**Green Pace Developer: Security Policy Guide**

**Author: Christopher Ivey**

**Date: 09/14/2022**



# 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 is set to validate input data from all untrusted data sources. The Proper input validation can eliminate most software vulnerabilities and bug errors. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user-controlled files [Seacord 05]. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code [C MSC00-A, C++ MSC00-A]. Developers fix these compiler warnings by using static and dynamic analysis tools to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set. |
| 1. Keep It Simple | Keep the design as simple and small as possible, because complex designs increase the likelihood that errors will be created during implementation, configuration, and use. In addition, the effort that is required to achieve an appropriate level of assurance increases noticeably as security mechanisms become more complex. |
| 1. Default Deny | Developers should base access decisions on permission (authentication and authorized and unauthorized access) rather than exclusion. This means that, by default, access is denied, and the protection strategy identifies conditions under which access is permitted or denied, based on user credentials entered. |
| 1. Adhere to the Principle of Least Privilege | This is where every process should execute with the least set of privileges necessary or required to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. By using this approach, reduces the opportunities an attacker (hacker) must execute arbitrary code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems [C STR02-A] such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers (;’hackers) may be able to invoke unused functionality in these components using SQL, command, or other SQL injection attacks using hacking algorithms. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem. |
| 1. Practice Defense in Depth | Manage risk with multiple layer defensive strategies, so that if one layer of defense turns out to be ineffective, then another layer of defense can prevent a security flaw/ vulnerability from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. A prime example, is in combining secure programming techniques with secure runtime environments that reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment. |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be included to create an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for your target development language and platform. Adopting these Secure Coding Standard, will help the developer to comply with the industries best coding standards when developing software applications. |

### 

### 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 depend on the order of evaluation for side effects** |
| --- | --- | --- |
| **Data Type** | [STD-DCL51-CPP] | Defining an identifier in a reserved context yields undefined behavior. |

| **Noncompliant Code** |
| --- |
| Many programs use reserved names such as header guards and may clash with a preestablished name. |
| #indef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_  // contents of <my\_header.h>  #endif \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| Avoids using leading underscores |
| #indef 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):** 1. Validate Input Data, we must understand the kind of undefined behavior we are dealing with. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | assert\_failure | • STR51-CPP. Do not attempt to create a std:: string from a null pointer  • STR53-CPP. Range check element access |
| Polyspace  Bug Finder | R2022b | CERT C++  STR53-CPP | Checks for:  • Array access out of bounds  • Array access with tainted index  • Pointer dereferences with tainted offset instances of the rule.  Range check element access. |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer Overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Helix QAC | 2022.3 | C++0014 | DCL52-CPP. Never qualify a reference type with const or volatile |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do not depend on the order of evaluation for side effects** |
| --- | --- | --- |
| **Data Value** | [STD-EXP50-CPP] | There are numerous unintended restrictions and sequencing restrictions that can occur from expressions that have unsequenced ordering. |

| **Noncompliant Code** |
| --- |
| Evaluated more than once and creates undefined expression |
| [void f(int i, const int \*b) {  int a = i + b[++i];  // ...  } |

| **Compliant Code** |
| --- |
| Evaluation can be interpreted in only 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):** 2. Heed Compiler Warnings, Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer Overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Polyspace  Bug Finder | R2022b | CERT C++  STR53-CPP | Checks for:  • Array access out of bounds  • Array access with tainted index  • Pointer dereferences with tainted offset.  Range check element access. |
| Helix QAC | 2022.3 | C++0014 | DCL52-CPP. Never qualify a reference type with const or volatile |
| Astrėe | 20.10 | assert\_failure | • STR51-CPP. Do not attempt to create a std:: string from a null pointer  • STR53-CPP. Range check element access |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Use valid references, pointers, and iterators to reference elements of a basic\_string** |
| --- | --- | --- |
| **String Correctness** | [STD-STR52-CPP] | Invalidated references result in undefined behaviors. |

| **Noncompliant Code** |
| --- |
| [Iterator loc is invalidated, behavior undefined |
| #include <stdio.h>  void fibonacci(void)  {  int i;  int fib[10];    for (i = 0; i < 10; i++)  {  if (i < 2)  fib[i] = 1;  else  fib[i] = fib[i-1] + fib[i-2];  }  printf("The 10-th Fibonacci number is %i .\n", fib[i]);  /\* Defect: Value of i is greater than allowed value of 9 \*/  } |

| **Compliant Code** |
| --- |
| Iterator is incremented at end of 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):** 3. Architect and Design for Security Policies, create a software architecture and design your software to implement and enforce security policies. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P6 b | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer Overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Polyspace  Bug Finder | R2022b | CERT C++  STR53-CPP | Checks for:  • Array access out of bounds  • Array access with tainted index  • Pointer dereferences with tainted offset.  Range check element access. |
| Helix QAC | 2022.3 | C++0014 | DCL52-CPP. Never qualify a reference type with const or volatile |
| Astrėe | 20.10 | assert\_failure | • STR51-CPP. Do not attempt to create a std:: string from a null pointer  • STR53-CPP. Range check element access |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-ISD00-J  ] | SQL queries originating from untrusted sources can create security vulnerabilities. |

| **Noncompliant Code** |
| --- |
| [Permits unsanitized input |
| 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);    void f(char \*password, size\_t bufferSize) {  char localToken[256];  init(localToken, password);  memset(password, ' ', strlen(password)); // Noncompliant, password is about to be freed  memset(localToken, ' ', strlen(localToken)); // Noncompliant, localToken is about to go out of scope  free(password);  } |

| **Compliant Code** |
| --- |
| Sanitized input |

void f(char \*password, size\_t bufferSize) {

char localToken[256];

init(localToken, password);

memset\_s(password, bufferSize, ' ', strlen(password));

memset\_s(localToken, sizeof(localToken), ' ', strlen(localToken));

free(password);

}

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

| **Principles(s):** 4. Keep It Simple, Keep the design as simple and small as possible, because complex designs increase the likelihood that errors will be created during implementation, configuration, and use. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | P6 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer Overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Polyspace  Bug Finder | R2022b | CERT C++  STR53-CPP | Checks for:  • Array access out of bounds  • Array access with tainted index  • Pointer dereferences with tainted offset.  Range check element access. |
| Helix QAC | 2022.3 | C++0014 | DCL52-CPP. Never qualify a reference type with const or volatile |
| Astrėe | 20.10 | assert\_failure | • STR51-CPP. Do not attempt to create a std:: string from a null pointer  • STR53-CPP. Range check element access |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Properly deallocate dynamically allocated resources** |
| --- | --- | --- |
| **Memory Protection** | [STD-MEM51-CPP] | Do not call a deallocation function on anything other than nullptr. |

| **Noncompliant Code** |
| --- |
| End functionality of this results in undefined behavior. |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...    delete s1;  } |

| **Compliant Code** |
| --- |
| Removes call to delete and calls s1’s destructor, one of a few times where this is permitted. |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...    s1->~S();  } |

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

| **Principles(s):** 5. Default Deny, Developers should base access decisions on permission (authentication and authorized and unauthorized access) rather than exclusion. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P3 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer Overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Polyspace  Bug Finder | R2022b | CERT C++  STR53-CPP | Checks for:  • Array access out of bounds  • Array access with tainted index  • Pointer dereferences with tainted offset.  Range check element access. |
| Helix QAC | 2022.3 | C++0014 | DCL52-CPP. Never qualify a reference type with const or volatile |
| Astrėe | 20.10 | assert\_failure | • STR51-CPP. Do not attempt to create a std:: string from a null pointer  • STR53-CPP. Range check element access |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-DCL03-CPP] | Find and eliminate software defects by using assertions |

| **Noncompliant Code** |
| --- |
| Diagnostic is only occurring at runtime. |
| #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** |
| --- |
| Preprocessor conditional statement is used and results in no runtime penalty. |
| 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):** 6. Adhere to the Principle of Least Privilege, this is where every process should execute with the least set of privileges necessary or required to complete the job. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P5 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer Overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Polyspace  Bug Finder | R2022b | CERT C++  STR53-CPP | Checks for:  • Array access out of bounds  • Array access with tainted index  • Pointer dereferences with tainted offset |
| Helix QAC | 2022.3 | C++0014 | DCL52-CPP. Never qualify a reference type with const or volatile |
| Astrėe | 20.10 | assert\_failure | • STR51-CPP. Do not attempt to create a std:: string from a null pointer  • STR53-CPP. Range check element access |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-ERR51-CPP] | Programs can encounter an unrecoverable exception and terminate but not allow it to remain uncaught. |

| **Noncompliant Code** |
| --- |
| f() nor main() catch exceptions. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| The main entry point handles all exceptions |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

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

| **Principles(s):** 7. Sanitize Data Sent to Other Systems, sanitize all data passed to complex subsystems [C STR02-A] such as command shells, relational databases, and commercial off-the-shelf (COTS) components. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P6 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer Overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Helix QAC | 2022.3 | C++0014 | DCL52-CPP. Never qualify a reference type with const or volatile |
| Polyspace  Bug Finder | R2022b | CERT C++  STR53-CPP | Checks for:  • Array access out of bounds  • Array access with tainted index  • Pointer dereferences with tainted offset |
| Astrėe | 20.10 | assert\_failure | • STR51-CPP. Do not attempt to create a std:: string from a null pointer  • STR53-CPP. Range check element access |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-OOP50-CPP] | Calling a derived class function from a base class under construction is dangerous and may access resources that have already been released. |

| **Noncompliant Code** |
| --- |
| Attempts to seize and release and object’s resources through calls to virtual function. |
| struct B {  B() { seize(); }  virtual ~B() { release(); }    protected:  virtual void seize();  virtual void release();  };    struct D : B {  virtual ~D() = default;    protected:  void seize() override {  B::seize();  // Get derived resources...  }    void release() override {  // Release derived resources...  B::release();  }  }; |

| **Compliant Code** |
| --- |
| Call a nonvirtual, private member function instead |
| class B {  void seize\_mine();  void release\_mine();    public:  B() { seize\_mine(); }  virtual ~B() { release\_mine(); }  protected:    virtual void seize() { seize\_mine(); }    virtual void release() { release\_mine(); }  };    class D : public B {    void seize\_mine();    void release\_mine();    public:    D() { seize\_mine(); }    virtual ~D() { release\_mine(); }    protected:    void seize() override {      B::seize();      seize\_mine();    }      void release() override {      release\_mine();      B::release();    }  }; |

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

| **Principles(s):** 8. Practice Defense in Depth, manage risk with multiple layer defensive strategies, so that if one layer of defense turns out to be ineffective, then another layer of defense can prevent a security flaw/ vulnerability from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P6 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer Overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Helix QAC | 2022.3 | C++0014 | DCL52-CPP. Never qualify a reference type with const or volatile |
| Polyspace  Bug Finder | R2022b | CERT C++  STR53-CPP | Checks for:  • Array access out of bounds  • Array access with tainted index  • Pointer dereferences with tainted offset |
| Astrėe | 20.10 | assert\_failure | • STR51-CPP. Do not attempt to create a std:: string from a null pointer  • STR53-CPP. Range check element access |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [STD-FIO51-CPP] | Use stream object by value semantics, not dynamic memory allocation |

| **Noncompliant Code** |
| --- |
| Underlying object not properly closed |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  std::terminate();  } |

| **Compliant Code** |
| --- |
| File resources properly closed |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    file.close();    if (file.fail()) {      // Handle error    }    std::terminate();  } |

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

| **Principles(s):** 9. Use Effective Quality Assurance Techniques, Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be included to create an effective quality assurance program. Independent security reviews can lead to more secure systems. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | P6 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer Overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Helix QAC | 2022.3 | C++0014 | DCL52-CPP. Never qualify a reference type with const or volatile |
| Polyspace  Bug Finder | R2022b | CERT C++  STR53-CPP | Checks for:  • Array access out of bounds  • Array access with tainted index  • Pointer dereferences with tainted offset |
| Astrėe | 20.10 | assert\_failure | • STR51-CPP. Do not attempt to create a std:: string from a null pointer  • STR53-CPP. Range check element access |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Concurrency | [STD-CON43-C] | Synchronization techniques will help avoid software flaws. |

| **Noncompliant Code** |
| --- |
| [Operations are not atomic, and operations can occur concurrently, resulting in saving only one operation. |
| static volatile int account\_balance;    void debit(int amount) {  account\_balance -= amount;  }    void credit(int amount) {  account\_balance += amount;  } |

| **Compliant Code** |
| --- |
| Attackers cannot hack the race condition to steal money from the bank because of the mutex making operations atomic. |
| #include <threads.h>    static int account\_balance;  static mtx\_t account\_lock;    int debit(int amount) {  if (mtx\_lock(&account\_lock) == thrd\_error) {  return -1; /\* Indicate error to caller \*/  }  account\_balance -= amount;  if (mtx\_unlock(&account\_lock) == thrd\_error) {  return -1; /\* Indicate error to caller \*/  }  return 0; /\* Indicate success \*/  }    int credit(int amount) {  if (mtx\_lock(&account\_lock) == thrd\_error) {  return -1; /\* Indicate error to caller \*/  }  account\_balance += amount;  if (mtx\_unlock(&account\_lock) == thrd\_error) {      return -1;   /\* Indicate error to caller \*/    }    return 0;   /\* Indicate success \*/  }    int main(void) {    if(mtx\_init(&account\_lock, mtx\_plain) == thrd\_error) {      /\* Handle error \*/    }    /\* ... \*/  } |

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

| **Principles(s):** 10. Adopt a Secure Coding Standard, Develop and/or apply a secure coding standard for your target development language and platform. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer Overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Helix QAC | 2022.3 | C++0014 | DCL52-CPP. Never qualify a reference type with const or volatile |
| Polyspace  Bug Finder | R2022b | CERT C++  STR53-CPP | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereferences with tainted offset |
| Astrėe | 20.10 | assert\_failure | * STR51-CPP. Do not attempt to create a std:: string from a null pointer * STR53-CPP. Range check element access |

### 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 is used for ensuring that Green Pace’s already well developed DevSecOps is obeying and complying to the standards defined in this policy. DevSecOps and automation can and will boost security measures for the Green Pace infrastructure. Within the integrity checks of the DevSecOps Toolchain, is where the automation bug finder can create Defense in depth techniques to ensure system security follows coding standards.

Within the monitoring and analytics section of the lifecycle is where automation will be able to be utilized before the release of the product. It is very important to implement (Triple A). We will want to ensure that all (Triple A) authentication methods return the correct access and privileges to certain actions and resources within our application. Conclusively we will additionally want to provide authorization to be correct and valid. We can implement our Triple A policies to make sure the overall application is secure enough for deployment and automated security checks during integration testing before it’s released to the masses for use.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STR53-CPP | High | Probable | Low | P6 | L2 |
| STR52-CPP | Low | Unlikely | Medium | P2 | L3 |
| STR50-CPP | Medium | Unlikely | Low | P18 | L1 |
| OOP55-CPP | High | Likely | Medium | P18 | L3 |
| MEM56-CPP | High | Probable | Medium | P18 | L3 |
| MEM53-CPP | Medium | Unlikely | Medium | P2 | L1 |
| OOP56-CPP | Low | Likely | Low | P18 | L3 |
| MSC51-CPP | Medium | Unlikely | Medium | P18 | L1 |
| OOP55-CPP | High | Probable | Medium | P4 | L3 |
| STR51-CPP | Low | Likely | Low | P4 | L3 |
| MSC51-CPP | Medium | Probable | High | P6 | L1 |
| DCL55-CPP | High | Unlikely | High | P4 | L2 |
| DCL54-CPP | Medium | Likely | High | P2 | L1 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | At rest encryption is like storing your data in a vault. |
| Encryption in transit | This is when data communications are protected against being intercepted while it moves between the site and the cloud provider, or between two separate services. |
| Encryption in use | This takes a different approach from the other techniques, that ensures that sensitive data is never left, unprotected, regardless of the current state of the data processing (at rest, in transit, or in use), or location, whether in the cloud or the primary database system. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The first step in securing an identification system. This aids in verifying that the person accessing the system is who they say they are. There are 3 main categories. What they know, Who they are, and What they have. This is basically a method to identifying a user. |
| Authorization | Right after authentication, a user is required to gain authorization for certain tasks. Authorization is simply a methodical process of enforcing system policies, determining what is acceptable and what is denied. |
| Accounting | This is the monitoring of the usage of a person who has logged into a network. This can be applied with a Security Information and Event Management (SIEM). Accounting is carried out when a user logs into a session for a certain amount of system time. |

**\***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 follow 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 |  |
| 2.0 | 10/11/2022 | Update | Chrstopher Ivey | Professor Alim |
| 3.0 | 10/11/2022 | NA | NA | NA |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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