**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 | To plug as many security gaps as possible, incoming data from both credible and uncredible sources should be verified to ensure no malicious data is entering the system. |
| 1. Heed Compiler Warnings | Try to compile and test the code as much as you can to maintain it so there are no issues.  Even though it could occasionally be redundant, check as much as you can. |
| 1. Architect and Design for Security Policies | You can save a good amount of effort later by either adding protections or correcting vulnerabilities after they are discovered if you design your code as much as practicable with security in mind. |
| 1. Keep It Simple | The security of the code is easier to maintain if the code is as clear and straightforward as possible. It will be simpler to defend if the security component is simple to comprehend. |
| 1. Default Deny | It's a good idea to use denial as a standard since it stops people who shouldn't have access from acquiring access. This ensures those who need access follow the correct procedures to get it. |
| 1. Adhere to the Principle of Least Privilege | Default denial is nicely complemented by the principle of least privilege since it provides a rule that only those who need additional access will go through the correct process to get it. Only the access that is necessary will be granted during these steps of getting access, lowering potential danger by making less information generally available. |
| 1. Sanitize Data Sent to Other Systems | Any data that is transmitted to other systems must be verified to ensure that any possible vulnerabilities have been addressed. This type of knowledge might be used in code to create attacks, which, depending on the situation, could lead to a range of terrible outcomes. |
| 1. Practice Defense in Depth | A system can avoid being infiltrated and wreaking havoc by combining layers of overlapping and shifting security. This is true for most systems. |
| 1. Use Effective Quality Assurance Techniques | Every process includes a sizeable amount of quality assurance, which is always undesirable by nature because it only serves to spot issues with any method or design while still being crucial to the result.  It is better for a team member to find any issues or vulnerabilities than for a client or possibly a malicious operator to do so, as the latter could have far more detrimental effects. |
| 1. Adopt a Secure Coding Standard | A secure coding standard complements other ideas like creating your code with security policies in mind and using efficient quality assurance procedures perfectly. Keep these ideas in mind as frequently as you can to avoid problems that will probably cost twice as much as they would have up front and to save time. |

### 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] | Avoid casting to an enumeration value that is out of range. |

| **Noncompliant Code** |
| --- |
| Inspects a value to see if it falls inside the range that is acceptable for enumeration values. The type might not be able to represent the given integer value after being cast. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| To ensure that the conversion won't produce an undefined value, the compliant solution verifies the value the enumeration type represents before converting. It thus restricts the translated value to a particular class of enumerator. |
| 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):** Unknown values may result in a buffer overflow, which could give an attacker the ability to run any code. Enumerators are infrequently used for indexing into arrays or other sorts of pointer arithmetic, so it is more likely that this scenario will result in data integrity issues than uncontrolled 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 | Only values that match the enumerators of the enumeration are allowed in an expression with enum as its underlying type. |
| PVS – Studio | 7.07 | V1016 | [Insert text.] |
| Parasoft C/C++  test | 2020.2 | CERT\_CPP – INT50-a | [Insert text.] |
| PRQA QA- C++ | 4.4 | 3013 | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002- CPP] | It can be problematic to define a reserved identity incorrectly because it won't be truly protected. |

| **Noncompliant Code** |
| --- |
| Standards for naming are not met, which results in inconsistent behavior. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_  // Contents of <my\_header.h>  #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| The problem is avoided by getting rid of the leading and trailing underscores.  #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H  // Contents of <my\_header.h>  #endif // MY\_HEADER\_H |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P18 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS – Studio | 7.07] | V783 | [Insert text.] |
| Parasoft C/C++  test | 2020.2 | CERT\_CPP-CTR51-a | When iterating over a container, do not change it. |
| Astree | 20.10 | Overflow\_unpon\_dereference | [Insert text.] |
| Helix QAC | [Insert text.] | 2021.1 | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003- CPP] | Make no effort to convert a null pointer into a std::string |

| **Noncompliant Code** |
| --- |
| This noncompliant code example produces a std::string object from the output of a std::getenv call (). If the environment variable is missing, this procedure could behave unexpectedly because of std::  If getenv() fails, a null pointer is returned (or some other error occurs) |
| #include <cstdlib>  #include <string>  void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| The results of the std::getenv() call are tested for null in this compatible approach before the std::string object is built. |
| #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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ParasoftC/C++  test | 2020.2 | CERT\_CPP-STR51-a | Avoid dereferencing null pointers |
| Astree | 20.10 | Assert\_failure | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004- CPP | Avoid making pronouncements that are syntactically confusing.  Write code that has only one possible interpretation. |

| **Noncompliant Code** |
| --- |
| This parameter can be regarded as defining an object named m and default building it or as creating an anonymous object and using its single argument converting constructor. |
| #include <mutex>  static std::mutex m;  static int shared\_resource;  void increment\_by\_42() {  std::unique\_lock<std::mutex>(m);  shared\_resource += 42;  } |

| **Compliant Code** |
| --- |
| The correct conversion constructor is invoked once the lock is given an identification. |
| #include <mutex>  static std::mutex m;  static int shared\_resource;  void increment\_by\_42() {  std::unique\_lock<std::mutex> lock(m);  shared\_resource += 42;  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug  Finder | R2020a | CERT C++: DCL53-CPP | Determines whether any declarations between function and object and unnamed object or function parameter |
| LDRA tool suite | 9.7.1 | 296 S | Partially carried out |
| Parasoft C/C+ +test | 2020.2 | CERT\_CPP-DCL53-a  CERT\_CPP-DCL53-b | An identifier specified in a global scope or namespace cannot be hidden by one that is declared in a local scope or function prototype. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005- CPP] | Reallocate dynamically assigned resources effectively. |

| **Noncompliant Code** |
| --- |
| The local variable space is provided as the expression to the placement new operator in this noncompliant code. The resultant reference is sent to::operator delete() after that call, which results in undefined behavior because::operator delete() tries to free memory that wasn't returned by::operator new () |
| #include <iostream>  struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;  // ...  delete s1;  } |

| **Compliant Code** |
| --- |
| The destructor of s1 is called explicitly in this compatible workaround, which eliminates the call to::operator delete().  Invoking a destructor explicitly in this situation is one of the rare occasions when it is necessary. |
| #include <iostream>  struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C+ +test | 2020.2 | CERT\_CPP-DCL54-a | Offer new and always delete simultaneously. |
| Clang | 3.9 | misc-new-delete-overloads | Checked with clang tidy. |
| Astrée | 20.10 | new-delete-pairwise | Partially checked |
| Polyspace Bug Finder | R2020a | CERT C++: DCL54-CPP | Verifies whether the overloaded operator new and the operator delete are inconsistent. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006- CPP] | Reduce information leakage when moving a class object across a trust barrier.  The data passing must be vetted before it can cause issues. |

| **Noncompliant Code** |
| --- |
| In this case, the data may contain sensitive information when it is conveyed, regardless of the method. |
| #include <cstddef>  struct test {  int a;  char b;  int c;  };  // Safely copy bytes to user space  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);  void do\_stuff(void \*usr\_buf) {  test arg{1, 2, 3};  copy\_to\_user(usr\_buf, &arg, sizeof(arg));  } |

| **Compliant Code** |
| --- |
| By serializing the structural data before duplicating it, this avoids these kinds of problems. |
| #include <cstddef>  #include <cstring>  struct test {  int a;  char b;  int c;  };  // Safely copy bytes to user space.  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);  void do\_stuff(void \*usr\_buf) {  test arg{1, 2, 3};  // May be larger than strictly needed.  unsigned char buf[sizeof(arg)];  std::size\_t offset = 0; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C+ +test | 2020.2 | CERT\_CPP-DCL55-a | A pointer to a structure shouldn't be passed to a function that can copy data to the user space. |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-DCL55 | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007- CPP] | When main() begins to run, handle all exceptions that have been thrown. |

| **Noncompliant Code** |
| --- |
| When globalS is created in this noncompliant example during program launch, the constructor for S may throw an exception that is not caught. |
| struct S {  S() noexcept(false);  };  static S globalS; |

| **Compliant Code** |
| --- |
| The constructor for S will be run the first time the method globalS() is called rather than at program startup, which allows for any exceptions thrown during object building to be caught with this compatible solution that turns globalS into a local variable with static storage lifetime. With this approach, the programmer must modify the source code so that earlier calls to globalS are replaced by function calls 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C+ +test | 2020.2 | CERT\_CPP-DCL56-a | By substituting local static objects with non-local static objects, problems with initialization order across translation units can be avoided. |
| LDRA tool suite | 9.7.1 | 6 D | Additional Enforcement |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Destructor Methods | [STD-008- CPP] | Avoid allowing exceptions to bypass deallocation or destructor methods. |

| **Noncompliant Code** |
| --- |
| The class destructor may throw an exception and cause undefinable behavior. |
| #include <stdexcept>  class S {  bool has\_error() const;  public:  ~S() noexcept(false) {  // Normal processing  if (has\_error()) {  throw std::logic\_error("Something bad");  }  }  }; |

| **Compliant Code** |
| --- |
| Any exceptions will be caught by this and destroyed as well. |
| class SomeClass {  Bad bad\_member;  public:  ~SomeClass()  try {  // ...  } catch(...) {  // Catch exceptions that nonconforming destructors throw.  // member objects or base class subobjects.  // Leaving a destructor function-try-block early results in  // Rather than an explicit rethrow of the captured exception  // return statement will prevent that from happening.  return;  }  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C+ +test | 2020.2 | CERT\_CPP-DCL57-a  CERT\_CPP-DCL57-b | Destructors, deallocations, and swaps shouldn't be allowed to throw exceptions.  Consistently look for exceptions |
| Astrée | 20.10 | destructor-without-noexcept delete-without-noexcep | Fully examined |
| LDRA tool suite | 9.7.1 | 453 S | Partially carried out |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programing | [STD-009- CPP] | Do not use constructors or destructors to call virtual functions. |

| **Noncompliant Code** |
| --- |
| The base class in this non-compliant code example makes calls to virtual functions from the constructor and destructor to take and release an object's resources. The B::B() constructor, however, uses B::seize() instead of D::seize (). The B::B() destructor uses B::release() instead of D::release (). |
| struct B {  B() { seize(); }  virtual ~B() { release(); }  protected:  virtual void seize();  virtual void release();  };  struct D : B {  virtual ~D() = default;  protected:  void seize() override {  B::seize();  // Get derived resources...  }  void release() override {  // Release derived resources...  B::release();  }  } |

| **Compliant Code** |
| --- |
| Instead of calling a virtual function, the constructors and destructors in this complying approach call a nonvirtual, private member function (suffixed with mine). As a result, each class oversees capturing and releasing its own resources. |
| class B { |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C+ +test | 2020.2 | CERT\_CPP-DCL58-a | Do not alter the "std" and "posix" standard namespaces. |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-DCL58 | [Insert text.] |
| Polyspace Bug Finder | R2020a | CERT C++: DCL58-CPP | Checks for changes to the default namespaces |
| PRQA QA-C++ | 4.4 | 4032, 4035, 4631 | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Unnamed Namespace | [STD-010- CPP | In a header file, avoid defining an unnamed namespace.  It can be problematic to utilize an unnamed namespace in a header file. |

| **Noncompliant Code** |
| --- |
| Each translation unit uses a unique instance of the variable because it is defined in an unnamed namespace. |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE  namespace {  int v;  }  #endif // A\_HEADER\_FILE  // a.cpp  #include "a.h"  #include <iostream>  void f() {  std::cout << "f(): " << v << std::endl;  v = 42;  // ...  }  // b.cpp  #include "a.h"  #include <iostream>  void g() {  std::cout << "g(): " << v << std::endl;  v = 100;  }  int main() {  extern void f();  f(); // Prints v, sets it to 42  g(); // Prints v, sets it to 100  f();  g();  } |

| **Compliant Code** |
| --- |
| One translation unit defines the variable, which is shared by all and produces the desired result. |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE  extern int v;  #endif // A\_HEADER\_FILE  // a.cpp  #include "a.h"  #include <iostream>  int v; // Definition of global variable v  void f() {  std::cout << "f(): " << v << std::endl;  v = 42;  // ...  }  // b.cpp  #include "a.h"  #include <iostream>  void g() {  std::cout << "g(): " << v << std::endl;  v = 100;  }  int main() {  extern void f();  f(); // Prints v, sets it to 42  g(); // Prints v, sets it to 100  f(); // Prints v, sets it back to 42  g(); // Prints v, sets it back to 100  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool suite | 9.7.1 | 286 S, 512 S | Entirely carried out |
| Clang | 3.9 | cert-dcl59-cpp | Checked by clang-tidy |
| Astrée | 20.10 | unnamed-namespace-header | Fully examined |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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.

[Insert your written explanations here.]

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| [STD-001- CPP] | Medium | Unlikely | Medium | P4 LOW | L3 |
| [STD-002- CPP] | High | Probable | High | P18 HIGH | L2 |
| [STD-003- CPP] | High | Likely | High | P1 LOW | L1 |
| [STD-004- CPP] | Low | Unlikely | Medium | P2 LOW | L3 |
| [STD-005- CPP] | Low | Probable | Low | P6 LOW | L2 |
| [STD-006- CPP] | Low | Unlikely | High | P1 HIGH | L3 |
| [STD-007- CPP] | Low | Unlikely | Medium | P2 MEDIUM | L3 |
| [STD-008- CPP] | Low | Likely | Medium | P6 MEDIUM | L2 |
| [STD-009- CPP] | High | Unlikely | Medium | P6 HIGH | L2 |
| [STD-010- CPP] | Medium | Unlikely | Medium | P4 HIGH | L3 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 | To prevent hackers from accessing the unencrypted data, it is an encryption that ensures the data is encrypted while it is on disk. To transform the customer's critical information into other types of data, encryption in rest is used. |
| Encryption at flight | Data files that are accessed are encrypted during the procedure using actual encryption techniques. Applications employ the technique of data encryption to safeguard data while it is being shared. |
| Encryption in use | The use of encryption has proven to be a successful strategy for protecting sensitive data over the long run. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This method aids in confirming the user's identity.  To prevent unauthorized access, the policy is applied in computer systems and other data storage devices. |
| Authorization | The user can now access the computer to do a specified task thanks to this method.  For access control, authorization is used, and authentication comes before authorization. |
| Accounting | Accounting describes the computerized software applications housed on a company computer or network server. User logins are essential for ensuring the privacy of the accounting data used by 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

### 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

Coding Standards:

1. STD-001-CPP

Principles:

Validate input data

Relationship:

The idea is related to coding standards since it is feasible for unstructured values to trigger a buffer

overflow, which would allow an attacker to execute arbitrary code.

Coding Standards:

1. STD-002-CPP

Principles:

Default Deny

Relationship:

Any arbitrary code could be executed by dereferencing a null pointer. Since it is impossible to run

arbitrary code on the platforms where the provided severity is visible, the actual severity is

minimal.

Coding Standards:

1. STD-003-CPP

Principles:

Simple is Best

Relationship:

A crucial diagnostic technique for identifying and removing software flaws that could lead to

vulnerabilities at build time are the static assertion.

Coding Standards:

1. STD-004-CPP

Principles:

Be aware of Compiler Warnings

Relationship:

In contrast to the null code's default value, the complying solution effectively initializes the code to

provide a valid pointer to item location.

Coding Standards:

1. STD-005-CPP

Principles:

Design and Architect for Security Policies

Relationship:

The burden of risk is primarily on the caller of the allocation but in some cases, dereferencing a null

pointer can lead to the execution of arbitrary code.

Coding Standards:

1. STD-006-CPP

Principles:

Utilize efficient quality assurance methods

Relationship:

The assertions' side effects, which depend on whether they are enabled or disabled, affect how the

program behaves; hence the principle applies to the code.

Coding Standards:

1. STD-007-CPP

Principles:

Embrace the Secure

Relationship:

The idea is applicable to the code because it encourages attempts to improve destructor and de-

allocation algorithm exceptions that could lead to attacks and characteristics that are difficult to

describe.

Coding Standards:

1. STD-008-CPP

Principles:

Intensive Defense Practices

Relationship:

Since attempts to destroy polymorphic objects without virtual destructors resulted in undeclared

behavior, the principle is related to the coding standard.

Coding Standards:

1. STD-009-CPP

Principles:

Ensure Data Sent to Other Systems is Clean

Relationship:

Since inputting and outputting similar files can make it easier for an attacker to take down the

system, the principle applies to good coding practices.

Coding Standards:

1. STD-010-CPP

Principles:

Observe the least-privilege Principle.

Relationship:

The liveness property is provided by the wake of all threads since each thread can complete its

condition, forecast test, and one will pass to proceed with execution.

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 | 11/27/2022 | Initial Template | Kao Chao |  |
| 2.0 | 12/2/2022 | Adding coding standards | Kao Chao | [Insert text.] |
| [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 |