**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 try to get rid of as many security holes as possible, we validate data from both known and untrusted sources to make sure no malicious data is getting into the system. |
| 1. Heed Compiler Warnings | Trying to make the code as tight as possible by compiling and testing it as much as possible. A warning might just be an annoyance, but it could also be something important, so check as many as you can just in case. |
| 1. Architect and Design for Security Policies | If you write your code with security in mind as much as possible while still meeting the standard for what the code is meant to do in the end, you can save a lot of time later either by building in protections or fixing flaws after they are found. |
| 1. Keep It Simple | Keeping the code as clean and easy as possible makes everything easier, including making sure the code is secure. This is one of the most important rules of life, and it's even more important when it comes to security, since something is easier to protect if it's easy to understand. |
| 1. Default Deny | Denial as a standard is a good idea because it keeps people from getting access to something they shouldn't, while also making sure that people who need access go through the right routes to get it. |
| 1. Adhere to the Principle of Least Privilege | Principle of least privilege works well with default denial because it makes it clear that only people who need more access will go through the right routes to get it. During these steps to get access, it will be made clear that only the access that is really needed will be given. This will lower the risk because there will be less information available in general. |
| 1. Sanitize Data Sent to Other Systems | It is very important to make sure that any data sent to other systems doesn't contain any possible security flaws or private information. Sending code with this kind of information could lead to exploits, which could have any number of bad results, based on the situation. |
| 1. Practice Defense in Depth | Most systems need more than one layer of protection. Using layers of defense that are multiple and different can help protect a system that would otherwise be broken into and cause chaos. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance is a big part of every process. It's not meant to be popular because it's only purpose is to find problems with a process or plan, but it's still important to the end result. It's better for someone on the team to find any problems or weaknesses than for a customer or someone with bad intentions to do so, which can have much worse results. |
| 1. Adopt a Secure Coding Standard | Having a secure coding standard goes well with other ideas, like making sure your code is designed with security policies in mind and using good quality assurance methods. Keeping these ideas in mind as much as possible is a great way to save time and avoid problems that will likely cost twice as much in the long run as they would have up front. |

### 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] | C-style variadic functions may be vulnerable because they don't check parameters. |

| **Noncompliant Code** |
| --- |
| This function reads values until it finds 0 or two arguments, which can cause problems. |
| #include <cstdarg>  int add(int first, int second, ...) {  int r = first + second;  va\_list va;  va\_start(va, second);  while (int v = va\_arg(va, int)) {  r += v;  }  va\_end(va);  return r;  } |

| **Compliant Code** |
| --- |
| Its built-in add statement prevents the aforesaid difficulties. |
| #include <type\_traits>  template <typename Arg, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* =  nullptr>  int add(Arg f, Arg s) { return f + s; }  template <typename Arg, typename... Ts, typename  std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  int add(Arg f, Ts... rest) {  return f + add(rest...);  } |

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

| **Principles(s):** Validate Input Data  Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Function-ellipsis | Fully Checked |
| Axivion Bahaus Suite | 7.2.0 | CertC++-DCL50 | [Insert text.] |
| Clang | 3.9 | cert-dcl50-cpp | Checked by clang\_tidy |
| Code Sonar | 7.3p0 | LANG.STRUCT.ELLIPSIS | Ellipsis |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Incorrectly defining a reserved identity can cause problems. |

| **Noncompliant Code** |
| --- |
| Unmet naming requirements produce unclear behavior. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_  // Contents of <my\_header.h>  #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| Removing trailing and leading underscores solves the problem. |
| #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):** Compiler Warnings  Security Policies  Clean code  QA techniques  Secure code |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Not likely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Reserved-identifier | Partial Check |
| Clang | 3.9 | -WRESERVED-ID-MACRO-Wuser-defined-literals | The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverything. This flag does not  catch all instances of this rule, such as redefining reserved names. |
| Axivion Bahaus Suite | 7.2.0 | CertC++-DCL51 | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Reference types should never be const or volatile. cv-qualifying a reference type is undefined. Without a fatal diagnostic, a compiler may give unexpected results. |

| **Noncompliant Code** |
| --- |
| Instead, a const-qualified char reference is created. |
| #include <iostream>  void f(char c) {  char &const p = c;  p = 'p';  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| Avoid this by removing the const qualifier. |
| #include <iostream>  void f(char c) {  char &p = c;  p = 'p';  std::cout << c << std::endl;  } |

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

| **Principles(s):** Compiler Warnings  Security Policies  Clean code  QA Techniques  Secure Coding |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Not Likely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL52 | [Insert text.] |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-DCL52-a | Never qualify a reference type with 'const' or 'volatile' |
| Polyspace Bug Finder | R2023a | CERT C++: DCL52-CPP | Checks for:  const-qualified reference types  Modification of const-qualified reference types  Rule fully covered. |
| SonarQube C/C++ Plugin | 4.10 | S3708 | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Declare syntactically. Write unambiguous code. |

| **Noncompliant Code** |
| --- |
| This parameter can define an anonymous object and call its single-argument converting constructor or declare an object named m and default construct it. |
| #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** |
| --- |
| A lock identifier and converter constructor are called. |
| #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):** Validate input  Security Policies  Clean Code  QA Techniques  Secure Coding |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool Suite | 9.71 | 296 S | Partially implemented |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-DCL53-a  CERT\_CPP-DCL53-b  CERT\_CPP-DCL53-c | Parameter names in function declarations should not be enclosed in parentheses  Local variable names in variable declarations should not be enclosed in parentheses  Avoid function declarations that are syntactically ambiguous |
| Polyspace Bug Finder | R2023a | CERT C++: DCL53-CPP | Checks for declarations that can be confused between:  Function and object declaration  Unnamed object or function parameter declaration  Rule fully covered. |
| Clang | 3.9 | -Wvexing-parse | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Overload allocation and deallocation together.  If not, undefined behavior may result. |

| **Noncompliant Code** |
| --- |
| Globally overloaded allocation has no deallocation method. |
| #include <Windows.h>  #include <new>  void \*operator new(std::size\_t size) noexcept(false) {  static HANDLE h = ::HeapCreate(0, 0, 0); // Private, expandable heap.  if (h) {  return ::HeapAlloc(h, 0, size);  }  throw std::bad\_alloc();  } |

| **Compliant Code** |
| --- |
| Declaring deallocation prevents overload. |
| #include <Windows.h>  #include <new>  class HeapAllocator {  static HANDLE h;  static bool init;  public:  static void \*alloc(std::size\_t size) noexcept(false) {  if (!init) {  h = ::HeapCreate(0, 0, 0); // Private, expandable heap.  init = true;  }    if (h) {  return ::HeapAlloc(h, 0, size);  }  throw std::bad\_alloc();  }  static void dealloc(void \*ptr) noexcept {  if (h) {  (void)::HeapFree(h, 0, ptr);  }  }  };  HANDLE HeapAllocator::h = nullptr;  bool HeapAllocator::init = false;  void \*operator new(std::size\_t size) noexcept(false) {  return HeapAllocator::alloc(size);  }  void operator delete(void \*ptr) noexcept {  return HeapAllocator::dealloc(ptr);  } |

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

| **Principles(s):** Validate Input  Security Policies  Clean Code  QA Techniques  Secure Coding |
| --- |
|  |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | new-delete-pairwise | Partially checked |
| Clang | 3.9 | misc-new-delete-overloads | Checked with clang-tidy. |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-DCL54-a | Always provide new and delete together |
| Polyspace Bug Finder | R2023a | CERT C++: DCL54-CPP | Checks for mismatch between overloaded operator new and operator delete (rule fully covered) |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Pass class objects across trust boundaries without leaking information. Before causing complications, data passing must be checked. |

| **Noncompliant Code** |
| --- |
| This example may involve sensitive data being sent. |
| #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** |
| --- |
| Serializing structural data before copying should prevent these issues. |
| #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;  std::memcpy(buf + offset, &arg.a, sizeof(arg.a));  offset += sizeof(arg.a);  std::memcpy(buf + offset, &arg.b, sizeof(arg.b));  offset += sizeof(arg.b);  std::memcpy(buf + offset, &arg.c, sizeof(arg.c));  offset += sizeof(arg.c);  copy\_to\_user(usr\_buf, buf, offset /\* size of info copied \*/);  } |

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

| **Principles(s):** Validate Input  Security Policies  Data Transfer  Clean Code  QA Techniques  Secure Coding |
| --- |

**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++-DCL55 | [Insert text.] |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-DCL55-a | A pointer to a structure should not be passed to a function that can copy data to the user space |
| [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] | Avoid static object initialization cycles. Reentering a function during static object initialization is undefined. |

| **Noncompliant Code** |
| --- |
| This caches factorial function, but the static array cache setup includes recursion, causing undefined behavior. |
| #include <stdexcept>  int fact(int i) noexcept(false) {  if (i < 0) {  // Negative factorials are undefined.  Throw std::domain\_error("i must be >= 0");  }  static const int cache[] = {  fact(0), fact(1), fact(2), fact(3), fact(4), fact(5),  fact(6), fact(7), fact(8), fact(9), fact(10), fact(11),  fact(12), fact(13), fact(14), fact(15), fact(16)  };  if (i < (sizeof(cache) / sizeof(int))) {  return cache[i];  }  return i > 0 ? i \* fact(i - 1) : 1;  } |

| **Compliant Code** |
| --- |
| This avoids the problem-causing static cache. |
| include <stdexcept>  int fact(int i) noexcept(false) {  if (i < 0) {  // Negative factorials are undefined.  throw std::domain\_error("i must be >= 0");  }  // Use the lazy-initialized cache.  static int cache[17];  if (i < (sizeof(cache) / sizeof(int))) {  if (0 == cache[i]) {  cache[i] = i > 0 ? i \* fact(i - 1) : 1;  }  return cache[i];  }  return i > 0 ? i \* fact(i - 1) : 1;  } |

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

| **Principles(s):** Security Policies  Clean Code  QA Techniques  Secure Coding |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool Suite | 9.7.1 | 6D | Enhanced Enforcement |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-DCL56-a | Avoid initialization order problems across translation units by replacing non-local static objects with local static objects |
| [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** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Avoid exceptions from destructors and deallocation functions. |

| **Noncompliant Code** |
| --- |
| Class destructor exceptions may cause ambiguous 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** |
| --- |
| This eliminates exceptions. |
| class SomeClass {  Bad bad\_member;  public:  ~SomeClass()  try {  // ...  } catch(...) {  // Catch exceptions from noncompliant destructors of  // objects or class sub objects.  // NOTE: Flowing off the end of destructor function block causes  // the exception to be implicitly rethrown, but an explicit  // return will prevent exception from happening.  return;  }  }; |

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

| **Principles(s):** Secure Policies  Clean Code  QA Techniques  Secure COding |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | destructor-without-noexcept  delete-without-noexcept | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL57 | [Insert text.] |
| LDRA tool Suite | 9.7.1 | 453 S | Partially implemented |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-DCL57-a  CERT\_CPP-DCL57-b | Never allow an exception to be thrown from a destructor, deallocation, and swap  Always catch exceptions |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | Avoid changing standard namespaces. When misused, new namespace declarations might produce undefined behavior. |

| **Noncompliant Code** |
| --- |
| x's namespace addition causes ambiguous behavior. |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| Placing without a reserved name does not create unclear behavior. |
| namespace nonstd {  int x;  } |

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

| **Principles(s):** Secure Policies  Clean Code  QA Techniques  Secure Coding |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL58 | [Insert text.] |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-DCL58-a | Do not modify the standard namespaces 'std' and 'posix' |
| Polyspace Bug Finder | R2023a | CERT C++: DCL58-CPP | Checks for modification of standard namespaces (rule fully covered) |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Header files should not define unnamed namespaces. Unnamed header namespaces might cause problems. |

| **Noncompliant Code** |
| --- |
| Each translation unit operates on its own instance because the variable 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** |
| --- |
| The variable is visible to all and produces the anticipated output. |
| // 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):** QA Techniques  Secure Coding |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | unnamed-namespace-header | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL59 | [Insert text.] |
| Clang | 3.9 | cert-dcl59-cpp | Checked by clang-tidy |
| LDRA tool Suite | 9.7.1 | 286 S, 512 S | Fully implemented |

### 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 | High | Probable | Medium | P12 | L1 |
| STD-002-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-003-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-004-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-005-CPP | Low | Probable | Low | P6 | L2 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CLG | Low | Unlikely | Medium | P2 | L3 |
| STD-008-CPP | Low | likely | Medium | P6 | L2 |
| STD-009-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-0010-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 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 | Encrypting data while it's saved keeps it safe. This could include things like hard drives, phones, computers, and the cloud. This data can be kept safe by using encryption tools, disk protection, and security for computers and mobile devices. |
| Encryption at flight | Encryption in flight is used to keep data safe while it is moving. This can happen between two devices in the same network or between devices in different networks. This can be kept safe with things like encrypting emails, DLP solutions, and firewalls and other strong network security features. It is also important to think about where the data may be going and how safe that road is. |
| Encryption in use | Encryption in use guards data that is being made, changed, or used in some other way. This data can be kept safe by making sure that data control and security are in place before it is used and are there in the first place. Managing who has access to this data and what they are will also lower the risk to it. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the act of making sure someone is who they say they are. This can mean a lot of different things, but static passwords, one-time passwords, certifications, and biometric identities are common examples. These kinds of ID work to make sure that a person is who they say they are. |
| Authorization | Authorization is an important part of information and computer protection because it says what a user can do and how they can do it. Authentication is the process of making sure someone is who they say they are. Authorization is the process of deciding what a user can and can't access in the first place. This limits security holes that could happen when someone interacts with private data they don't need to or when they have permissions to do so. |
| Accounting | Accounting is the process of keeping track of what people do while using a system. It shows timestamps, available resources, and information about data transfers. This is helpful because it leaves a "breadcrumb trail" of the user's actions and can also be used for forensic analysis and research if needed. |

**\***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 |  |
| [Insert text.] | 05/14/2023 | M3 Milestone | Larry Williams Jr | [Insert text.] |
| [Insert text.] | 06/09/2023 | M6 Milestone | Larry Williams Jr | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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