created using the same pointer value. When local automatic variable p2 is destroyed it deleted the existing pointer value. Then when p1 is destroyed it deletes the same pointer value. This 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 example when p2 is destroyed, the use count for the shared point value is decremented but not zero. When p1 is destroyed the use count for the shared pointer is decremented to zero and the pointer is destroyed. We also use std::make\_shared() instead of using a raw pointer and storing 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 into a deallocation function that was not appropriately used can lead to undefined behavior that can lead to exploitation. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree’ | 22.10 | Dangling\_pointer\_use |  |
| Axivion Bauhaus | 7.2.0 | CertC++-MEM56 |  |
| Helix QAC | 2024.2 | DF4721, DF4722, DF4723 |  |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-MEM56-a | Do not store an already owned pointer value in an unrelated smart pointer |
| Polyspace Bug Finder | R2024a | CERT C++: MEM56-CPP | Checks for use of already owned pointers(rule fully covered) |
| PVS-Studio | 7.33 | V1006 |  |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| Local variable space passed as expression to placement new operator. The result is then passed to ::operator delete(). This results in unexpected behavior since ::operator delete() attempts to free memory that is 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** |
| --- |
| This code removed the call to ::operator delete(). The code now 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;  // ...  s1->~S();  } |

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

| **Principles(s):** Passing a pointer to a deallocation function can lead to undefined behavior which can then lead to attackers exploiting this vulnerability. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree’ | 22.10 | Invalid\_dynamic\_memory\_allocation\_dangling\_pointer\_use |  |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM51 |  |
| Clang | 3.9 | Clang-analyzer-cplusplus.NewDeleteLeaks | Checked by clang-tidy, but does not catch all violations of this rule |
| CodeSonar | 8.1p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM  ALLOC.LEAK | Free non-heap variable  Double free  Type mismatch  Leak |
| Helix QAC | 2024.2 | C++2110, C++2111, C++2112. C++2113. C++2118, C++3337, C++4262, C++ 4263, C++4264 |  |
| Klocwork | 2024.2 | CL.FFM.ASSIGN  CL.FFM.COPY  CL.FMM  CL.SHALLOW.ASSIGN  CL.SHALLOW.COPY  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++ test | 2023.1 | CERT\_CPP-MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d | 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 |

#### 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** |
| --- |
| This code uses assert() macro to assert a property for memory mapped structure that is required to execute as expected. |
| #include <assert.h>  struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned  int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| This code uses a preprocessor conditional statement since constant expressions are involved. |
| 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 an effective diagnostic tool to identify and remove software defects. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus | 7.2.0 | CertC-DCL03 |  |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 8.1p0 | (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 main() begins executing |

| **Noncompliant Code** |
| --- |
| In this code, the constructor S throws an exception that is not recognized when globalS is constructed during startup |
| struct S {  S() noexcept(false);  };  static S globalS; |

| **Compliant Code** |
| --- |
| In this code globalS is a local variable with static storage duration. This allows exceptions thrown during object construction to be caught. This is possible since S is executed when globalS() is called instead of during startup. Prior uses of globalS are replaced by 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):** Exceptions that can’t be caught may result in denial of service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree’ | 22.10 | potentially-throwing-static-initialization | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR58 |  |
| Clang | 3.9 | cert-err58-cpp | Checked by clang-tidy |
| CodeSonar | 8.1p0 | LANG.STRUCT.EXCP.THROW | Use of throw |
| Helix QAC | 2024.2 | C++4634, C++4636, C++4637, C++4639 |  |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-ERR58-a | Exceptions shall be raised only after start up and before termination of program |
| Polyspace Bug Finder | R2024a | CERT C++:ERR58-CPP | Checks for exceptions raised during program startup |
| RuleChecker | 22.10 | potentially-throwing-static-initialization | Partially checked |

#### Coding Standard 8

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

| **Noncompliant Code** |
| --- |
| This code appends data to the end of a file and then reads from that same file. A position must be created to intervene between input and output calls. |
| #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** |
| --- |
| This solution calls the function std::basic\_istream<T>::seekg() between input and output. |
| #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):** You can prevent undefined behavior by using an intervening flush or positioning call when inputting and outing from a stream |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree’ | 24.04 |  | Supported, but no explicit checker |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO39 |  |
| CodeSonar | 8.1p0 | IO.IOWOP  IO.OIWOP | Input After Output Without Positioning  Output After Input Without Positioning |
| Compass/ROSE |  |  | Can detect simple violations of this rule |
| Helix QAC | 2024.2 | DF4711, DF4712, DF4713 |  |
| Klocwork | 2024.2 | CERT.FIO.NO\_FLUSH |  |
| LDRA tool suite | 9.7.1 | 84 D | Fully implemented |
| Parasoft C/C++ test | 2023.1 | CERT\_C-FIO39-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| PC-lint Plus | 1.4 | 2478, 2479 | Fully supported |
| Polyspace Bug Finder | R2024a | CERT C: Rule FIO39-C | 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** |
| --- |
| [Noncompliant description] |
| 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 this code constructors and destructors call a nonvirtual, private member function rather than a virtual function. In this case each class seizes and releases 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):** In either case, f() must be the final overrider, guaranteeing consistent behavior of the function being called. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree’ | 22.10 | virtual-call-in-constructor  invalid\_function\_pointer | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP50 |  |
| Clang | 3.9 | clang-analyzer-alpha.cplusplus.VirtualCall | Checked by clang-tidy |
| CodeSonar | 2024.2 | C++4260, C++4261, C++4273, C++4274, C++4275, C++4276, C++4277, C++4278, C++4279, C++4280, C++4281, C++4282 |  |
| Klocwork | 2024.2 | CERT.OOP.CTOR.VIRTUAL\_FUNC |  |
| LDRA tool suite | 9.7.1 | 467 S, 92 D | Fully implemented |
| Parasoft C/C++ test | 2023.1 | 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 use dynamic type of an object under construction  Do not use dynamic type of an object under destruction |
| Polyspace Bug Finder | R2024a | CERT C++: OOP50-CPP | Checks for virtual function call from constructors and destructors (rule fully covered) |
| PVS-Studio | 7.33 | V1053 |  |
| RuleChecker | 22.10 | virtual-call-in-constructor | 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** |
| --- |
| No value is returned if the input is a positive value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| This code accounts for positive and negative inputs |
| 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):** Failure to return a value from a code path in value returning function may result in exploitation and cause integrity violations. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree’ | 22.10 | Return-implicit | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MSC52 |  |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this rule, such as function-try-blocks |
| CodeSonar | 8.1p0 | LANG.STRUCT.MRS  LANG.STRUCT.NVNR | Missing return statement  Non-void noreturn, |
| Helix QAC | 2024.2 | DF2888 |  |
| Klocwork | 2024.2 | FUNCRET.GEN  FUNCRET.IMPLICIT |  |
| LDRA tool suite | 9.7.1 | **2 D, 36 S** | Fully implemented |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-MSC52-a** | All exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression |
| Polyspace Bug Finder | R2024a | **CERT C++: MSC52-CPP** | Checks for missing return statements (rule partially covered) |
| SonarQube C/C++ Plugin | 4.10 | **S935** |  |
| PVS-Studio | 7.33 | **V591** |  |
| RuleChecker | 22.10 | **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.

Automation plays an important role in ensuring security standards and maintaining compliance are achieved during the software development lifecycle. We will integrate automation into the DevOps pipeline and use a wide range of tools throughout the development process. Threat modeling would be best used during the planning stage. Tools like DAST, IAST, and SCA can be used during the creation and verification stage. This can help to find vulnerabilities earlier in the development process. The prevent and detect phase can benefit from RASP, UEBA tools and penetration testing. The response and adaption stage requires security orchestration, WAF’s, and obfuscation. This ensures that a consistent and efficient security enforcement is accomplished.

### 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 |
| --- | --- | --- | --- | --- | --- |
| INT50-CPP | Medium | Unlikely | Medium | P4 | L3 |
| CTR51-CPP | High | Probable | High | P6 | L2 |
| STR51-CPP | High | Likely | Medium | P18 | L1 |
| MEM56-CPP | High | Likely | Medium | P18 | L1 |
| MEM51-CPP | High | Likely | Medium | P1 | L1 |
| DCL03-C | Low | Unlikely | High | P1 | L3 |
| ERR58-CPP | Low | Likely | Low | P9 | L2 |
| FIO39-C | Low | Likely | Medium | P6 | L2 |
| OOP50-CPP | Low | Unlikely | Medium | P2 | L3 |
| MSC52-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 at rest | Encryption at rest is encryption that takes place when data is stored on a device. This works with all data that is stored or archived in a database within Green Pace’s systems. Encryption at rest prevents attacks when unauthorized access occurs. This type of encryption should be used on all sensitive data. |
| Encryption in flight | Encryption in flight secures data while it is in transit to another device or network. This transfer can happen via the internet, intranet, or extranet. This can help prevent unauthorized interception of data. This process should be followed during any transfer of data including but limited to emails, file transfers, and remote access sessions. |
| Encryption in use | Encryption in use is encryption that happens during active access or usage of data in the database. Encryption in use ensures that only usage and access is only conducted by authorized personnel. This protects sensitive data from inside threats and attacks. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication verifies the identity of authorized users that access the same and the data within. |
| Authorization | Authorization ensures that only authorized users access the system. This step also determines access based on granted permissions. |
| Accounting | Accounting monitors usage and history of system usage and access. This ensures that usage is conducted compliant and secure manner. |

**\***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 | 10/13/2024 | Completed Template | Johnny Segura |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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