**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 | Checking and ensuring the validity of the user input data to prevent unauthorized access and attacks on the system. |
| 1. Heed Compiler Warnings | This type of warning helps the developers identify potential errors in the code that can compromise the quality and the security of the code. Such warning may not prevent codes from compiling however may cause errors. |
| 1. Architect and Design for Security Policies | This includes integrating security policies into the architect and design of the software from the beginning. |
| 1. Keep It Simple | A practice adopted by developers to keep their code clear, and error-free while focusing on the security of the code. This practice involves avoiding the complexity in the code and following the standard coding practice. |
| 1. Default Deny | This is to ensure only the authorized users are granted access to the system, by default all the users who do not meet the access protocol are denied access. |
| 1. Adhere to the Principle of Least Privilege | This principle only grants the minimum level of privilege to complete the task. This principle adds to the security of the software in case of security breaches since each task is granted only the least level of privilege and prevents exploitation. |
| 1. Sanitize Data Sent to Other Systems | Checking the data before transmitting it to other systems to avoid attacks like SQL injections. |
| 1. Practice Defense in Depth | Having multiple layers of defense systems to mitigate possible exploitation. In case any layer is penetrated other defensive layers are still in place to prevent the attacks. |
| 1. Use Effective Quality Assurance Techniques | Reviewing and correcting errors during the developmental cycle helps the dependability of the code. This principle involves identifying the issues within the codes by proper testing and aids the reliability of the code. |
| 1. Adopt a Secure Coding Standard | Following the standard coding helps the readability, manageability, and security of the code. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not define a C-style variadic function. |

| **Noncompliant Code** |
| --- |
| This function is to find the zero value. IF the value is not found in two arguments it can cause issue. |
| #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** |
| --- |
| By implementing add() function, eliminates the undefined variable hence solving the issue. |
| #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):** Architect and Design for Security Policies, Keep It Simple,  Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.1 | Function-ellipsis | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL50 |  |
| Clang | 3.9 | Cert-del50-cpp | Checked by clang-tidy |
| CodeSonar | 8.1p0 | LANG.STRUCT.ELLIPSIS | Ellipsis |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not declare or define a reserved identifier. |

| **Noncompliant Code** |
| --- |
| Using the reserved names can create conflicts. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| Avoid using the leading or trailing underscore in front of the header guard. |
| #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):** Validate Input Data, Keep it Simple, Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 2.10 | Reserved-identifier | Partially checked |
| Axivion Bauhas Suite | 7.2.0 | CertC++-DCL51 |  |
| 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. |
| CodeSonar | 8.1p0 | LANG.ID.NU.MK  LANG.STRUCT.DECL.RESERVED | Macro name C keyword  Declaration of reserved name |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Never qualify a reference type with const or volatile. |

| **Noncompliant Code** |
| --- |
| A const-qualified reference to a char is incorrectly created 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** |
| --- |
| This can be solved 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):** Validate Input Data, Keep it Simple, Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL52 |  |
| Helix QAC | 2024.2 | C++0014 |  |
| Klocwork | 2024.2 | CERT>DCL>REF\_TYPE.CONST\_OR\_VOLATILE |  |
| Parasoft c/C++ test | 2023.1 | CERT\_CP-DCL51-a | Never qualify a reference type with ‘const’ or ‘volatile’ |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Do not write syntactically ambiguous declarations. |

| **Noncompliant Code** |
| --- |
| An ambiguous declaration ends up creating an object named ‘m’ instead of locking the mutex ‘m’. |
| #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 object is given an identifier other than ‘m’. |
| #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 Data, Keep it Simple, Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.STRUCT.DELC.FNEST | Nested Function Declaration |
| Helix QAC | 2024.2 | C++1109, C++2510 |  |
| Klocwork | 2024.2 | CERT.DCL>AMBIGUOUS\_DECL |  |
| LDRA tool suite | 9.7.1 | 296 S | Partially implement |

#### 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 |

| **Noncompliant Code** |
| --- |
| The function is overloaded without a corresponding function. When an object is overloaded using the allocating functions, it can lead to undefined behavior when deleting the object. |
| #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** |
| --- |
| The deallocation function is defined at global scope. |
| #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 Data, Keep it Simple, Adopt a Secure Coding Standard |
| --- |

**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 |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL54 |  |
| Clang | 3.9 | Misc-new-delete-overloads | Checked with clang-tidy |
| Helix QAC | 2024.2 | C++2160 |  |

#### 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 |

| **Noncompliant Code** |
| --- |
| This code operates in kernel spaces and copies the argument data from user space. This can be the link to the data leakage from the user space. |
| #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 the data structure before copying addresses the issue by making sure only the relevant data is transferred. |
| #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{};      arg.a = 1;    arg.b = 2;    arg.c = 3;      copy\_to\_user(usr\_buf, &arg, sizeof(arg));  } |

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

| **Principles(s):** Validate Input Data, Keep it Simple, Adopt a Secure Coding Standard |
| --- |

**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 |  |
| CodeSonar | 8.1p0 | MISC.PADDING.POTB | Padding Passed Across a Trust Boundary |
| Helix QAC | 2024.2 | DF4941, DF4942, DF4943 |  |
| Parasoft C/C++ | 2023.1 | CERT\_CPP-DCL55-a | A pointer to a structure should not be passed to a function that can copy data to the user space |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Avoid cycles during initialization of static objects |

| **Noncompliant Code** |
| --- |
| The functional behavior is undefined since the local array is initialized through recursion. |
| #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** |
| --- |
| By not initializing the local array and by checking if each array has value assigned, this returns the cached value when computed. |
| #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):** Validate Input Data, Keep it Simple, Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Codesonar | 8.1p0 | LANG.STRUCT.INIT.CYCLE  LANG.STRUCT.INIT.UNOREDERED | Initialization Cycle  Unordered Initialization |
| HelixQAC | 2024.2 | C++1552, C++1554, C++1704 |  |
| LDRA tool suite | 9.7.1 | 6 D | Enhanced Enforcement |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-DCL56-a | Avoid initialization order problem across translation units by replacing non-locals 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 may cause undefined 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** |
| --- |
| The destructor should operate consistently even when an exception is being handled by performing action that don’t throw exceptions. |
| class SomeClass {    Bad bad\_member;  public:    ~SomeClass()    try {      // ...    } catch(...) {      // Catch exceptions thrown from noncompliant destructors of      // member objects or base class subobjects.        // NOTE: Flowing off the end of a destructor function-try-block causes      // the caught exception to be implicitly rethrown, but an explicit      // 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):** Validate Input Data, Keep it Simple, Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Destructor-without-noexception  Delete-without-noexception | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL57 |  |
| CodeSonar | 8.1p0 | LANG.STRUCT.EXCP.CATCH  LANG.STRUCT.EXCP.THROW | Use of catch  Use of throw |
| Helix QAC | 2024.2 | C++2045, C++2047, C++4032, C++4631 |  |

#### Coding Standard 9

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

| **Noncompliant Code** |
| --- |
| The declaration of integer x to the std namespace cause undefined behaviour. |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| Instead of declaring x in the std namespace, it is placed in a custom namespace which do not have a reserved name. |
| namespace nonstd {  int x;  } |

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

| **Principles(s):** Validate Input Data, Keep it Simple, Adopt a Secure Coding Standard |
| --- |

**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 |  |
| CodeSonar | 8.1p0 | LANG.STRUCT.DECL.SNM | Modification of Standard Namespaces |
| Helix QAC | 2024.2 | C++3180, C++3181, C++3182 |  |
| Klocwork | 2024.2 | CERT.DCL.STD\_NS\_MODIFIED |  |

#### 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 space in the header file and accessed in two different translation units, which leads to inconsistent outputs. |
| // 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** |
| --- |
| By defining in a single translation unit it ensures consistent behavior in the program. |
| // 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):** Validate Input Data, Keep it Simple, Adopt a Secure Coding Standard |
| --- |

**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 |  |
| Clang | 3.9 | Cert-dcl59-cpp | Checked by clang-tidy |
| CodeSonar | 8.1p0 | LANG.STRUCT.DECL.ANH | Anonymous Namespace in Header File |

### 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 | Unlikely | Medium | High | 2 |
| STD-002-CPP | Low | Unlikely | Low | Low | 3 |
| STD-003-CPP | Low | Unlikely | Low | Low | 3 |
| STD-004-CPP | Low | Unlikely | Medium | Low | 3 |
| STD-005-CPP | Low | Probable | Low | Medium | 2 |
| STD-006-CPP | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Low | Unlikely | Medium | Low | 3 |
| STD-008-CPP | Low | Likely | Medium | Medium | 2 |
| STD-009-CPP | High | Unlikely | Medium | Medium | 2 |
| STD-010-CPP | Medium | Unlikely | Medium | Low | 3 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | It protects the stored data in physical drives and cloud storage. It works by encrypting the data and maintaining access control by distributing the access key to authorized users. |
| Encryption in flight | This takes place when the data is being transferred from one media to another media or location. Data is encrypted before transfer of data takes place and it is transferred through a secure channel and only person with valid authentication can access the data on the receiving end. |
| Encryption in use | This refers to the state of data that is actively in use by the system or user. Data is encrypted during the whole process and is decrypted when it is being processed by the valid user with valid authentications. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is a process of validating the identity. This can be either for the user or the device that is being used. This is a multi-factor process and there are different types of authentication processes. Whether a single-factor authentication is used that only factors one type of authentication factor such as password or pin, or multi-factor authentication that uses more than one factor of authentication, which can be a combination of password, biometric, face recognition, and PIN, its goal is to authenticate and verify the identity of the user or device. |
| Authorization | Authorization is the process that takes place after authentication. After successfully authenticating the identity of the user and the device, access to the data is provided. Access to the data is implemented in a multi-layer authorization process. Only users with the defined role can access or are authorized to access certain types of data. Periodically auditing the role is required to maintain the integrity of the role-based access control. |
| Accounting | Accounting is a process of tracking the distribution of the data and the transactions that took place within the organization. |

**\***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.] | [Insert text.] | [Insert text.] | [Insert text.] | [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 |