**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 | Validate input from all untrusted data sources. Proper input validation can help remove most software vulnerabilities. This does not mean you should not be careful of eternal data sources, user files and different network interfaces. |
| 1. Heed Compiler Warnings | You should compile code at the highest warning level possible for your set compiler and then remove warnings by adjusting the code correctly. You can use additional static and dynamic analysis tools to detect additional flaws. |
| 1. Architect and Design for Security Policies | When creating software architecture your software should be designed to implement and enforce security policies. |
| 1. Keep It Simple | Keep your designs simple and small as possible to avoid any unnecessary flaws. The more complex your design increase the likelihood that errors will be made in the implementation, configuration, and use. |
| 1. Default Deny | Base all access based on permissions and not on exclusions, what this means is everyone is denied until access is permitted. |
| 1. Adhere to the Principle of Least Privilege | This policy means you should execute with the least number of permissions needed to complete the job. When elevating privilege, you should do it for as short as possible to complete the job. This policy limits the attacker’s chances to execute code at elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data that is passed to complex subsystems such as command shells, databases, commercial off the shelf components. By sanitizing the subsystems prevents attackers from using different functionalities of these systems. |
| 1. Practice Defense in Depth | Manage risk by invoking multiple layers of defenses so that if one layer fails there are other layers that can prevent flaws in the system. By having multiple layers keeps your systems secure from additional threats and vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | Having good quality assurance techniques can help you identify and eliminate vulnerabilities. Different types of testing and source code audits should be conducted in order to have an effective quality assurance. |
| 1. Adopt a Secure Coding Standard | Develop and apply as secure coding standard for each programming language and platform. |

### 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-CCP] | Integers (INT) – Do not cast to an out-of-range enumeration value. |

| **Noncompliant Code** |
| --- |
| This code attempts to check whether a given value is within the range of acceptable enumeration values. |
| **enum** EnumType {    First,    Second,    Third  };    **void** f(**int** intVar) {    EnumType enumVar = **static\_cast**<EnumType>(intVar);    **if** (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| This compliant code checks if the value of the can be used by the enumeration before preforming the conversion to unsure the conversion does not result in an unexcepted result. |
| **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):** Validating Input Data – Attackers could use this to create a buffer overflow because unspecified values could happen if the code is not compliant. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Cast-integer-to-enum | Partially checked |
| Parasoft C/C++ Test | 2023.1 | CERT\_CPP\_INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| Polyspace Bug Finder | R2023b | CERT C++: INT50-CPP | Checks for casting to out-of-range enumeration value (rule fully covered) |
| CodeSonar | 8.0p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Characters and Strings (STR) – Range check element access |

| **Noncompliant Code** |
| --- |
| The value returns by get\_index() may be greater than the number of elements stored in the string. |
| #include <string>    **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| The compliant code checks the value returned by get\_index() is within the correct range before calling the operator. |
| #include <string>    **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");    std::**size\_t** i = get\_index();  **if** (i < s.length()) {      s[i] = '1';    } **else** {      // Handle error    }  } |

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

| **Principles(s):** Validate Input Data, Use Effective Quality Assurance Techniques. By ensuring the code is correct and the data is valid within the function ensures that the unchecked elements that can lead to execution of arbitrary code making a vulnerable process. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Assert\_failure |  |
| CodeSonar | 8.0p0 | 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 |
| Parasoft C/C++ Test | 2023.1 | CERT\_CPP-STR53-a | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2023b | CERT C++: STR53-CPP | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CCP] | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, 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 (or some other error occurs). |
| #include <cstdlib>  #include <string>    **void** f() {    std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, 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):**  Architect and Design for Security Policies - building code to prevent vulnerabilities.  Keep it simple - always applies as keeping code as lightweight as possible is best practice.  Dereferencing a null pointer is undefined behavior, typically abnormal program termination In some situations, however, dereferencing a null pointer can lead to the execution of arbitrary code |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Assert\_failure |  |
| Parasoft C/C++ Test | 2023.1 | CERT\_CPP\_DCL52-a | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2023b | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule partially covered). |
| CodeSonar | 8.0p0 | LANG.MEM.NPD | Null Pointer Dereference |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CCP] | Do not use pointer-to member operators to access nonexistent members |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, an anonymous local variable of type std:: unique\_lock is expected to lock and unlock the mutex m by virtue of RAII |
| **struct** B {  **virtual** ~B() = **default**;  };    **struct** D : B {  **virtual** ~D() = **default**;  **virtual** **void** g() { /\* ... \*/ }  };    **void** f() {    B \*b = **new** B;      // ...    **void** (B::\*gptr)() = **static\_cast**<**void**(B::\*)()>(&D::g);    (b->\*gptr)();  **delete** b;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the upcast is removed, rendering the initial code ill-formed and emphasizing the underlying problem that B::g() does not exist. This compliant solution assumes that the programmer's intention was to use the correct dynamic type for the underlying object. |
| **struct** B {  **virtual** ~B() = **default**;  };    **struct** D : B {  **virtual** ~D() = **default**;  **virtual** **void** g() { /\* ... \*/ }  };    **void** f() {    B \*b = **new** D; // Corrected the dynamic object type.      // ...  **void** (D::\*gptr)() = &D::g; // Moved static\_cast to the next line.    (**static\_cast**<D \*>(b)->\*gptr)();  **delete** b;  } |

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

| **Principles(s):** Heed Complier Warnings, adopt a secure coding standard – Ensure you are coding with the highest warnings to ensure code security to have the code properly connect to the right values. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.MEM.UVAR | Uninitialized Variable |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP55 |  |
| Parasoft C/C++ | 2023.1 | CERT\_CPP-OOP55-a | A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type |
| Polyspace Bug Finder | R2023b | CERT C++: OOP55-CPP | Checks for pointers to member accessing non-existent class members (rule fully covered). |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CCP] | Detect and handle memory allocation errors |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an array of int is created using ::operator new[](std::size\_t) and the results of the allocation are not checked. The function is marked as noexcept, so the caller assumes this function does not throw any exceptions. Because ::operator new[](std::size\_t) can throw an exception if the allocation fails, it could lead to abnormal termination of the program. |
| #include <cstring>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** **int**[size];    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

| **Compliant Code** |
| --- |
| When using std::nothrow, the new operator returns either a null pointer or a pointer to the allocated space. Always test the returned pointer to ensure it is not nullptr before referencing the pointer. This compliant solution handles the error condition appropriately when the returned pointer is nullptr. |
| #include <cstring>  #include <new>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** (std::**nothrow**) **int**[size];  **if** (!copy) {      // Handle error  **return**;    }    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

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

| **Principles(s):** Architect and Design for Security Policies, Keep it simple, Adopt a secure coding standard. Failing to detect allocation failures can lead to abnormal program termination and denial-of-service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| LDRA tool suite | 9.7.1 | 45 D | Partially implemented |
| Parasoft C/C++ | 2023.1 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| Polyspace Bug Finder | R2023b | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CCP] | Do not depend on the order of evaluation for side effects |

| **Noncompliant Code** |
| --- |
| In this noncompliant code 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** |
| --- |
| These examples are independent of the order of evaluation of the operands and can each be interpreted in only one way. |
| **void** f(**int** i, **const** **int** \*b) {    ++i;  **int** a = i + b[i];    // ...  }  **void** f(**int** i, **const** **int** \*b) {  **int** a = i + b[i + 1];    ++i;    // ...  } |

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

| **Principles(s):** Validating Input Data, Keep it simple, Adopt a secure coding standard. Attempting to modify an object in an unsequenced or indeterminately sequenced evaluation may cause that object to take on an unexpected value, which can lead to unexpected program behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.STRUCT.SE.DEC  LANG.STRUCT.SE.INC | Side Effects in Expression with Decrement  Side Effects in Expression with Increment |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-EXP50 |  |
| Parasoft C/C++ | 2023.1 | CERT\_CPP-EXP50-a  CERT\_CPP-EXP50-b  CERT\_CPP-EXP50-c  CERT\_CPP-EXP50-d  CERT\_CPP-EXP50-e  CERT\_CPP-EXP50-f | The value of an expression shall be the same under any order of evaluation that the standard permits  Don't write code that depends on the order of evaluation of function arguments  Don't write code that depends on the order of evaluation of function designator and function arguments  Don't write code that depends on the order of evaluation of expression that involves a function call  Between sequence points an object shall have its stored value modified at most once by the evaluation of an expression  Don't write code that depends on the order of evaluation of function calls |
| Polyspace Bug Finder | R2023b | CERT C++: EXP50-CPP | Checks for situations where expression value depends on order of evaluation (rule fully covered). |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CCP] | Do not abruptly terminate the program |

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

| **Compliant Code** |
| --- |
| In this compliant solution, f() handles all exceptions thrown by throwing\_func() and does not rethrow. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.  try {  throwing\_func();  } catch (...) {  // Handle error  }  }    int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  }    }  **return** cache[i];    }    **return** i > 0 ? i \* fact(i - 1) : 1;  } |

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

| **Principles(s):** Keep it simple, Use Effective Quality Assurance Techniques, Adopt a secure coding standard. Keeping a clean code allows you to avoid the program from having issues. Allowing the application to abnormally terminate can lead to resources not being freed, closed, and so on. It is frequently a vector for denial-of-service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| Parasoft C/C++ Test | 2023.1 | CERT\_CPP-ERR50-a  CERT\_CPP-ERR50-b  CERT\_CPP-ERR50-c  CERT\_CPP-ERR50-d  CERT\_CPP-ERR50-e  CERT\_CPP-ERR50-f  CERT\_CPP-ERR50-g  CERT\_CPP-ERR50-h  CERT\_CPP-ERR50-i  CERT\_CPP-ERR50-j  CERT\_CPP-ERR50-k  CERT\_CPP-ERR50-l  CERT\_CPP-ERR50-m  CERT\_CPP-ERR50-n | The execution of a function registered with 'std::atexit()' or 'std::at\_quick\_exit()' should not exit via an exception Never allow an exception to be thrown from a destructor, deallocation, and swap Do not throw from within destructor There should be at least one exception handler to catch all otherwise unhandled exceptions An empty throw (throw;) shall only be used in the compound-statement of a catch handler Exceptions shall be raised only after start-up and before termination of the program 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 Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) Function called in global or namespace scope shall not throw unhandled exceptions Always catch exceptions Properly define exit handlers The 'abort()' function from the 'stdlib.h' or 'cstdlib' library shall not be used Avoid throwing exceptions from functions that are declared not to throw The 'quick\_exit()' and '\_Exit()' functions from the 'stdlib.h' or 'cstdlib' library shall not be used |
| Polyspace Bug Finder | R2023b | CERT C++: ERR50-CPP | Checks for implicit call to terminate() function (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input/Output | [STD-008-CCP] | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::fstream object file is constructed. The constructor for std::fstream calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. Consequently, 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 compliant solution, 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):** Keep it simple- Failing to properly close files may allow an attacker to exhaust system resources and can increase the risk that data written into in-memory file buffers will not be flushed in the event of abnormal program termination. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2023.3 | DF4786, DF4787, DF4788 |  |
| CodeSonar | 8.0p0 | ALLOC.LEAK | Leak |
| Parasoft C/C++ Test | 2023.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Polyspace Bug Finder | R2023b | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Concurrency | [STD-009-CCP] | Do not destroy a mutex while it is locked |

| **Noncompliant Code** |
| --- |
| This noncompliant code example creates several threads that each invoke the do\_work() function, passing a unique number as an ID.  Unfortunately, this code contains a race condition, allowing the mutex to be destroyed while it is still owned, because start\_threads() may invoke the mutex's destructor before all of the threads have exited.. |
| #include <mutex>  #include <thread>    const size\_t maxThreads = 10;    void do\_work(size\_t i, std::mutex \*pm) {  std::lock\_guard<std::mutex> lk(\*pm);    // Access data protected by the lock.  }    void start\_threads() {  std::thread threads[maxThreads];  std::mutex m;    for (size\_t i = 0; i < maxThreads; ++i) {  threads[i] = std::thread(do\_work, i, &m);  }  } |

| **Compliant Code** |
| --- |
| This compliant solution eliminates the race condition by extending the lifetime of the mutex. |
| #include <mutex>  #include <thread>    const size\_t maxThreads = 10;    void do\_work(size\_t i, std::mutex \*pm) {  std::lock\_guard<std::mutex> lk(\*pm);    // Access data protected by the lock.  }    std::mutex m;    void start\_threads() {  std::thread threads[maxThreads];    for (size\_t i = 0; i < maxThreads; ++i) {  threads[i] = std::thread(do\_work, i, &m);  }  } |

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

| **Principles(s):** Architect and Design for Security Policies, Keep it simple, Use Effective Quality Assurance Techniques; Adopt a secure coding standard. Ensuring your coding is up to date and free can help prevent data corruption. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Klocwork | 2023.3 | CERT.CONC.MUTEX.DESTROY\_WHILE\_LOCKED |  |
| CodeSonar | 8.0p0 | CONCURRENCY.LOCALARG | Local Variable Passed to Thread |
| Parasoft C/C++ Test | 2023.1 | CERT\_CPP-CON50-a | Do not destroy another thread's mutex |
| Polyspace Bug Finder | R2023b | CERT C++: CON50-CPP | Checks for destruction of locked mutex (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-010-CCP] | Do not use an additive operator on an iterator if the result would overflow |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a random access iterator from a std::vector is used in an additive expression, but the resulting value could be outside the bounds of the container rather than a past-the-end value. |
| #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  for (auto i = c.begin(), e = i + 20; i != e; ++i) {  std::cout << \*i << std::endl;  }  } |

| **Compliant Code** |
| --- |
| This compliant solution assumes that the programmer's intention was to process up to 20 items in the container. Instead of assuming all containers will have 20 or more elements, the size of the container is used to determine the upper bound on the addition. |
| #include <algorithm>  #include <vector>    **void** f(**const** std::vector<**int**> &c) {  **const** std::vector<**int**>::size\_type maxSize = 20;  **for** (auto i = c.begin(), e = i + std::min(maxSize, c.size()); i != e; ++i) {      // ...    }  } |

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

| **Principles(s):** Validate Input Data, Keep it Simple. If adding or subtracting an integer to a pointer results in a reference to an element outside the array or one past the last element of the array object, the behavior is undefined but frequently leads to a buffer overflow or buffer underrun, which can often be exploited to run arbitrary code. Iterators and standard template library containers exhibit the same behavior and caveats as pointers and arrays. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2023.3 | DF3526, DF3527, DF3528, DF3529, DF3530, DF3531, DF3532, DF3533, DF3534 |  |
| LDRA tool suite | 9.7.1 | 567 S | Enhanced Enforcement |
| Parasoft C/C++ Test | 2023.1 | CERT\_CPP-CTR55-a | Do not add or subtract a constant with a value greater than one from an iterator |
| Polyspace Bug Finder | R2023b | CERT C++: CTR55-CPP | Checks for possible iterator overflows (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.

[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 | Medium | Likely | Medium | P4 | L3 |
| STD-002-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Probable | High | P6 | L2 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-007-CPP | Low | Probable | Medium | P4 | L2 |
| STD-008-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-009-CPP | Medium | Probable | High | P4 | L3 |
| STD-010-CPP | High | Likely | Medium | P18 | L1 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption for data at rest is the process of securely encoding data as it is written into storage and decrypting that data as it is pulled from storage for use. Using a symmetric encryption key when the data is written into storage to protect it from unauthorized access by anyone who does not have that key to decrypt the data from storage into use. It should be used anytime data / information is of any level of sensitivity and would cause harm if accessed by unauthorized actors |
| Encryption at flight | Encryption of data in - flight is the process of securely encoding data as it is being transmitted in some fashion. Depending on exactly how you will be transferring any data will determine how to apply encryption. When using a web browser always utilize secure protocols, when sending emails always encrypt before sending and use digital signatures |
| Encryption in use | Encryption of data in - use is the process of protecting data as it is utilized in memory, the main way of doing this is by utilizing password protected profiles as they protect the memory of each user for the data stored in memory for that profile could be used to compromise their data in rest / flight. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The process used to prove who a user is by, userID, passwords, possibly higher-level security such as secure tokens, CAC/PIN and other hardware credentials |
| Authorization | Once a user is authenticated, and allowed access to a system, they are granted specific access to parts of that system. Authorized access to certain drives, folders, programs, or data allowed by the system administrators. |
| Accounting | After authentication and authorization, it is always a good idea to monitor and record any transaction or anything that does happen. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 11/11/2023 | Coding Standards Added | Daniel Anderson |  |
| 1.2 | 12/2/2023 | Risk Assessment Added | Daniel Anderson |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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