**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 | This principle goal is to ensure all input data is verified as being honest data prior to being used. This could include checking for malicious input data but also the type and size of data coming in. |
| 1. Heed Compiler Warnings | During the developmental stages, this principle’s goal is to heed or abide to the security warnings that are given. This could be issues with vulnerabilities as well as bugs. |
| 1. Architect and Design for Security Policies | This principle is very similar to the one above, however this principle’s goal is during the design and architectural phase of software development. This is to ensure that security requirements are met and implemented. This is achieved by applying security measures such as Defense-in-Depth, threat modeling as well as secure authentication and testing. |
| 1. Keep It Simple | This principle is at it speaks, keep it simple. The goal here is to ensure the design and security system is setup simply. This not only ensures vulnerabilities are subsided but it also is an easier way to maintain. If it is set up correctly, it can be done simple and making it simple to maintain while keeping it secure. |
| 1. Default Deny | This principle ensures the authorization of access is grated to only those who have access. This will default to always deny access to whoever may not be granted that access. This could be user login based and creating security mechanisms to control access. |
| 1. Adhere to the Principle of Least Privilege | This principle focus is to grant users the minimum access to accomplish the tasks they may be required to do. This not only minimizes risks that may cause damage but it also sets the program or software up for a need-to-know basis. |
| 1. Sanitize Data Sent to Other Systems | Exactly what the title states, this principle will use a process of sanitizing data prior to sending out to external systems. This minimizes the risk of malicious data to be transmitted to another external system. By following similar procedures as mentioned above, helps eliminate that risk. |
| 1. Practice Defense in Depth | This principle implements multilayered security measures to protect systems data from threats. This takes in consideration that nothing is bulletproof and assumes that everything has an open door. By taking the steps necessary and implementing security layers to ensure no door is left ajar. |
| 1. Use Effective Quality Assurance Techniques | This principle focus on implementing vigorous quality assurance steps to ensure security steps are maintained and implemented. When those vulnerabilities arise, it is identified and preventative steps to ensure the security steps are taken and implemented. |
| 1. Adopt a Secure Coding Standard | This principle ensures that the proper security coding standard is implemented and in practice. This helps guide security development as well as guide when coding errors arise. |

### 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** |
| --- |
| Using a C-style variadic function will read to see when value 0 is found. If this is found, it will result in undefined behavior. |
| #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** |
| --- |
| Compliant Code doesn’t result in undefined behavior if it isn’t terminated with 0. When including add() function it allows call sites to be identical. |
| #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…\_;  } |

| **Principles(s):** By reducing the usage of incorrect variadic functions by setting up error checks as well as ensuring that the code is compatible. This will ensure that best practices are placed and followed so undefined behavior doesn’t occur. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Function-ellipsis | Fully Checked |
| Clang | 3.9 | Cert-dc150-cpp | Checked by clang-tidy. |
| CodeSonar | 7.4p0 | LANG.STRUCT.ELLIPSIS | Ellipsis |
| LDRA tool Suite | 9.7.1 | 41 S | Fully Implemented |

#### Coding Standard 2

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

| **Noncompliant Code** |
| --- |
| Uses reserved names for header guards. Use of trailing underscores. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_  // Contents of <my\_header.h>  #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| Doesn’t or avoids using reserved names for header guard. The following codes doesn’t use trailing underscores |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H  // Contents of <my\_header.h>  #endif // MY\_HEADER\_H |

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

| **Principles(s):** Code that doesn’t declare or define a reserved identifier means that will need identifier that can reduce errors. By following best practices and maintaining consistent behavior even over time. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Reserved-identifier | Partially Checked |
| Clang | 3.9 | -Wreserved-id-marco  -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 |
| CodeSonar | 7.4p0 | LANG.ID.NU.MK LANG.STRUCT.DECL.RESERVED | Macro name is C keyboard |
| LDRA tool Suite | 9.7.1 | 86 S, 218 S, 219 S, 580 S | Fully Immplemented. |

#### 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** |
| --- |
| Declaring p as a const char making the code systematically incorrect |
| #include <iostream>  void f(char c) {  char &const p = c;  p = ‘p’;  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| Removes ‘const’ from the code |
| #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):** No ambiguous code will lead to clear intent to what the code is intending on accomplishing as well as it makes it clear for monitoring and maintaining. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C+ +test | 2023.1 | CERT\_CPP-DCL52-a | Never qualify a reference type with ‘const’ or ‘volatile’ |
| Polyspace Bug Finder | R2023a | CERT C++: DCL52-CPP | Checks for: constqualified reference types, modification of constqualified reference types. |
| Clang | 3.9 |  | Clang checks for violations of this rule and produces errors without the need to specify any special flags or options |
| SonarQube C/C++ Plugin | 4.10 | S3708 |  |

#### Coding Standard 4

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

| **Noncompliant Code** |
| --- |
| This code will specifically locks when working with mutexes. After locking it there will be’ shared\_resources’ by variables of 42 which allows no other thread to be accessed during that time. |
| #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** |
| --- |
| [Compliant description] |
| #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):** continuous monitoring wouldn’t be possible without clear data  Symmetry is important across all aspects because it gives a clear guide on how to maintain code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | LANG.STRUCT.DECL.FNEST | Nested Function Declaration |
| LDRA tool suite | 9.7.1 | 296 S | Partially implemented |
| Parasoft C/C+ +test | 2023.1 | CERT\_CPP-DCL53-a, CERT\_CPP-DCL53-b, CERT\_CPP-DCL53-c | Shouldn’t be used with () |
| Polyspace Bug Finder | R2023a | CERT C++: DCL53-CPP | Checks for confusion between a function and object declaration and unnamed objects or functions |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Overload allocation and deallocation function as a pair in the same scope |

| **Noncompliant Code** |
| --- |
| Deallocation function isn’t present and this would cause an undefined result because if overload were to occur, you wouldn’t be able to delete 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();  }  // No corresponding global delete operator defined. |

| **Compliant Code** |
| --- |
| Creating this solution helps deallocate the object and define 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  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):** Memory allocation is important because important data that is needed should be strategized for optimizing memory. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | New-delete-pairwise | Partially checked |
| Clang | 3.9 | misc-new-delete-overloads | Checked with clang-tidy |
| Parasoft Bug Finder | R2023a | CERT C++: DCL54-CPP | This will check mismatches between overloaded operator and and deleted |
| RuleChecker | 22.10 | new-delete-pairwise | Partially checked |

#### 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 noncompliant code is too vulnerable to sensitive information being leaked because the bits that are used for padding can include sensitive information that could be copied to another 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** |
| --- |
| This is a stronger structure of data and it will serialize it before its released or leaked to an untrusted area. |
| #include <cstddef>  #include <cstring>  struct test {  int a;  char b;  int c;  };  // Safely copy bytes to user space.  Extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);  Void do\_stuff(void \*usr\_buf) {  Test arg{1, 2, 3};  // May be larger than strictly needed.  unsigned char buf[sizeof(arg)];  std::size\_t offset = 0;  std::memcpy(buf + offset, &arg.a, sizeof(arg.a));  offset += sizeof(arg.a);  std::memcpy(buf + offset, &arg.b, sizeof(arg.b));  offset +=sizeof(arg.b);  std::memcpy(buf + offset, &arg.c, sizeof(arg.c));  offset += sizeof(arg.c);  copy\_to\_user(usr\_buf, buf, offset /\* size of info copied \*/);  } |

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

| **Principles(s):** Continuously monitoring will prevent data from leaking as well as being breached by unauthorized user. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | MISC.PADDING.POTB | Padding Passed Across a Trust Boundary |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-DCL55-a | Pointer to a structure that shouldn’t be passed to a function so that it can copy data to the user space |
| Polyspace Bug Finder | R2023a | CERT C++: DCL55-CPP | Checks for leakage due to structure padding |
| Helix QAC | 2023.1 | DF4941, DF4942, DF4943 |  |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| Caching is used in an attempt to identify static local array cache this becomes undefined. |
| #include <stdexcept>  int fact(int i) noexcept(false) {  if (i < 0) {  // Negative factorials are undefined.  Throw std::domain\_error(“I must be >= 0”);  }  static const int cache[] = {  fact(0), fact(1), fact(2), fact(3), fact(4), fact(5), fact(6), fact(7), fact(8), fact(9), fact(10), fact(11), fact(12), fact(13), fact(14), fact(15), fact(16)  };  if (i < (sizeof(cache) / sizeof(int))) {  return cache[i];  }  return i > 0 ? i \* fact(i – 1) : 1;  } |

| **Compliant Code** |
| --- |
| This will avoid the static local array cache and takes zero-initialization in an attempt to see if each array have a value. |
| #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[17];  if (i < (sizeof(cache) / sizeof(int))) {  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):** continuously monitoring code for constancy and efficacy check  No ambiguous code to keep things clear and understandable. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | LANG.STRUCT.INIT.CYCLE, LANG.STRUCT.INIT.UNORDERED | Initialization Cycle and Unordered Initialization |
| Helix QAC | 2023.1 | C++1552, C++1554, C++1704 |  |
| LDRA toll suite | 9.7.1 | 6 D | Enhanced Enforcement |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-DCL56-a | Avoids order problems across units by replacing non-local static objects with static objects |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-008-CPP | Do not depend on the order of evaluation for side effects |

| **Noncompliant Code** |
| --- |
| i is evaluated more than once so it results in undefined |
| void f(int i, const int \*b){  int a = i +b[++i];  //…  } |

| **Compliant Code** |
| --- |
| The order is changed and can only be processed one way |
| void f(int i, const int \*b){  ++i;  int a = i +b[i];  //…  } |

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

| **Principles(s):** No ambiguous code will allow for simple code and clarity if multiple side effects are needed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion | 7.2.0 | CertC++-EXP50 |  |
| Clang | 3.9 | -Wunsequenced | Can find simple violations of this rule |
| CodeSonar | 7.4p0 | LANG.STRUCT.SE.DEC, LANG.STRUCT.SE.INC | Decrement side effects, Increment side effects |
| Coverity | V7.5.0 | EVALUATION\_ORDER | This can detect when there is a statement that contains multiple side effects with the same value on an undefined evaluation. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-009-CPP | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| Although this attempts to check for values to be within a range, it does so by casting enumeration type. This may never fully represent its value resulted in unspecified behavior. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  if (enumVar < First || enumVar > Third) {  //Handle error  }  } |

| **Compliant Code** |
| --- |
| This will now verify checks the that the value didn’t result in an unspecified. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  if (enumVar < First || enumVar > Third) {  //Handle error  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** Continuous monitoring will ensure that the value didn’t result in unspecified and ensure that it was in range. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | cast-integer-to-enum | Partially Checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| CodeSonar | 7.4p0 | LANG.CAST.COERCE, LANG.CAST.VALUE | Coercion Alters Value, Casts Alters Value |
| RuleChecker | 22.10 | cast-integer-to-enum | Partially Checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-010-CPP | Guarantee that container indices and iterators are within the valid range |

| **Noncompliant Code** |
| --- |
| Pos and value both can have data coming from untrusted sources. Because of the check if value is in range, the pos only check upper bound and not lower bound, this can result in negative and e outside the table bounds. |
| #include <cstddef>  void insert\_in\_table(int \*table, std::size\_t tableSize, int pos, int value) {  if(pos >= tableSize){  //Handle error  return;  }  table[pos] = value;  } |

| **Compliant Code** |
| --- |
| Pos instead is placed and declared as size\_t and that will stop it from getting negative results |
| #include <cstddef>  void insert\_in\_table(int \*table, std::size\_t tableSize, std::size\_t pos, int value) {  if(pos >= tableSize){  //Handle error  return;  }  table[pos] = value;  } |

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

| **Principles(s):** Continuous monitoring will allow to always check that the value is in range and prevent it from going out of bounds. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | overflow\_upon\_dereference |  |
| LDRA tool suite | 9.7.1 | 45 D, 47 S, 476 S, 489 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X | Partially Implemented |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-CTR50-a | A Guarantee that indices are within a valid range |
| Polyspace Bug Finder | R2023a | CERT C++: CTR50-CPP | Checks to see if array is out of bounds, tainted with an index pointer and if the pointer dereference with tainted offset. |

### 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 a great way to provide not just consistency as the code evolves but it also provides real time audits that won’t only improve over time but allow efficiency. This is a great way to save not just time but money as well to keep accuracy at its sharpest.

### 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 | P12 | L2 |
| STD-002-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-003-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-004-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-005-CPP | Low | Probable | Low | P6 | L2 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-008-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| 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 | Encryption in rest is the first step to setting up secure measures to block unauthorized users from having access to this encrypted data. This could include setting up a secure password to grant access as well as setting up boundaries from security attacks |
| Encryption at flight | Encryption in flight is when data is being encrypted while it’s being process between devices and network. This places data at a higher risk of security breaches because security measures need to be in place so unauthorized users will intercept the data. |
| Encryption in use | Encryption in use is when it is processing while the CPU is being used. This allows the code to not have to be decrypted prior making this security higher. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The process of authentication is to set up user name and password and possibly more access alternatives such as a two-factor authentication code that may needed to access data. Time frames can be set as well to ensure when a login may have no activity and force a logout for security measures. |
| Authorization | This process of authorization is setting up the access controls for a user. This could be by setting up what a user has access to such as read and write permissions. Once that is established, the user credentials can grant that access. There will need to be authorization to the one who built the access but also block out access to each user’s login data. |
| Accounting | This is the process of where we combine all of the above to know when a user logs in and the duration that may be placed while using their login functionalities. This not only sets up security measures to ensure that the user using it is only accessing that information they have been permitted to them but it will analyze the timestamps and usage not just for the improvement of the program being used, but also a way to analyze when security breaches may take place and possibly find out how they were broken. |

**\***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
  + - 4: Don’t write ambiguous declarations
    - 5: Overload and deallocation are paired together
    - 6: Avoid information leakage
    - 10: Guarantees indices are within a range
* Firewall logs
  + - 4: Don’t write ambiguous declarations
    - 5: Overload and deallocation are paired together
    - 6: Avoid information leakage
    - 7: Avoid cycles with static objects
    - 9: Don’t cast out an out-of-range enum value
    - 10: Guarantees indices are within a range
* Anti-malware logs
  + - 4: Don’t write ambiguous declarations
    - 5: Overload and deallocation are paired together
    - 6: Avoid information leakage
    - 7: Avoid cycles with static objects
    - 9: Don’t cast out an out-of-range enum value
    - 10: Guarantees indices are within a range

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 |  |
| 1.2 | 08/06/2023 | Project One | Katelin Kaneen | Katelin Kaneen |
| [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 |