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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Input validation is the process of checking if the input is within the expected domain of valid program input. Inputs are required to conform to type and numeric range requirements, as well as input invariants for the class or subsystem. Input validation can eliminate the vast majority of software vulnerabilities. |
| 1. Heed Compiler Warnings | Code is supposed to be compiled using the highest level of warning available for the compiler and eliminate warnings by editing the code. Static and dynamic analysis tools should also be used to detect and eliminated additional security flaws. |
| 1. Architect and Design for Security Policies | Software should have an architecture and be designed to implement and enforce security policies. For example, different a system could be divided into subsystems based on privileges. |
| 1. Keep It Simple | The design should be kept as simple and small as possible because complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. As the security mechanism become more complex, the effort required to achieve an appropriate level of assurance increases greatly. |
| 1. Default Deny | Access decisions should be based on permission, so the protection scheme identifies conditions under which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Every process of a software program should be executed with the least set of privileges necessary to complete the job, and any elevated privilege should only be accessed for the least amount of time to complete the privileged task. This approach reduces the amount of time an attacker has to execute arbitrary code if he or she were to gain access. |
| 1. Sanitize Data Sent to Other Systems | All data passed to complex subsystems, such as relational database, should be sanitized. Attackers may use SQL commands or other injections to invoke unused functionality in these subsystems. |
| 1. Practice Defense in Depth | Multiple layers of defense should be implemented, so that if one layer turns out to be inadequate, an attacker would have to go through another layer of defense. For example, combining secure programming techniques with secure runtime environments should prevent a security flaw from becoming an exploitable vulnerability. |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques, such as fuzz testing, penetration testing, and source code audits can be effective in identifying and eliminating vulnerabilities. Independent security reviews from external reviewers can lead to more secure systems. |
| 1. Adopt a Secure Coding Standard | You should develop and/or apply a secure coding standard for the programming language and platform you are using. |

### 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** | **Never qualify a reference type with const or volatile** |
| --- | --- | --- |
| **Data Type** | DCL-052-CPP | Qualifying a reference type with const or volatile results in undefined behavior. A conforming compiler is required to issue a diagnostic message. However, if the compiler does not emit a fatal diagnostic, the program may produce surprising results. |

| **Noncompliant Code** |
| --- |
| A const-qualified reference to a char is formed instead of a reference to a const-qualified char. This results in undefined behavior. |
| #include <iostream>  void f(char c) {  char &const p = c;  p = ‘p’;  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| This compliant code correctly declares p to be a reference to a char and removes the const qualifier. |
| #include <iostream>    void f(char c) {  char &p = c;  p = 'p';  std::cout << c << std::endl;  } |

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

| **Principles(s):** 1. Validating input data maps to Data Type because the user needs to input the correct data type for the input to be valid. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL52 |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-DCL52-a | Never qualify a reference type with ‘const’ or ‘volatile’ |
| Polyspace Bug Finder | R2023a | CERT C++:DCL52-CPP | Checks for:   * Const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| Clang | 3.9 |  | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do not depend on the order of evaluation for side effects** |
| --- | --- | --- |
| **Data Value** | EXP-050-CPP | The execution of unsequenced evaluations can overlap. |

| **Noncompliant Code** |
| --- |
| In this example, i is evaluated more than once in an unsequenced manner, so the behavior of the expression is undefined. |
| void f(int i, const int \*b) {    int a = i + b[++i];    // ...  } |

| **Compliant Code** |
| --- |
| This example is independent of order of evaluation of the operands and 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):** 1. Validating input data maps to data value because entering the correct value is part of the validation process. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wunsequenced | Can detect simple violations of this rule where path-sensitive analysis is not required |
| CodeSonar | 7.2p0 | LANG.STRUCT.SE.DEC  LANG.STRUCT.SE.INC | Side Effects in Expression with Decreement  Side Effects in Expression with Decrement |
| Coverity | V7.5.0 | EVALUATION\_ORDER | Can detect the specific instance where a statement contains multiple side effects on the same value with an undefined evaluation order because, with different compiler flags or different compilers or platforms, the statement may behave differently |
| ECLAIR | 1.2 | CC2.EXP30 | Fully implemented |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do not attempt to create a std::string from a null pointer** |
| --- | --- | --- |
| **String Correctness** | STR-051-CPP | Using a null pointer would result in dereferencing a null pointer. |

| **Noncompliant Code** |
| --- |
| A std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist. |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| The results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>    void f() {  const char \*tmpPtrVal = std::getenv("TMP");  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  if (!tmp.empty()) {  // ...  }  } |

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

| **Principles(s):** 1. Validating data input maps to String Correctness because the string entered needs to match the string stored. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Assert\_failure |  |
| CodeSonar | 7.3p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2023a | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Sanitize data passed to complex subsystems** |
| --- | --- | --- |
| **SQL Injection** | STR-002-C | String data passed to complex subsystems may contain special characters that can trigger commands or actions, resulting in a software vulnerability. As a result, it is necessary to sanitize all string data passed to complex subsystems so that the resulting string is innocuous in the context in which it will be interpreted. |

| **Noncompliant Code** |
| --- |
| The program connects to a database, prompts the user for a user ID and password, and hashes the password. Unfortunately, this code example permits a SQL injection attack because the string passed to prepare accepts unsanitized input arguments. |
| use DBI;    my $dbfile = "users.db";  my $dbh = DBI->connect("dbi:SQLite:dbname=$dbfile","","")  or die "Couldn't connect to database: " . DBI->errstr;  sub hash {  # hash the password  }    print "Enter your id: ";  my $userid = <STDIN>;  chomp $userid;  print "Enter your password: ";  my $password = <STDIN>;  chomp $password;  my $hashed\_password = hash( $password);    my $sth = $dbh->prepare("SELECT \* FROM Users WHERE userid = '$userid' AND password = '$hashed\_password'")  or die "Couldn't prepare statement: " . $dbh->errstr;  $sth->execute()  or die "Couldn't execute statement: " . $sth->errstr;    if (my @data = $sth->fetchrow\_array()) {  my $username = $data[1];  my $id = $data[2];  print "Access granted to user: $username ($userid)\n";  }    if ($sth->rows == 0) {  print "Invalid username / password. Access denied\n";  }    $sth->finish;  $dbh->disconnect; |

| **Compliant Code** |
| --- |
| The prepare() method properly escapes input strings, preventing SQL injection when used properly. This is an example of component-based sanitization. |
| # ... beginning of code    my $sth = $dbh->prepare("SELECT \* FROM Users WHERE userid = ? AND password = ?")    or die "Couldn't prepare statement: " . $dbh->errstr;  $sth->execute($userid, $hashed\_password)    or die "Couldn't execute statement: " . $sth->errstr;    # ... rest of code |

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

| **Principles(s):** 1, 7. Validation can be a way to prevent SQL Injection and attackers may use SQL injections to invoke unused functionality. Sanitizing data sent to other system is a way for to prevent sensitive data from being used. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.2p0 | IO.INJ.COMMAND  IO.INJ.FMT  IO.INJ.LDAP  IO.INJ.LIB  IO.INJ.SQL  IO.UT.LIB  IO.UT.PROC | Command injection  Format string injection  LDAP injection  Library injection  SQL injection  Untrusted Library Load  Untrusted Process Creation |
| Coverity | 6.5 | TAINTED\_STRING | Fully implemented |
| LDRA tool suite | 9.7.1 | 108 D, 109 D | Partially implemented |
| Parasoft C/C++test | 2022.2 | CERT\_C-STR02-a  CERT\_C-STR02-b  CERT\_C-STR02-c | Protect against command injection  Protect against file name injection  Protect against SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Provide placement new with properly aligned pointers to sufficient storage capacity** |
| --- | --- | --- |
| **Memory Protection** | MEM-054-CPP | Passing a pointer that has insufficient storage capacity for the object being constructed may result in initialization of memory outside of the bounds of the object being constructed, which results in undefined behavior. |

| **Noncompliant Code** |
| --- |
| A pointer to a short is passed to placement new, which is attempting to initialize a long. On architectures where sizeof (short) < sizeof (long), this results in undefined behavior. |
| #include <new>    void f() {    short s;    long \*lp = ::new (&s) long;  } |

| **Compliant Code** |
| --- |
| The long is constructed into a buffer of sufficient size and with suitable alignment. |
| #include <new>    void f() {    char c; // Used elsewhere in the function    std::aligned\_storage<sizeof(long), alignof(long)>::type buffer;    long \*lp = ::new (&buffer) long;      // ...  } |

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

| **Principles(s):** 10. Memory protection requires secure coding standard, such as passing a pointer that has sufficient storage capacity. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.2p0 | LANG.MEM.BO | Buffer Overrun |
| LDRA tool suite | 9.7.1 | 597 S | Enhanced Enforcement |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-MEM54-a  CERT\_CPP-MEM54-b | Do not pass a pointer that has insufficient storage capacity or that is not suitably aligned for the object being constructed to placement ‘new’  An overhead should be used when an array of objects is passed to the placement ‘new’ allocation function |
| Polyspace Bug  Finder | R2023a | CERT C++: MEM54-CPP | Checks for placement new used with insufficient storage or misaligned pointers (rule fully covered) |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Exception objects must be nothrow copy constructible** |
| --- | --- | --- |
| **Assertions** | ERR-060-CPP | If the copy constructor for the exception object type throws during the copy initialization, std::terminate() is called, which can result in possibly unexpected implementation-defined behavior. |

| **Noncompliant Code** |
| --- |
| S has a std::string data member, and the copy constructor for std::string is not declared noexcept, the implicitly-defined copy constructor for S is also not declared to be noexcept. In low-memory situations, the copy constructor for std::string may be unable to allocate sufficient memory to complete the copy operation, resuting in a std::bad\_alloc exception being thrown. |
| #include <exception>  #include <string>    class S : public std::exception {    std::string m;  public:    S(const char \*msg) : m(msg) {}      const char \*what() const noexcept override {      return m.c\_str();    }  };    void g() {    // If some condition doesn't hold...    throw S("Condition did not hold");  }    void f() {    try {      g();    } catch (S &s) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| A std::runtime\_error object is required to correctly handles an arbitrary-length error message that is exception safe and guarantees the copy constructor will not throw. |
| #include <stdexcept>  #include <type\_traits>    struct S : std::runtime\_error {    S(const char \*msg) : std::runtime\_error(msg) {}  };    static\_assert(std::is\_nothrow\_copy\_constructible<S>::value,                "S must be nothrow copy constructible");    void g() {    // If some condition doesn't hold...    throw S("Condition did not hold");  }    void f() {    try {      g();    } catch (S &s) {      // Handle error    }  } |

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

| **Principles(s):** 9. Effective Quality Assurance maps to Assertion because assertion tests can be effective in identifying and preventing vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | cert-err60-cpp | Checked by clang-tidy |
| Helix QAC | 2023.1 | C++3508 |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-ERR60-a  CERT\_CPP-ERR60-b | Exception objects must be nothrow copy constructible  An explicitly declared copy constructor for a class that inherits from ‘std::exception’ should have a non-throwing exception specification |
| PRQA QA-C++ | 4.4 | 3508 |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | ERR-051-CPP | Even if the exception cannot be gracefully recovered from, using the matching exception handler ensures that the stack will be properly unwound and provides an opportunity to gracefully manage external resources before terminating the process. |

| **Noncompliant Code** |
| --- |
| Neither f() nor main() catch exceptions are thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    f();  } |

| **Compliant Code** |
| --- |
| The main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| 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):** 9. Effective quality assurance maps to exceptions because throwing exceptions is an effective way to identify errors or vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | main-function-catch-all  early-catch-all | Partially checked |
| CodeSonar | 7.3p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions  Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not cast to an out-of-range enumeration value** |
| --- | --- | --- |
| Integer | INT-050-CPP | The arithmetic value being cast must be within the range of values the enumeration can represent to avoid operating on unspecified values. |

| **Noncompliant Code** |
| --- |
| The code attempts to check whether a given value is within the range of acceptable enumeration values, but it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| The code checks that the value can be represented by the enumeration type before performing the conversions to guarantee the conversion does not result in an unspecified value. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    if (intVar < First || intVar > 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):** 1. Validating data input maps to Integer because the arithmetic value being cast might not be within the range of the enumeration values. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Cast-integer-to-enum | Partially checked |
| CodeSonar | 7.3p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| RuleChecker | 22.10 | Cast-integer-to-enum | Partially checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Guarantee that container indices and iterators are within the valid range** |
| --- | --- | --- |
| Indices and Iterators | CTR-050-CPP | Indices and iterators going beyond the bounds of the array can produce unexpected results or failure. |

| **Noncompliant Code** |
| --- |
| The function performs a range check to ensure that pos does not exceed the upper bound of the array, specified by tableSize, but fails to check the lower bound. |
| #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** |
| --- |
| The parameter pos is declared as size\_t, which prevents the passing of negative arguments. |
| #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):** 10. Adopting a secure coding standard maps to Indices and Integers because going beyond a valid range produces undefined behavior leading to vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TO  LANG.MEM.TU  LANG.MEM.TBA  LANG.STRUCT.PBB  LANG.STRUCT.PPE  LANG.STRUCT.PARITH | Buffer overrun  Buffer underrun  Type overrun  Type underrun  Tainted buffer access  Pointer before beginning of object  Pointer past end of object  Pointer Arithmetic |
| LDRA tool suite | 9.7.1 | 45 D, 47 S, 476 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X | Partially implemented |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-CTR50-a | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2023a | CERT C++:CTR50-CPP | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Close files when they are no longer needed** |
| --- | --- | --- |
| Input Output Files | FIO-051-CPP | The performance of the system can be affected if too many files are open at the same time. |

| **Noncompliant Code** |
| --- |
| The underlying std::basic\_filebuf<T> object maintained by the object is 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** |
| --- |
| In this code, std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are 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):** 1. Validating data input maps to Input Output File because you need to check if there is a file to open and that it is actually being read, and you need to check if a file was created. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | ALLOC.LEAK | Leak |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Klockwork | 2023.1 | RH.LEAK |  |
| Polyspace Bug Finder | R2023a | CERT C++:FIO51-CPP | Checks for resource leak (rule partially covered) |

### 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 should take place in the verify and test process. It will be used for the enforcement of and compliance to the standards defined in this policy by providing quality assurance and adding to the defense in depth. The verify and test process includes vulnerability scanning, which can be turned into automated testing depending on the functions of programming that are involved.

### 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 |
| DCL-052-CPP | Low | Unlikely | Low | Low | 3 |
| EXP-050-CPP | Medium | Probable | Medium | Medium | 2 |
| STR-051-CPP | High | Likely | Medium | High | 1 |
| STR-002-C | High | Likely | Medium | High | 1 |
| MEM-054-CPP | High | Likely | Medium | High | 1 |
| ERR-060-CPP | Low | Probable | Medium | Low | 3 |
| ERR-051-CPP | Low | Probable | Medium | Low | 3 |
| INT-050-CPP | Medium | Unlikely | Medium | Low | 3 |
| CTR-050-CPP | High | Likely | High | Medium | 2 |
| FIO-051-CPP | Medium | Unlikely | Medium | Low | 3 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | It is the practice of encrypting stored data to prevent unauthorized access. This policy applies when an attacker is trying to gain access to a hard drive containing the stored data. |
| Encryption at flight | It is the practice of encrypting data moving over a network. This policy applies when using the open internet for transporting data. |
| Encryption in use | It is the practice of encrypting data when data is in use. This policy applies when an attacker is trying to obtain unauthorized access to the database, or when cloud infrastructure and applications are being used, or when endpoints are insecure. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | It is the process by which a user’s identity is validated by asking for credentials during user logins. This policy applies when someone wants to access the network. |
| Authorization | It is the process of determining what resources the user is allowed to access and the operations that can be performed and setting the user level access. This policy applies when there is an administrative site, and the administrator wants to limit files accessed by users or task they can perform. |
| Accounting | It is the process of monitoring and capturing events done by the user while accessing the network resources. This policy applies when a company wants to keep track of changes to the database, addition of new users, and which resources individuals or groups use. |

**\***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 |  |
| 2.0 | 03/19/2023 | Coding Standards Added | Vincent Garza |  |
| 3.0 | 04/09/2023 | Completed Template | Vincent Garza |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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