**Green Pace Developer: Security Policy Guide**



# 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 | Input validation can help avoid most vulnerabilities. It ensures that your program is only taking in input that is expected. It also avoids errors and therefore ensures that a hacker cannot alter your code. |
| 1. Heed Compiler Warnings | Compiler warnings may appear benign but could result in vulnerabilities. It is important to address all warnings by modifying your code. You should also use tools to detect additional security flaws. |
| 1. Architect and Design for Security Policies | Create architecture and design that enforces Security Policies. This can help prevent small issues from cropping up later and becoming security flaws. Be mindful of Security Policies from the beginning and build up your code to adhere to them. |
| 1. Keep It Simple | Keep your code simple. Complicated code can make errors more likely. This can lead to security vulnerabilities. By contrast, keeping you code simple makes it easier to maintain and to fix errors. |
| 1. Default Deny | By default, you should deny access within your code. You should use permissions rather than exclusions to determine who has access to what. Basically, users are denied access by default unless they have the right permissions. Do not allow access by default and exclude those you do not wish to have access. Doing so may lead to oversight. |
| 1. Adhere to the Principle of Least Privilege | Only grant the amount of permission necessary to complete a job. This means do not grant permissions unless they are needed. Higher permissions should only be accessed for the least amount of time for which they are needed. |
| 1. Sanitize Data Sent to Other Systems | Sanitize the data which is passed to subsystems. Otherwise, hackers may be able to access unused functionality using SQL injection. |
| 1. Practice Defense in Depth | Be sure to manage risk with multiple, redundant strategies. This translates to having layers of security tools. Doing this can ensure security even if a layer fails. |
| 1. Use Effective Quality Assurance Techniques | Using good Quality Assurance techniques and testing practices makes it easier to identify and address vulnerabilities. Incorporate testing strategies such as Fuzz testing, penetration testing, and source code audits to ensure the effectiveness of your Quality Assurance. |
| 1. Adopt a Secure Coding Standard | Be sure to keep a secure standard in mind when developing your code. |

### 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] | Programs should only have a single definition for each data type. This is also known as the one definition rule. |

| **Noncompliant Code** |
| --- |
| The noncompliant code has two definitions for a class named S with an integer named a. This code will result in undefined behavior. |
| // a.cpp  struct S {  int a;  };    // b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| This example defines S once in a header file and includes it within a.cpp and b.cpp. I does this instead defining it in both cpp files in different ways like in the noncompliant example. |
| // S.h  struct S {  int a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):** DCL60-CPP. Obey the one-definition rule |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | type-compatibility  definition-duplicate  undefined-extern  undefined-extern-pure-virtual  external-file-spreading  type-file-spreading | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL60 | N/A |
| CodeSonar | 8.1p0 | LANG.STRUCT.DEF.FDH  LANG.STRUCT.DEF.ODH | Function defined in header file  Object defined in header file |
| Helix QAC | 2024.1 | C++1067, C++1509, C++1510 | N/A |
| LDRA tool suite | 9.7.1 | 286 S, 287 S | Fully implemented |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| Polyspace Bug Finder | R2023b | CERT C++: DCL60-CPP | Checks for inline constraints not respected (rule partially covered) |
| RuleChecker | 22.10 | type-compatibility  definition-duplicate  undefined-extern  undefined-extern-pure-virtual  external-file-spreading  type-file-spreading | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do try to read a data value from uninitialized memory. Doing so will trigger undefined behavior. |

| **Noncompliant Code** |
| --- |
| The noncompliant code tries to output the value of int i. While int i is declared, the value is never defined, resulting in undefined behavior. |
| #include <iostream>    void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| The compliant code instead declares int i as 0 before attempting to output it. This ensures that int i is defined and that cout does not attempt to read from uninitialized memory. |
| #include <iostream>    void f() {  int i = 0;  std::cout << i;  } |

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

| **Principles(s):** EXP53-CPP. Do not read uninitialized memory |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | uninitialized read | Partially checked |
| Clang | 3.9 | -Wuninitialized  clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory. |
| CodeSonar | 8.1p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local  Uninitialized variable |
| Helix QAC | 2024.1 | DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978 | N/A |
| Klocwork | 2024.1 | UNINIT.CTOR.MIGHT  UNINIT.CTOR.MUST  UNINIT.HEAP.MIGHT  UNINIT.HEAP.MUST  UNINIT.STACK.ARRAY.MIGHT  UNINIT.STACK.ARRAY.MUST  UNINIT.STACK.ARRAY.PARTIAL.MUST  UNINIT.STACK.MIGHT  UNINIT.STACK.MUST | N/A |
| LDRA tool suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially implemented |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-EXP53-a | Avoid use before initialization |
| Parasoft Insure++ | N/A | N/A | Runtime detection |
| Polyspace Bug Finder | R2023b | CERT C++: EXP53-CPP | Checks for:  Non-initialized variable  Non-initialized pointer  Rule partially covered. |
| PVS-Studio | 7.30 | V546, V573, V614, V670, V679, V730, V788, V1007, V1050 | N/A |
| RuleChecker | 22.10 | uninitialized-read |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not try to create a string from a null pointer. This can later lead to undefined behavior when trying to access the string later. |

| **Noncompliant Code** |
| --- |
| The noncompliant code tries to declare a string tmp from getenv("TMP"). However, this function will return a null pointer on failure which may cause undefined behavior if an error occurs. |
| #include <cstdlib>  #include <string>    void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| The compliant code instead checks to see that getenv("TMP") is not a null pointer before assigning string tmp with it. It also gives string tmp a value of “” in the case that it is. |
| #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):** STR51-CPP. Do not attempt to create a std::string from a null pointer |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Assert\_failure | N/A |
| CodeSonar | 8.1p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Helix QAC | 2024.1 | DF4770, DF4771, DF4772, DF4773, DF4774 | N/A |
| Klocwork | 2024.1 | 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 | N/A |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2023b | 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] | You should take steps to prevent SQL injection. Failure to do so can result in untrusted data maliciously altering an SQL query and causing leaks and modifications. |

| **Noncompliant Code** |
| --- |
| The noncompliant code directly uses user input which can result in injection. |
| sqlite3 \*db;  sqlite3\_stmt \* st;  int id = 0;  string title = "";  string password = "";  cout << "Insert ID:\n";  cin >> id;  cout << "Insert title of password:\n";  cin >> title;  cout << "Insert password:\n";  cin >> password;  string sql = "INSERT INTO passwords (ID,title,password) VALUES (" + id +  ',' + title + ',' + password + ");";  if(sqlite3\_open("pw.db", &db) == SQLITE\_OK){  sqlite3\_prepare( db, sql.c\_str(), -1 &st, NULL);  sqlite3\_step( st );  }  else{  cout << "Failed to connect\n";  }  sqlite3\_finalize(st);  sqlite3\_close(db); |

| **Compliant Code** |
| --- |
| The compliant code instead uses a prepared statement and parameter binding to avoid this issue. |
| sqlite3 \*db;  sqlite3\_stmt \* st;  int id = 0;  string title = "";  string password = "";  string sql = "INSERT INTO passwords (ID,title,password) VALUES (?,?,?)";  if(sqlite3\_open("pw.db", &db) == SQLITE\_OK){  sqlite3\_prepare( db, sql.c\_str(), -1 &st, NULL);  sqlite3\_bind\_int(st, 1, ID);  sqlite3\_bind\_text(st, 2, title.c\_str(), title.length(),  SQLITE\_TRANSIENT);  sqlite3\_bind\_text(st, 3, password.c\_str(), password.length(), SQLITE\_TRANSIENT);  sqlite3\_step( st );  }  else{  cout << "Failed to connect\n";  }  sqlite3\_finalize(st);  sqlite3\_close(db); |

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

| **Principles(s):** IDS00-J. Prevent SQL injection |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors (see Chapter 8) |
| CodeSonar | 8.1p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Findbugs | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Fortify | 1.0 | HTTP\_Response\_Splitting  SQL\_Injection\_\_Persistence  SQL\_Injection | Implemented |
| Klocwork | N/A | SV.DATA.BOUND  SV.DATA.DB  SV.HTTP\_SPLIT  SV.PATH  SV.PATH.INJ  SV.SQL | Implemented |
| Parasoft Jtest | 2023.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| SonarQube | 9.9 | S2077  S3649 | Executing SQL queries is security-sensitive  SQL queries should not be vulnerable to injection attacks |
| SpotBugs | 4.6.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE  SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | You should not try to access memory that has already been freed. Doing so will lead to dangling pointers which can result in vulnerabilities. |

| **Noncompliant Code** |
| --- |
| The noncompliant code attempts to access an instance of S after it has been deleted. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| The compliant code instead deletes the instance of S after it has been accessed. Programmers should ensure that they only free up memory once the program is finished accessing it. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):** MEM50-CPP. Do not access freed memory |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | dangling\_pointer\_use | N/A |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM50 | N/A |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| CodeSonar | 8.1p0 | ALLOC.UAF | Use after free |
| Compass/ROSE | N/A | N/A | N/A |
| Coverity | v7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Helix QAC | 2024.1 | C++4303, C++4304 | N/A |
| Klocwork | 2024.1 | UFM.DEREF.MIGHT  UFM.DEREF.MUST  UFM.FFM.MIGHT  UFM.FFM.MUST  UFM.RETURN.MIGHT  UFM.RETURN.MUST  UFM.USE.MIGHT  UFM.USE.MUST | N/A |
| LDRA tool suite | 9.7.1 | 483 S, 484 S | Partially implemented |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| Parasoft Insure++ | N/A | N/A | Runtime detection |
| Polyspace Bug Finder | R2023b | CERT C++: MEM50-CPP | Checks for:  Pointer access out of bounds  Deallocation of previously deallocated pointer  Use of previously freed pointer  Rule partially covered. |
| PVS-Studio | 7.30 | V586, V774 | N/A |
| Splint | 5.0 | N/A | N/A |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use assertions to find defects that might become vulnerabilities. However, it is to be used for identifying defects and not for checking errors at runtime. It is therefore unsuitable for servers and embedded systems. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, the assert() macro concerning a memory-mapped structure that is needed for correct 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** |
| --- |
| The compliant code instead uses an if statement to check for errors in this scenario. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int))){  std::cout << "Structure must not have any padding";  } |

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

| **Principles(s):** DCL03-C. Use a static assertion to test the value of a constant expression |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 | N/A |
| 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 | N/A | N/A | 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 |
| ECLAIR | 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] | Be sure to handle all exceptions. If an exception handler is not found, then the program may terminate abruptly. |

| **Noncompliant Code** |
| --- |
| The noncompliant code does not catch exceptions thrown. Therefore, no handler is found. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| The compliant code instead uses a try catch statement in main to handle exceptions. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

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

| **Principles(s):** ERR51-CPP. Handle all exceptions |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | main-function-catch-all  early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 | N/A |
| CodeSonar | 8.1p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Helix QAC | 2024.1 | C++4035, C++4036, C++4037 | N/A |
| Klocwork | 2024.1 | MISRA.CATCH.ALL | N/A |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions  Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| Polyspace Bug Finder | R2023b | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| RuleChecker | 22.10 | main-function-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-008-CPP] | Do not slice derived objects. Doing so can result in information loss. |

| **Noncompliant Code** |
| --- |
| The noncompliant code an object derived from the manager class is passed by value to a function accepting an Employee type |
| #include <iostream>  #include <string>    class Employee {  std::string name;    protected:  virtual void print(std::ostream &os) const {  os << "Employee: " << get\_name() << std::endl;  }    public:  Employee(const std::string &name) : name(name) {}  const std::string &get\_name() const { return name; }  friend std::ostream &operator<<(std::ostream &os, const Employee &e) {  e.print(os);  return os;  }  };    class Manager : public Employee {  Employee assistant;    protected:  void print(std::ostream &os) const override {  os << "Manager: " << get\_name() << std::endl;  os << "Assistant: " << std::endl << "\t" << get\_assistant() << std::endl;  }    public:  Manager(const std::string &name, const Employee &assistant) : Employee(name), assistant(assistant) {}  const Employee &get\_assistant() const { return assistant; }  };    void f(Employee e) {  std::cout << e;  }    int main() {  Employee coder("Joe Smith");  Employee typist("Bill Jones");  Manager designer("Jane Doe", typist);    f(coder);  f(typist);  f(designer);  } |

| **Compliant Code** |
| --- |
| The compliant code instead modifies the definition of f() to require raw pointers to the object. |
| // Remainder of code unchanged...    void f(const Employee \*e) {  if (e) {  std::cout << \*e;  }  }    int main() {  Employee coder("Joe Smith");  Employee typist("Bill Jones");  Manager designer("Jane Doe", typist);    f(&coder);  f(&typist);  f(&designer);  } |

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

| **Principles(s):** OOP51-CPP. Do not slice derived objects |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.CAST.OBJSLICE | Object Slicing |
| Helix QAC | 2024.1 | C++3072 | N/A |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-OOP51-a | Avoid slicing function arguments / return value |
| Polyspace Bug Finder | R2023b | CERT C++: OOP51-CPP | Checks for object slicing (rule partially covered) |
| PVS-Studio | 7.30 | V1054 | N/A |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers | [STD-009-CPP] | Do not cast enumeration values that are out of range. |

| **Noncompliant Code** |
| --- |
| The noncompliant code checks if a value falls within a range after casting to the enumeration type. |
| 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 code instead checks if intVar’s a value falls within a range before casting to the enumeration 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):** INT50-CPP. Do not cast to an out-of-range enumeration value |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | cast-integer-to-enum | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 | N/A |
| CodeSonar | 8.1p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2024.1 | C++3013 | N/A |
| 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 | 7.30 | V1016 | N/A |
| RuleChecker | 22.10 | cast-integer-to-enum | Partially checked |
| Polyspace Bug Finder | R2023b | CERT C++: INT50-CPP | Checks for casting to out-of-range enumeration value (rule fully covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [STD-010-CPP] | Ensure that you close files once they are no longer needed to properly deallocate resources and prevent memory leaks. |

| **Noncompliant Code** |
| --- |
| The noncompliant code does not close fstream after it is done being used. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  std::terminate();  } |

| **Compliant Code** |
| --- |
| The compliant code uses file.close(); to close fstream after it is done being used. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  file.close();  if (file.fail()) {  // Handle error  }  std::terminate();  } |

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

| **Principles(s):** FIO51-CPP. Close files when they are no longer needed |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2024.1 | DF4786, DF4787, DF4788 | N/A |
| Klocwork | 2024.1 | RH.LEAK | N/A |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Parasoft Insure++ | N/A | N/A | Runtime detection |
| Polyspace Bug Finder | R2023b | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered) |

### Defense-in-Depth Illustration

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





Automation will be used for the enforcement of and compliance with 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.

As demonstrated in earlier sections of this document, there are numerous tools available for automating each of the security principles outlined. As such, these tools can be incorporated into the earliest part of our DevOps process. It is essential to keep security in mind, from the beginning of a project, to publishing, and beyond. Automation enables us to save time and money in the process of securing our code. Once we have an idea of the product we are producing, something done in the Assess and Plan phase, we can plan the tool which will best suit the automation of our security.

### 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 | High | P3 | L3 |
| STD-002-CPP | High | Probable | Medium | P12 | L1 |
| 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 | Probable | Medium | P4 | L3 |
| STD-008-CPP | Low | Probable | Medium | P4 | L3 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 | L3 |

### 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 they are, how they 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 extends to data that is currently not in transit or in use. Essentially, it applies data was lays dormant in storage or a database. Because it is not actively being used or in motion, it is the least likely to be compromised by hackers. However, it should still remain encrypted in case our storage or databases are somehow accessed by unauthorized parties. |
| Encryption in flight | Encryption in flight refers to the encryption of data that is currently in motion between devices. This could be through collaborative work, data transfer, emails, and more. If data is moving, Encryption in flight applies to it. Due to it being in motion, it has a major risk of being intercepted by unauthorized parties. As such, communication channels must be secured. |
| Encryption in use | Encryption in use refers to the encryption of data that is in active use. Since this data is in active use, it is considered extremely at risk. It may be read by unauthorized parties, updated to perform malicious activities, or more. Things like SQL injection might also happen at this stage. As such, security of this data is a high priority, and steps must be taken to mitigate risk. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication refers to the validation of parties that are accessing data. This may be though usernames, passwords, biometrics, or more. This is an important policy which plays into the other area of a triple A framework. |
| Authorization | Authorization refers to the permissions given to users. It ensures that only certain users can access some data, can make use of specific commands, and more. It is necessary to maintain security as only those with an appropriate level of authorization have access to important roles. A lower end user does not have access to admin privileges for example. |
| Accounting | Accounting refers to the act of keeping an account of who has access to what and when they access it. This allows us to trace what is being done and who has done. Not only does this ensure only necessary tasks are being performed by the proper parties, but it also allows us to trace compromises back to the source. Whether it’s a compromised account or a bad actor within the company. |

**\***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

## 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 | 03/21/2024 | Milestone 1 | Jeremy Depenhart |  |
| 1.2 | 04/12/2024 | Project 1 | Jeremy Depenhart |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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