**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 | Input validation for all data being entered by users into the system. Prevention of malformed data and that all data being entered into the system is formatted as intended. |
| 1. Heed Compiler Warnings | As code is being written, the compiler warns about errors that come up during compiling. Errors should be addressed and corrected as they are discovered to prevent compounding issues with the program. |
| 1. Architect and Design for Security Policies | Recorded security policies describe how assets operate and interact with each other. This can help identify where potential vulnerabilities exist, such as using unauthorized access to one asset to bridge into another asset. |
| 1. Keep It Simple | Programs should always be designed with simplicity in mind so that functions are easily understood, and unidentified features can’t be used to gain access. |
| 1. Default Deny | Unless features or access are not explicitly allowed for a function, they should always be denied by default. This prevents accidental access to features that cause vulnerability. |
| 1. Adhere to the Principle of Least Privilege | Goes hand in hand with default deny. Users should have the least number of permissions possible to perform their role so that malicious or unintended breaches are less likely to occur. |
| 1. Sanitize Data Sent to Other Systems | Before data is transferred between systems, it should always be validated and scrubbed of unnecessary pieces of information. This prevents the potential for maliciously added data to be transferred between systems. |
| 1. Practice Defense in Depth | Multiple layers of security measures will provide additional overlapping defenses for assets but allowing each layer to cover the weaknesses of another layer, increasing overall security. |
| 1. Use Effective Quality Assurance Techniques | Use the variety of quality assurance techniques available to developers such as testing throughout the development process, identify the requirements for the project, and use good coding standards to create both an efficient and secure product. |
| 1. Adopt a Secure Coding Standard | Identify a consistent secure coding standard for all projects to maintain conformance with. This can vary from company to company with different standards such as CERT. |

### 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** | **Do not define an unnamed namespace in a header file** |
| --- | --- | --- |
| **Data Type** | DCL59-CPP | Defining variables using unnamed namespaces can cause functions using the variable to print unexpected results. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the variable v is defined in an unnamed namespace within a header file and is accessed from two separate translation units. Each translation unit prints the current value of v and then assigns a new value into it. However, because v is defined within an unnamed namespace, each translation unit operates on its own instance of v, resulting in unexpected output. |
| // 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** |
| --- |
| In this compliant solution, v is defined in only one translation unit but is externally visible to all translation units, resulting in the expected behavior. |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE    extern int v;    #endif // A\_HEADER\_FILE    // a.cpp  #include "a.h"  #include <iostream>    int v; // Definition of global variable v    void f() {  std::cout << "f(): " << v << std::endl;  v = 42;  // ...  }    // b.cpp  #include "a.h"  #include <iostream>    void g() {  std::cout << "g(): " << v << std::endl;  v = 100;  }    int main() {  extern void f();  f(); // Prints v, sets it to 42  g(); // Prints v, sets it to 100  f(); // Prints v, sets it back to 42  g(); // Prints v, sets it back to 100  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | cert-dcl59-cpp | Checked by clang-tidy |
| Astrée | 22.10 | unnamed-namespace-header | Fully checked |
| CodeSonar | 8.0p0 | LANG.STRUCT.DECL.ANH | Anonymous Namespace in Header File |
| LDRA tool suite | 9.7.1 | 286 S, 512 S | Fully implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that unsigned integer operations do not wrap** |
| --- | --- | --- |
| **Data Value** | INT30-C | Avoid operations that would result in integers wrapping. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example can result in an unsigned integer wrap during the addition of the unsigned operands ui\_a and ui\_b. If this behavior is unexpected, the resulting value may be used to allocate insufficient memory for a subsequent operation or in some other manner that can lead to an exploitable vulnerability. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution performs a precondition test of the operands of the addition to guarantee there is no possibility of unsigned wrap: |
| #include <limits.h>    void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum;  if (UINT\_MAX - ui\_a < ui\_b) {  /\* Handle error \*/  } else {  usum = ui\_a + ui\_b;  }  /\* ... \*/  } |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | INTEGER\_OVERFLOW | Implemented |
| LDRA tool suite | 9.7.1 | 493 S, 494 S | Partially implemented |
| TrustInSoft Analyzer | 1.38 | unsigned overflow | Exhaustively verified. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-INT30 | Implemented |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do not attempt to modify string literals** |
| --- | --- | --- |
| **String Correctness** | STR30-C | String literals are stored as read-only and thus will result in access violations if attempts are made to alter them. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the char pointer str is initialized to the address of a string literal. Attempting to modify the string literal is undefined behavior: |
| char \*str = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| As an array initializer, a string literal specifies the initial values of characters in an array as well as the size of the array. (See STR11-C. Do not specify the bound of a character array initialized with a string literal.) This code creates a copy of the string literal in the space allocated to the character array str. The string stored in str can be modified safely. |
| char str[] = "string literal";  str[0] = 'S'; |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | string-literal-modfication  write-to-string-literal | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR30 | Fully implemented |
| LDRA tool suite | 9.7.1 | 157 S | Partially implemented |
| PC-lint Plus | 1.4 | 489, 1776 | Partially supported |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Use valid references, pointers, and iterators to reference elements of a basic\_string** |
| --- | --- | --- |
| **SQL Injection** | STR52-CPP | Avoid using invalid references, pointers, and iterators. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example copies input into a std::string, replacing semicolon (;) characters with spaces. This example is noncompliant because the iterator loc is invalidated after the first call to insert(). The behavior of subsequent calls to insert() is undefined. |
| #include <string>    void f(const std::string &input) {  std::string email;    // Copy input into email converting ";" to " "  std::string::iterator loc = email.begin();  for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {  email.insert(loc, \*i != ';' ? \*i : ' ');  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the value of the iterator loc is updated as a result of each call to insert() so that the invalidated iterator is never accessed. The updated iterator is then incremented at the end of the loop. |
| #include <string>    void f(const std::string &input) {  std::string email;    // Copy input into email converting ";" to " "  std::string::iterator loc = email.begin();  for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {  loc = email.insert(loc, \*i != ';' ? \*i : ' ');  }  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | ALLOC.UAF | Use After Free |
| Helix QAC | 2023.3 | DF4746, DF4747, DF4748, DF4749 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR52-a | Use valid references, pointers, and iterators to reference elements of a basic\_string |
| Polyspace Bug Finder | R2023b | CERT C++: STR52-CPP | Checks for use of invalid string iterator (rule partially covered). |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Explicitly construct and destruct objects when manually managing object lifetime** |
| --- | --- | --- |
| **Memory Protection** | MEM53-CPP | Ensure that memory is properly allocated when use of an object begins and deallocated when usage ends. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a class with nontrivial initialization (due to the presence of a user-provided constructor) is created with a call to std::malloc(). However, the constructor for the object is never called, resulting in undefined behavior when the class is later accessed by calling s->f(). |
| #include <cstdlib>    struct S {  S();    void f();  };    void g() {  S \*s = static\_cast<S \*>(std::malloc(sizeof(S)));    s->f();    std::free(s);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the constructor and destructor are both explicitly called. Further, to reduce the possibility of the object being used outside of its lifetime, the underlying storage is a separate variable from the live object. |
| #include <cstdlib>  #include <new>    struct S {  S();    void f();  };    void g() {  void \*ptr = std::malloc(sizeof(S));  S \*s = new (ptr) S;    s->f();    s->~S();  std::free(ptr);  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2023.3 | DF4761, DF4762, DF4766, DF4767 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MEM53-a | Do not invoke malloc/realloc for objects having constructors |
| Polyspace Bug Finder | R2023b | CERT C++: MEM53-CPP | Checks for objects allocated but not initialized (rule fully covered). |
| PVS-Studio | 7.29 | V630, V749 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Do not abruptly terminate the program** |
| --- | --- | --- |
| **Assertions** | ERR50-CPP | Ensure all errors are handled to prevent the program from abruptly terminating. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the call to f(), which was registered as an exit handler with std::at\_exit(), may result in a call to std::terminate() because throwing\_func() may throw an exception. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.  throwing\_func();  }    int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  } |

| **Compliant Code** |
| --- |
| In this compliant solution, f() handles all exceptions thrown by throwing\_func() and does not rethrow. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.  try {  throwing\_func();  } catch (...) {  // Handle error  }  }    int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | stdlib-use | Partially checked |
| CodeSonar | 8.0p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| Polyspace Bug Finder | R2023b | CERT C++: ERR50-CPP | Checks for implicit call to terminate() function (rule partially covered) |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Honor exception specifications** |
| --- | --- | --- |
| **Exceptions** | ERR55-CPP | Ensure all exceptions are of the correct type. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a function is declared as nonthrowing, but it is possible for std::vector::resize() to throw an exception when the requested memory cannot be allocated. |
| #include <cstddef>  #include <vector>    void f(std::vector<int> &v, size\_t s) noexcept(true) {  v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the function's noexcept-specification is removed, signifying that the function allows all exceptions. |
| #include <cstddef>  #include <vector>    void f(std::vector<int> &v, size\_t s) {  v.resize(s); // May throw, but that is okay  } |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | unhandled-throw-noexcept | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR55 |  |
| CodeSonar | 8.0p0 | LANG.STRUCT.EXCP.THROW | Use of throw |
| LDRA tool suite | 9.7.1 | 56 D | Partially implemented |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not use std::rand() for generating pseudorandom numbers** |
| --- | --- | --- |
| Random Integers | MSC50-CPP | Avoid using std::rand() for creating random numbers, because this does not produce truly random numbers. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code generates an ID with a numeric part produced by calling the rand() function. The IDs produced are predictable and have limited randomness. Further, depending on the value of RAND\_MAX, the resulting value can have modulo bias. |
| #include <cstdlib>  #include <string>    void f() {  std::string id("ID"); // Holds the ID, starting with the characters "ID" followed  // by a random integer in the range [0-10000].  id += std::to\_string(std::rand() % 10000);  // ...  } |

| **Compliant Code** |
| --- |
| The C++ standard library provides mechanisms for fine-grained control over pseudorandom number generation. It breaks random number generation into two parts: one is the algorithm responsible for providing random values (the engine), and the other is responsible for distribution of the random values via a density function (the distribution). The distribution object is not strictly required, but it works to ensure that values are properly distributed within a given range instead of improperly distributed due to bias issues. This compliant solution uses the Mersenne Twister algorithm as the engine for generating random values and a uniform distribution to negate the modulo bias from the noncompliant code example. |
| #include <random>  #include <string>    void f() {  std::string id("ID"); // Holds the ID, starting with the characters "ID" followed  // by a random integer in the range [0-10000].  std::uniform\_int\_distribution<int> distribution(0, 10000);  std::random\_device rd;  std::mt19937 engine(rd());  id += std::to\_string(distribution(engine));  // ...  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | bad-function (AUTOSAR.26.5.1A) | Fully checked |
| Clang | 4.0 (prerelease) | cert-msc50-cpp | Checked by clang-tidy |
| CodeSonar | 8.0p0 | BADFUNC.RANDOM.RAND | Use of rand |
| ECLAIR | 1.2 | CC2.MSC30 | Fully implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Value-returning functions must return a value from all exit paths** |
| --- | --- | --- |
| Proper Returns | MSC52-CPP | When a function returns a value, you must return values from all possible exit paths. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the programmer forgot to return the input value for positive input, so not all code paths return a value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, all code paths now return a value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  return a;  } |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | return-implicit | Fully checked |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this rule, such as function-try-blocks |
| CodeSonar | 8.0p0 | LANG.STRUCT.MRS | Missing return statement |
| LDRA tool suite | 9.7.1 | 2 D, 36 S | Fully implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Ensure that operations on signed integers do not result in overflow** |
| --- | --- | --- |
| Overflow | INT32-C | Make sure that operations being performed on signed integers do not result in overflow. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example can result in a signed integer overflow during the addition of the signed operands si\_a and si\_b: |
| void func(signed int si\_a, signed int si\_b) {  signed int sum = si\_a + si\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution ensures that the addition operation cannot overflow, regardless of representation: |
| #include <limits.h>    void f(signed int si\_a, signed int si\_b) {  signed int sum;  if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||  ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {  /\* Handle error \*/  } else {  sum = si\_a + si\_b;  }  /\* ... \*/  } |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | integer-overflow | Fully checked |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| LDRA tool suite | 9.7.1 | 493 S, 494 S | Partially implemented |
| TrustInSoft Analyzer | 1.38 | signed\_overflow | Exhaustively verified (see one compliant and one non-compliant example). |

### 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.

DevSecOps can certainly be used to enhance an existing DevOps policy, enabling better implementation and enforcement of the existing set of policies. Standard DevOps cannot keep up with the security needs of companies in today’s environment, so enhancing the implementation of these procedures throughout the entire process is essential to maximizing program security. The figure above displays how DevSecOps applies the existing DevOps principles throughout the entire development process, enabling companies to better adhere to the standards of maintaining security throughout their entire

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CPP | Low | Likely | Low | P9 | L2 |
| STD-004-CPP | High | Probable | High | P6 | L2 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Probable | Medium | P4 | L3 |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CPP | Medium | Unlikely | Low | P6 | L2 |
| STD-009-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-010-CPP | High | Likely | High | P9 | L2 |

### 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 | This is the term for encryption on data that is not being used currently. This would include protecting data that was being stored, for example, on a hard drive. This protects data that would not be included with attacks on active processes. Encrypting data at rest allows programmers to delay access to it and limit any potential damage if it is eventually compromised. |
| Encryption at flight | This is the term for encryption on data that is in motion, such as data that is pulled by a program from a database. This allows data to be protected while it moves from object to object and may be at its most vulnerable. An example of something that might protect data at flight is something like a VPN that encrypts and prevents the data, and thus, the person using it, from being tracked. |
| Encryption in use | This is the term for encryption on data that is being used. This would include data that a program is actively using for some function, such as editing, removing, or adding to the data. By protecting his type of data, it can delay the effectiveness of attacks and limit the utility of doing so. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication involves validating who is accessing the system. This is used to differentiate who is allowed to access the system and prevents access if the user is not allowed. A common example of this is a login system using a username and password, and often text-based authentication such as sending a code to the provided cell phone number or email. This is great for preventing unauthorized 3rd party access to critical systems. |
| Authorization | Authorization builds on the concepts of authentication by providing the actual credentials to different users determining what they are allowed to access. The intent is for only intended users to be able to access their assigned resources and to be blocked form non-essential systems. This policy protects from both external and internal threats to existing systems. |
| Accounting | Accounting is the final policy or “step” in the Triple-A Framework. This policy is for tracking instances of access or changes to the system so that there is always a trail leading back to whoever interacted with it. This policy can help with both preventing and responding to security incidents since it can be used to identify when the system is accessed maliciously or if changes are made as a result. |

**\***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 |  |
| 1.1 | 1/28/2024 | Milestone | Zachary DeRepentigny | [Insert text.] |
| 1.2 | 02/25/2024 | Project One | Zachary DeRepentigny |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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