**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external sources. |
| 1. Heed Compiler Warnings | Compile code using highest warning level available for your compiler and eliminate warnings. Use static and dynamic analysis tools to detect and eliminate other security flaws. |
| 1. Architect and Design for Security Policies | Design your code with security in mind while still meeting the standard of what the code is set out to do you can save time later by either building safeguards or eliminating vulnerabilities after they are found. |
| 1. Keep It Simple | Keep your code clean and concise (simple). Keeping things simple is a good trait for all of life. It makes it easier to secure things when it is simple enough to understand it’s purpose. |
| 1. Default Deny | Denial as a standard is good for preventing anyone who should not have access to something from gaining the access to it. At the same time ensuring those who need access go through the proper channels for access. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with he least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems such as command shells, rational databases, and commercial off the shelf components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injections attacks. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies so you have multiple layers of protection in case one is broken there can be another to takes its place. Multiple layers of security can prevent sensitive information from getting out into the hands of the wrong person/entity. |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques can be effective in identifying vulnerabilities. Independent security reviews can lead to a more secure system. External reviewers also bring in a “fresh set of eyes” which can sometimes be good for finding other vulnerabilities not caught by you. |
| 1. Adopt a Secure Coding Standard | Develop or apply a secure coding standard for your development language you are using. Use said secure coding for the platform to ensure it is easy to follow yet secure. |

### 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] | Defining C-style variadic functions can lead to vulnerabilities because they do not check arguments being passed. |

| **Noncompliant Code** |
| --- |
| This function is designed to read the values until 0 value is found or if it is not found after two arguments it can cause issues in the program. |
| #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** |
| --- |
| This function has a built in add statement that helps prevent the issue above |
| #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):** The principle seen here would be validating input data, ensuring proper inputs. The KISS principle can also be applied here and to all code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| high | probable | medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astee | 20.10 | Function-ellipsis | Fully checked |
| clang | 3.9 | Cert-dc150-cpp | Checked by clang-tidy- |
| CodeSonar | 5.4p0 | LANG.STRUCT.ELLIPSIS | Ellipsis |
| Axivion Bauhaus Suite | [Insert text.] | CertC++DCL50 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Defining a reserved identifier incorrectly can cause issues as it will not be reserves |

| **Noncompliant Code** |
| --- |
| Naming standards not met causes errors |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_  //contents of <my\_header.h>  #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| Removing the leading and trailing underscores resolves the issue as they are not part of conventional naming |
| #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):** The main principle seen here would be to pay attention to compiler warnings to avoid any unnecessary issues in your code. Using the KISS method applies to this code as well. Using a secure and well understood coding standard is important in avoiding the issues above. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Reserved-identifier | Partially checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC++DCL51 |  |
| Clang | 3.9 | Wreserved-id-macro | The -Wreserved-id-macro flag is not enabled by edault or with -wall but is enabled with -Weverything. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Never qualify a reference type with const or volatile.cv-qualifying a reference type will result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| A const-qualified reference to a char is formed instead of a reference to a const-qualified char |
| #include <iostream>  void f(char c) {  char &const p = c;  p = ‘p’;  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| Remove the const qualifier to prevent such an error |
| #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):** This principle also includes paying attention to compiler warnings and the KISS method. Also adopting a secure coding standard like the last block. Effective testing is also important for all code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | CertC++DCL52 |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-CDL52-a | Never qualify a reference type with ‘const’ or ‘volatile’ |
| Polyspace Bug Finder | R2020a | CERT C++: DCL52-CPP | Checks for const-qualified reference types modification of const qualified reference types rule fully covered. |
| PRQA QA-C++ | 4.4 | 0014 |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Do not write syntactically ambiguous declarations. Write code that can only be understood one way. |

| **Noncompliant Code** |
| --- |
| The argument can be taken to declare an anonymous object and calling its single-argument converting constructor or interpreted as declaring an object named m and default constructing it instead. |
| #include <mutex>  static std:: mutex m;  static int shared\_resource;  void increment-by\_42() {  std:: unique\_lock <std:: mutex> (m);  shared\_resources += 42;  } |

| **Compliant Code** |
| --- |
| The lock is given an identifier and proper converting constructor is 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):** This one also contains the validating input principle, KISS method, proper testing, and secure coding. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| low | Not likely | medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool Suite | 9.7.1 | 296 S | Partially implemented |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-DCL53-a  CERT\_CPP-DCL53-b | Always declare functions at file scope |
| Polyspace Bug Finder | R2020a | CERT C++: DCL53-CPP | Check for declarations that can be confused between function and object declaration, unnamed object or function parameter declaration. Rule fully covered. |
| PRQA QA-C++ | 4.4 | 2502, 2510 |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Overload allocation and deallocation functions as a pair in the same scope. Failure to do so will likely cause errors. |

| **Noncompliant Code** |
| --- |
| The allocation is overloaded at a global scale but there is no deallocation function declared. |
| #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();  }  //no corresponding global delete operator defined |

| **Compliant Code** |
| --- |
| The deallocation is declared which should prevent 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);  }  } |

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

| **Principles(s):** The principles seen here contain secure coding, validating input, and of course the KISS method. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| low | probable | low | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | New-delete-pairwise | Partially checked |
| Clang | 3.9 | Misc-new-delete-overloads | Checked with clang-tidy |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-DCL54-a | Always provide new and delete together |
| Polyspace Bug Finder | R2020a | CERT C++:D L54-CPP | Checks for mismatch between overload operator new and operator delete |

#### 

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Avoid information leakage when passing a class object across a trust boundary. The data passing needs to be verified before it can cause issues. |

| **Noncompliant Code** |
| --- |
| The data when transferred regardless of means may contain sensitive information in this example. |
| #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** |
| --- |
| This serializes the structure data before copying it which should prevent the issue above. |
| #include <cstddef>  #include <cststring>  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};  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 information copied \*/ );  } |

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

| **Principles(s):** The principles seen here are the same as the others except this one also contains the principle for cleaning up data sent to other systems. This is important to ensure code works properly and does not clog up the system or leak information. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Not likely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axicion Bauhaus Suite | 6.9.0 | CertC++DCL55 |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-DCL55-a | A pointer to a structure should not be passed to a function that can copy data to user space. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Avoid cycles during initialization of static objects, if a function is reentered during initialization of static object it will cause errors. |

| **Noncompliant Code** |
| --- |
| Attempts to implement factorial function utilizing caching but the initialization of the static array cache involves recursion creating errors. |
| #include <stdexcept>  int fact(int i) noexcept(false) {  if (i < 0) {  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** |
| --- |
| Does not utilize static cache |
| #include <stdexcept>  int fact(int i) noexcept(false) {  if (i < 0) {  throw std::domain\_error(“i must be >= 0”);  }  static int cache[17];  if (i < (sizeofcache) / sizeof(int))) {  if (0 == cache[i]) {  cache[i] = i > ? 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):** The principles seen here are building secure code, KISS, and effective testing. Without effective testing things could be missed and keeping code simple is a good way to avoid errors and you should always be concerned with building secure code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Not likely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool suite | 9.7.1 | 6D | Enhanced enforcement |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-DCL56-a | Avoid initialization order problems across translation units by replacing non-local static objects with local static objects |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Do not let exceptions escape from destructors or deallocation functions. |

| **Noncompliant Code** |
| --- |
| The class destructor may throw an exception and cause errors. |
| #include <stdexcept>  Class S {  Bool has\_error() const;  Public:  -S() noexcept(false) {  If(has\_error())) {  Throw std::logic\_error(“Something went wrong”);  }  }  }; |

| **Compliant Code** |
| --- |
| This will catch any exceptions and destroy them. |
| class RandomClass {  Bad bad\_member;  public:  -RandomClass()  try{  //...  } catch(...) {  //catch exceptions thrown  return;  }  }; |

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

| **Principles(s):** The principles here would also be to keep it simple, effective testing, and secure coding. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Desctructor-without-noexcept  Delete-without-noexcept | Fully checked |
| AXivion Bauhaus Suite | 6.9.0 | CertC++-DCL57 |  |
| LDRA tool suite | 9.7.1 | 453S | Partially implemented |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-DCL57.a  CERT\_CPP-DCL57.b | Never allow exception to be thrown from destructor. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | Do not modify standard namespaces |

| **Noncompliant Code** |
| --- |
| X is added to namespace causing error |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| By not reserving name it will fix the issue |
| namespace nonstd {  int x;  } |

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

| **Principles(s):** KISS, effective testing, and secure coding. These are the most crucial principles of coding to avoid most errors and confusion. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Not likely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus suite | 6.9.0 | CertC++DCL58 |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-DCL58-a | Do not modify the standard namespaces (std and posix |
| Polyspace Bug Finder | R2020a | CERT C++:DCL58-CPP | Checks for modification of standard namepace |
| PRQA QA-C++ | 4.4 | 4032, 4035, 4631 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Do not define an unnamed namespace in a header file |

| **Noncompliant Code** |
| --- |
| The variable is defined in an unnamed namespace and as a result causes errors |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE  namespace {  int r;  }  #endif //A\_HEADER\_FILE  //a.cpp  #include “a.h”  #include <iostream>  Void f() {  std::cout << “f(): “ << r << std::endl;  r = 7;  //...  }  //b.cpp  #include “a.h”  #include <iostream>  void d() {  std::cout << “d(): “ << r << std::endl;  r = 90;  }  int main () {  extern void f();  f();  d();  f();  d(); |

| **Compliant Code** |
| --- |
| The variable is defined by one translation unit but visible to all and results in expected output. |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE  extern int r;  #endif //A\_HEADER\_FILE  //a.cpp  #include “a.h  #include <iostream>  int r;  void f() {  std::cout << “f(): “ << r << std::endl;  r = 77;  //...  }  //b.cpp  #include “a.h”  #include <iostream>  void d() {  std::cout << “d(): “ << r << std:: endl;  r = 90;  }  int main() {  extern void f();  f();  d();  f();  d();  } |

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

| **Principles(s):** The principles here are to keep the code simple and secure with effective testing techniques. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Not likely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Unnamed-namespace-header | Fully checked |
| Axivion Bauhaus suite | 6.9.0 | CertC++-DCL59 |  |
| 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.

* Automation is good to use throughout the process of enforcing the standards defined in this policy. The product should be automated to ensure that it is constantly updating and improving as more data and information is gathered by the system.

### 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 | likely | Medium | P12 | L1 |
| STD-002-CPP | Low | Not likely | Low | P3 | L3 |
| STD-003-CPP | Low | Not likely | Low | P3 | L3 |
| STD-004-CPP | Low | Not likely | Medium | P2 | L3 |
| STD-005-CPP | Low | Likely | Low | P6 | L2 |
| STD-006-CPP | Low | Not likely | High | P1 | L3 |
| STD-007-CPP | Low | Not likely | Medium | P2 | L3 |
| STD-008-CPP | Low | Likely | Medium | P6 | L2 |
| STD-009-CPP | High | Not likely | Medium | P6 | L2 |
| STD-010-CPP | Medium | Not likely | 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 | Encryption in rest is the process of securely encoding data as it is written into storage and decrypting the data as it is pulled from the storage for use. Encryption in rest should be used anytime data is any level of sensitivity and would cause harm if the data is accessed by somebody not authorized to possess such data. |
| Encryption at flight | Encryption at flight is the process of securely encoding data as it is being transmitted (hence “at flight”). How to encode data in flight depends on how you are transferring said data. For example, if you are using email that needs to be encrypted you should encrypt the email before sending and use a digital signature. |
| Encryption in use | Encryption in use is the process of protecting data as it is being used by the memory. An example of this would be using a password protected profile. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of determining who the user is and this is important to ensure an unauthorized individual does not gain access to data. |
| Authorization | Authorization is the next framework after authentication in which users have different forms of authorization such as a network admin or a manager having access to more sensitive data. |
| Accounting | Accounting not to be mistaken with a monetary accountant is the process of monitoring activity. This process is good to ensure you know who is doing what on the system. |

**\***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.] | 08/08/2023 | Project one | Ronald Spires | Ronald Spires |
| [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 |