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



# Green Pace Secure Development Policy

# Michael Tricoli

## 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 | Programs that take in Data have a vulnerability of malicious inputs being used like SQL injection. Having your code verify and make sure all inputs are only being received in formats the programmer is looking for. This can be done by making having statements that look at the users input before accepting it and only running if the input matches whatever security parameters you have set in place. |
| 1. Heed Compiler Warnings | When compiling code in most IDEs like Visual studio the IDE will look for possible issues in the code. These warnings might go off but still allow you to run the program even with vulnerabilities in case the user purposely wanted to include them. Reading these warnings can save you by finding possible exploits or buffer overflows and warning you to change them before completing the code. Reading these and applying the fix can prevent your code from running into issues on release and increase your security. |
| 1. Architect and Design for Security Policies | Many program languages have built in or website that lays out all the rules for building a secure code. Following all these procedures while designing the code will help to keep your work secure while you’re writing the program. Following them from the start will prevent you from having to re write entire sections to build up a strong set of defenses. |
| 1. Keep It Simple | Follow the rules as they are don’t try too hard to layer on excess security into the code. Too much code will sacrifice the speed of the code and may even end up in crashes for some end users. So, keep your code simple and just apply the security layers you need, and nothing more so not only will your code be secure and run proper, but others will be able to follow along your work if you work with a team to change things later. |
| 1. Default Deny | All access to your software should instantly deny those without permission. Don’t allow hackers to even attempt to get into your software. All credentials should be added before a user can try to access your software. |
| 1. Adhere to the Principle of Least Privilege | Whatever access the machine or user needs to finish the task is all the users should be allowed to do or interact with. If a User uses a program to read data they shouldn’t be able to inject code that would net a response from the program. |
| 1. Sanitize Data Sent to Other Systems | When data is being sent to other systems, they should only receive the data they need. Like with least privilege you don’t want hackers to grab extra data that could be sent though even though it may not be displayed or printed it will be seen in the execution and assembly of the code. |
| 1. Practice Defense in Depth | Layering redundant protections will keep hackers away from your systems. Having security on top or security makes a maze that discourages people from trying to get through. Even if they breach one wall there’s more walls and wrong turns turning the code into a labyrinth to breach. |
| 1. Use Effective Quality Assurance Techniques | Run code through proper tests to check for all vulnerabilities. If you test the code trying to breach your own work, you can stop as many exploits as possible before the product is released. This is to iron out any possible entry points to make sure once code is released it will not get breached. |
| 1. Adopt a Secure Coding Standard | Having a secure coding standard means all the code you write or anyone in the company writes will come with a strong amount of security built in. Following it for every code means everyone will have strong code and eventually not even think about it just constantly outputting work with the security already at a higher level. |

### 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** |
| --- |
| The function reads arguments until the value 0 is found. Calling this function without passing the value 0 as an argument (after the first two arguments) results in undefined behavior. Furthermore, passing any type other than an int also results 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** |
| --- |
| Unlike the C-style variadic function used in the noncompliant code example, this compliant solution does not result in undefined behavior if the list of parameters is not terminated with 0. Additionally, if any of the values passed to the function are not integers, the code is ill-formed rather than producing undefined behavior. |
| #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):** Not defining C-style variadic functions is a type of data validating. Without this implementation the function will keep reading arguments until the value of 0 is found which could cause a buffer overflow. It will also prevent undefined behavior from passing incorrect data types. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | function-ellipsis | Fully checked |
| Clang | 3.9 | cert-dcl50-cpp | Checked by clang-tidy. |
| CodeSonar | 7.0p0 | 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] | Overload allocation and deallocation functions as a pair in the same scope |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an allocation function is overloaded at global scope. However, the corresponding deallocation function is not declared. |
| #include <Windows.h>  #include <new>    void \*operator new(std::size\_t size) noexcept(false) {  static HANDLE h = ::HeapCreate(0, 0, 0); // Private, expandable heap.  if (h) {  return ::HeapAlloc(h, 0, size);  }  throw std::bad\_alloc();  }    // No corresponding global delete operator defined. |

| **Compliant Code** |
| --- |
| In this compliant solution, the corresponding deallocation function is also 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);  } |

| **Principles(s):** This falls under two principles Architect and Design for Security Policies, Adopt a Secure Coding Standard. When designing any code allocating resources should be deallocated properly to prevent any overload. This should be a standard for any company and incorporated into the design of the code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Low | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | new-delete-pairwise | Partially checked |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-DCL54-a | Always provide new and delete together |
| Polyspace Bug Finder | R2022a | CERT C++: DCL54-CPP | Checks for mismatch between overloaded operator new and operator delete (rule fully covered) |
| RuleChecker | 20.10 | new-delete-pairwise | Partially checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| Because the input is unbounded, the following code could lead to a buffer overflow. |
| #include <iostream>    void f() {  char bufOne[12];  char bufTwo[12];  std::cin.width(12);  std::cin >> bufOne;  std::cin >> bufTwo;  } |

| **Compliant Code** |
| --- |
| The best solution for ensuring that data is not truncated and for guarding against buffer overflows is to use std::string instead of a bounded array, as in this compliant solution. |
| #include <iostream>  #include <string>    void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):** 2. There are three principles related to this section Heed Compiler Warnings, Keep it Simple, Use Effective Quality Assurance Techniques. With strings if Visual studio doesn’t think you chose the right size to store them it will throw a warning. Your code will still function, but the warning should be taken into consideration. Keep it simple also applies as you should assign the proper size and nothing more than needed for memory. Quality assurance applies as having proper size selection makes sure the code will run efficiently keeping the program response times adequate. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the 'char' buffer to store input from 'std::cin' |
| Polyspace Bug Finder | R2022a | CERT C++: STR50-CPP | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation   Rule partially covered. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| his noncompliant code example shows JDBC code to authenticate a user to a system. The password is passed as a char array, the database connection is created, and then the passwords are hashed. |
| import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Statement;    class Login {  public Connection getConnection() throws SQLException {  DriverManager.registerDriver(new  com.microsoft.sqlserver.jdbc.SQLServerDriver());  String dbConnection =  PropertyManager.getProperty("db.connection");  // Can hold some value like  // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  return DriverManager.getConnection(dbConnection);  }    String hashPassword(char[] password) {  // Create hash of password  }    public void doPrivilegedAction(String username, char[] password)  throws SQLException {  Connection connection = getConnection();  if (connection == null) {  // Handle error  }  try {  String pwd = hashPassword(password);    String sqlString = "SELECT \* FROM db\_user WHERE username = '"  + username +  "' AND password = '" + pwd + "'";  Statement stmt = connection.createStatement();  ResultSet rs = stmt.executeQuery(sqlString);    if (!rs.next()) {  throw new SecurityException(  "User name or password incorrect"  );  }    // Authenticated; proceed  } finally {  try {  connection.close();  } catch (SQLException x) {  // Forward to handler  }  }  }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses a parametric query with a ? character as a placeholder for the argument. This code also validates the length of the username argument, preventing an attacker from submitting an arbitrarily long user name. |
| public void doPrivilegedAction(  String username, char[] password  ) throws SQLException {  Connection connection = getConnection();  if (connection == null) {  // Handle error  }  try {  String pwd = hashPassword(password);    // Validate username length  if (username.length() > 8) {  // Handle error  }    String sqlString =  "select \* from db\_user where username=? and password=?";  PreparedStatement stmt = connection.prepareStatement(sqlString);  stmt.setString(1, username);  stmt.setString(2, pwd);  ResultSet rs = stmt.executeQuery();  if (!rs.next()) {  throw new SecurityException("User name or password incorrect");  }    // Authenticated; proceed  } finally {  try {  connection.close();  } catch (SQLException x) {  // Forward to handler  }  }  } |

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

| **Principles(s):** This one follows the principles of validate input data, Architect and Design for Security Policies, Keep It Simple, Default Deny, Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard. The reason it follows so many principles is because of how easy yet effective it can be. The most basic attack someone can use on an SQL database is SQL injection. Because it is so easy for someone to breach using injection and its also just as easy to defend against all these policies apply to this standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| CodeSonar | 7.0p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| Parasoft Jtest | 2022.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| SonarQube | 6.7 | S2077  S3649 | Executing SQL queries is security-sensitive  SQL queries should not be vulnerable to injection attacks |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Only free memory allocated dynamically |

| **Noncompliant Code** |
| --- |
| This noncompliant code example sets c\_str to reference either dynamically allocated memory or a statically allocated string literal depending on the value of argc. In either case, c\_str is passed as an argument to free(). |
| #include <stdlib.h>  #include <string.h>  #include <stdio.h>    enum { MAX\_ALLOCATION = 1000 };    int main(int argc, const char \*argv[]) {  char \*c\_str = NULL;  size\_t len;    if (argc == 2) {  len = strlen(argv[1]) + 1;  if (len > MAX\_ALLOCATION) {  /\* Handle error \*/  }  c\_str = (char \*)malloc(len);  if (c\_str == NULL) {  /\* Handle error \*/  }  strcpy(c\_str, argv[1]);  } else {  c\_str = "usage: $>a.exe [string]";  printf("%s\n", c\_str);  }  free(c\_str);  return 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution eliminates the possibility of c\_str referencing memory that is not allocated dynamically when passed to free(). |
| #include <stdlib.h>  #include <string.h>  #include <stdio.h>    enum { MAX\_ALLOCATION = 1000 };    int main(int argc, const char \*argv[]) {  char \*c\_str = NULL;  size\_t len;    if (argc == 2) {  len = strlen(argv[1]) + 1;  if (len > MAX\_ALLOCATION) {  /\* Handle error \*/  }  c\_str = (char \*)malloc(len);  if (c\_str == NULL) {  /\* Handle error \*/  }  strcpy(c\_str, argv[1]);  } else {  printf("%s\n", "usage: $>a.exe [string]");  return EXIT\_FAILURE;  }  free(c\_str);  return 0;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard. This one can probably fill a few more principles but I chose the basic it covers. Freeing memory that is not allocated dynamically can result in heap corruption and other serious errors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | invalid-free | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM34 | Can detect memory deallocations for stack objects |
| Clang | 3.9 | clang-analyzer-unix.Malloc | Checked by clang-tidy; can detect some instances of this rule, but does not detect all |
| CodeSonar | 7.0p0 | ALLOC.TM | Type Mismatch |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use a static assertion to test the value of a constant expression. |

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** Use Effective Quality Assurance Techniques and Adopt a Secure Coding Standard. Testing code is an important aspect of secure coding and should be utilized to maximize protection. Assertions are diagnostic tools and using static assertions will make sure if anything tries to change you will be made aware of it during testing. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 7.0p0 | customization | Users can implement a custom check that reports uses of the assert() macro |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Guarantee exception safety. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows a flawed copy assignment operator. The implicit invariants of the class are that the array member is a valid (possibly null) pointer and that the nElems member stores the number of elements in the array pointed to by array. The function deallocates array and assigns the element counter, nElems, before allocating a new block of memory for the copy. As a result, if the new expression throws an exception, the function will have modified the state of both member variables in a way that violates the implicit invariants of the class. |
| #include <cstring>    class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }      IntArray(const IntArray& that); // nontrivial copy constructor  IntArray& operator=(const IntArray &rhs) {  if (this != &rhs) {  delete[] array;  array = nullptr;  nElems = rhs.nElems;  if (nElems) {  array = new int[nElems];  std::memcpy(array, rhs.array, nElems \* sizeof(\*array));  }  }  return \*this;  }    // ...  }; |

| **Compliant Code** |
| --- |
| In this compliant solution, the copy assignment operator provides the strong exception safety guarantee. The function allocates new storage for the copy before changing the state of the object. Only after the allocation succeeds does the function proceed to change the state of the object. |
| #include <cstring>    class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }    IntArray(const IntArray& that); // nontrivial copy constructor    IntArray& operator=(const IntArray &rhs) {  int \*tmp = nullptr;  if (rhs.nElems) {  tmp = new int[rhs.nElems];  std::memcpy(tmp, rhs.array, rhs.nElems \* sizeof(\*array));  }  delete[] array;  array = tmp;  nElems = rhs.nElems;  return \*this;  }    // ...  }; |

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

| **Principles(s):** Practice Defense in Depth. Errors will tend to close the program before seeing what will happen further down the line. Having exceptions allows the code to continue operating even after the exception is thrown. This will help to see what happens to the program after you have passed the exception and paint a better picture for how to correct and secure it. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | ALLOC.LEAK | Leak |
| LDRA tool suite | 9.7.1 | 527 S, 56 D, 71 D | Partially implemented |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-ERR56-a  CERT\_CPP-ERR56-b | Always catch exceptions  Do not leave 'catch' blocks empty |
| Polyspace Bug Finder | R2022a | CERT C++: ERR56-CPP | Checks for exceptions violating class invariant (rule fully covered). |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input/Output | [STD-008-CPP] | Do not alternately input and output from a file stream without an intervening positioning call. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example appends data to the end of a file and then reads from the same file. However, because there is no intervening positioning call between the formatted output and input calls, the behavior is undefined. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }    file << "Output some data";  std::string str;  file >> str;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::basic\_istream<T>::seekg() function is called between the output and input, eliminating the undefined behavior. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }    file << "Output some data";    std::string str;  file.seekg(0, std::ios::beg);  file >> str;  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems. This standard is to prevent input/output from happening at the same time of a file opened with update. Having multiple input and outputs mean data may not be saving properly and something that you just saved may not come back or will duplicate when you complete the task. Making sure when data is going back and fourth that the proper steps are taken to order the operations being done and that that file. Whenever a task is complete the file should be saved and closed before the next command begins. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | IO.IOWOP  IO.OIWOP | Input After Output Without Positioning  Output After Input Without Positioning |
| LDRA tool suite | 9.7.1 | 84 D | Fully implemented |
| Parasoft C/C++test | 2021.2 | CERT\_C-FIO39-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| PC-lint Plus | 1.4 | 2478, 2479 | Fully supported |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers | [STD-009-CPP] | Converting a pointer to integer or integer to pointer. |

| **Noncompliant Code** |
| --- |
| The size of a pointer can be greater than the size of an integer, such as in an implementation where pointers are 64 bits and unsigned integers are 32 bits. This code example is noncompliant on such implementations because the result of converting the 64-bit ptr cannot be represented in the 32-bit integer type. |
| void f(void) {  char \*ptr;  /\* ... \*/  unsigned int number = (unsigned int)ptr;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Any valid pointer to void can be converted to intptr\_t or uintptr\_t and back with no change in value. (See INT36-EX2.) The C Standard guarantees that a pointer to void may be converted to or from a pointer to any object type and back again and that the result must compare equal to the original pointer. Consequently, converting directly from a char \* pointer to a uintptr\_t, as in this compliant solution, is allowed on implementations that support the uintptr\_t type. |
| #include <stdint.h>    void f(void) {  char \*ptr;  /\* ... \*/  uintptr\_t number = (uintptr\_t)ptr;  /\* ... \*/  } |

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

| **Principles(s):** Adopt a Secure Coding Standard. To prevent alignment issues with pointers they should be used the way they are first implemented. Changing between integer and pointer can cause addressing inconsistencies and your code can crash or misbehave leaving vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | pointer-integral-cast  pointer-integral-cast-implicit  function-pointer-integer-cast  function-pointer-integer-cast-implicit | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-INT36 | Fully implemented |
| Clang | 3.9 | -Wint-to-pointer-cast, -Wint-conversion | Can detect some instances of this rule, but does not detect all |
| CodeSonar | 7.0p0 | LANG.CAST.PC.CONST2PTR  LANG.CAST.PC.INT | Conversion: integer constant to pointer  Conversion: pointer/integer |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Error Handler | [STD-010-CPP] | Set errno to zero before calling a library function known to set errno, and check errno only after the function returns a value indicating failure. |

| **Noncompliant Code** |
| --- |
| his noncompliant code example fails to set errno to 0 before invoking strtoul(). If an error occurs, strtoul() returns a valid value (ULONG\_MAX), so errno is the only means of determining if strtoul() ran successfully. |
| #include <errno.h>  #include <limits.h>  #include <stdlib.h>    void func(const char \*c\_str) {  unsigned long number;  char \*endptr;    number = strtoul(c\_str, &endptr, 0);  if (endptr == c\_str || (number == ULONG\_MAX  && errno == ERANGE)) {  /\* Handle error \*/  } else {  /\* Computation succeeded \*/  }  } |

| **Compliant Code** |
| --- |
| This compliant solution sets errno to 0 before the call to strtoul() and inspects errno after the call. |
| #include <errno.h>  #include <limits.h>  #include <stdlib.h>    void func(const char \*c\_str) {  unsigned long number;  char \*endptr;    errno = 0;  number = strtoul(c\_str, &endptr, 0);  if (endptr == c\_str || (number == ULONG\_MAX  && errno == ERANGE)) {  /\* Handle error \*/  } else {  /\* Computation succeeded \*/  }  } |

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

| **Principles(s):** Adopt a Secure Coding Standard. Errno is initialized at zero when program starts but doesn’t reset to 0 whenever another library is called. Having it set to 0 everytime a new library opens helps to narrow down any issues and where they come from. This will help to speed up testing as you can track errors faster and correct them. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | errno-reset | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ERR30 | Fully implemented |
| CodeSonar | 7.0p0 | LANG.STRUCT.RC | Redundant Condition |
| Parasoft C/C++test | 2021.2 | CERT\_C-ERR30-a  CERT\_C-ERR30-b | Properly use errno value  Provide error handling for file opening errors right next to the call to fopen |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Probable | Medium | High | 1 |
| STD-002-CPP | Low | Probable | Low | Medium | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Probable | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | High | Likely | High | Medium | 2 |
| STD-008-CPP | Low | Likely | Medium | Medium | 2 |
| STD-009-CPP | Low | Probable | High | Low | 3 |
| STD-010-CPP | Medium | Probable | Medium | Medium | 2 |

### 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 | Whenever Data is encrypted and saved to a storage like a hard drive it is considered at rest. This data can only be stolen by having direct access to the hardware. To some extent all these policies will have to be at rest when not in use. Especially since all actions done on a server should be saved somewhere to have a record of everything the program has been used for and by. |
| Encryption at flight | Whenever Encrypted data being sent over something like a server, router, or anything outside the host pc. Once again, all these policies apply at some point. When a user logs in the login database needs to check that the user is in the system. Changes will need to be sent back to host to apply changes. Level of access might only travel once for initial setup but will still be in flight at some point. |
| Encryption in use | While data is in use it is the most vulnerable to attack. Someone can hack in between that flow of data to steal information or gain access to a system. Protecting the data at flight and in rest help to increase security when the data is in use. The combination of all the standards help to make sure while the data is in use it is safe from any attacks. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Protecting information starts with authentication. Usernames and passwords are only the beginning of this protection. Making sure someone is who they claim to be tough for many programs. The best defense is not only a strong password but 2 forms of authentication like mobile apps or email one-time codes help to provide stronger auth. |
| Authorization | Authorization is to make sure that every user has the correct read/write privileges to a system. Only give them the exact permissions they need and nothing more. The less people that have access to sensitive information the safer it will be. |
| Accounting | Most business level applications have a system to view everything that has been done to a system. Who has logged in and when, with what they did while they were in the system. Having this form of accounting is a way to keep everyone accountable and have an overview of the system at work. The only downside is if these logs are hacked then a hacker can see every detail about the system. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.5 | 05/22/22 | Milestone 3 completion | Michael Tricoli |  |
| 2.0 | 06/12/22 | Completed Template | Michael Tricoli |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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