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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | This is the process of testing any input that is received by the application. All input from outside the system is to be considered as being from an untrusted source. This ranges from strictly typing parameters up to and including using business logic to validate input. Validating input verifies that no malicious data is entering the system from outside sources. |
| 1. Heed Compiler Warnings | This process is used to compile code using the highest available warning level for the compiler. The goal of using the highest available warning level is for the developer to modify the code in such a way as to eliminate any warnings. This will drastically reduce any errors in the code and eliminate vulnerabilities that could lead to an exploit. |
| 1. Architect and Design for Security Policies | This process is used to design security within the system before beginning to write the code. This looks at how information controls and safeguards are put in place within the application. Using Architect and Design for security policies ensures that the confidentiality, integrity, and availability of data that exists within the application is protected. |
| 1. Keep It Simple | This process is a design principle used by developers so that they don’t overembellish the code. This means that developers should only write code that is necessary for both functionality and security without adding anything extra to it. Overembellishing code not only makes readability more complex but also allows vulnerabilities to creep into the extra code that creates the potential for exploits. |
| 1. Default Deny | This process is used to deny all access to the application, data, and resources by default. Only by the user entering their correct credentials will they be allowed access according to the application, its data, and its resources. This ensures that a malicious attack on the system does not give the attacker access to anything. This is a great complement to the principle of least privilege using defense in depth. |
| 1. Adhere to the Principle of Least Privilege | This process is used to limit a users’ access to the data, resources, and applications that they need to complete a required task. This means that users will only be allowed to access what is required for them to be able to complete their task. By using the principle of least privilege attackers that gain access to a users’ login will have minimal access to the system and not be allowed control to the entire application, its data, or its resources. |
| 1. Sanitize Data Sent to Other Systems | This process is used to check, clean, and filter data sent to other systems. This works by removing data that is compromising to the application, the user, and the company before sending that data outside of its own system. This reduces the risk of an attacker being able to exploit a vulnerability and gain access to confidential information. |
| 1. Practice Defense in Depth | This process is used to implement layers of monitoring and protection to reduce cybersecurity risks. No single layer of security exists that does not have exploitable holes, however different security layers have different holes in different places. Developers can use this to layer security implementations that complement each other leaving all holes covered by another layer. |
| 1. Use Effective Quality Assurance Techniques | This process is used to ensure a quality product at every stage of development, deployment, and maintenance. This can be done through pre-production, during production, pre-shipment, and post shipment inspections. The purpose of these inspections is to ensure that the product meets the standard at the current level before beginning on to the next level. This may be an iterative process until the product can meet the level of standards put in place such as meeting user requirements as well as passing security checks. |
| 1. Adopt a Secure Coding Standard | This process is used to create and implement a set of rules and guidelines to reduce vulnerabilities and errors during development. This can be done in many ways such as using a static code analyzer that enforces rules, security standards, and flags security violations. By doing this, developers can ensure that their code is both secure and up to standards while maintaining the fast pace that modern methodologies adhere to. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Pass an object of the correct type to va\_start |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the object passed to va\_start() will undergo a default argument promotion, which results in undefined behavior. |
| #include <cstdarg>    extern "C" void f(float a, ...) {  va\_list list;  va\_start(list, a);  // ...  va\_end(list);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, f() accepts a double instead of a float. |
| #include <cstdarg>    extern "C" void f(double a, ...) {  va\_list list;  va\_start(list, a);  // ...  va\_end(list);  } |

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

| **Principles(s):**  Adopt a Secure Coding Standard - implements a set of rules and guidelines to reduce vulnerabilities and errors during development. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wvarargs | Does not catch the violation in the third noncompliant code example (it is conditionally supported by Clang) |
| CodeSonar | 7.3p0 | BADMACRO.STDARG\_H | Use of <stdarg.h> Feature |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-EXP58-a | Use macros for variable arguments correctly |
| Polyspace Bug Finder | R2023a | CERT C++: EXP58-CPP | Checks for incorrect data types for second argument of va\_start (rule fully covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Use valid references, pointers, and iterators to reference elements of a container |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, pos is invalidated after the first call to insert(), and subsequent loop iterations have undefined behavior. |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  d.insert(pos, items[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, pos is assigned a valid iterator on each insertion, preventing undefined behavior. |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  pos = d.insert(pos, items[i] + 41.0);  }  } |

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

| **Principles(s):**  Adopt a Secure Coding Standard - implements a set of rules and guidelines to reduce vulnerabilities and errors during development. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | ALLOC.UAF | Use After Free |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-CTR51-a | Do not modify container while iterating over it |
| Polyspace Bug Finder | R2023a | CERT C++: CTR51-CPP | Checks for use of invalid iterator (rule partially covered). |
| Astrée | 22.10 | overflow\_upon\_dereference |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Use valid references, pointers, and iterators to reference elements of a basic\_string |

| **Noncompliant Code** |
| --- |
| This noncompliant code example copies input into a std::string, replacing semicolon (;) characters with spaces. This example is noncompliant because the iterator loc is invalidated after the first call to insert(). The behavior of subsequent calls to insert() is undefined. |
| #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) {  email.insert(loc, \*i != ';' ? \*i : ' ');  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the value of the iterator loc is updated as a result of each call to insert() so that the invalidated iterator is never accessed. The updated iterator is then incremented at the end of the 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):**  Adopt a Secure Coding Standard - implements a set of rules and guidelines to reduce vulnerabilities and errors during development. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | ALLOC.UAF | Use After Free |
| Helix QAC | 2023.1 | DF4746, DF4747, DF4748, DF4749 |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-STR52-a | Use valid references, pointers, and iterators to reference elements of a basic\_string |
| Polyspace Bug Finder | R2023a | CERT C++: STR52-CPP | Checks for use of invalid string iterator (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CLG] | Sanitize data passed to complex subsystems |

| **Noncompliant Code** |
| --- |
| Data sanitization requires an understanding of the data being passed and the capabilities of the subsystem. John Viega and Matt Messier provide an example of an application that inputs an email address to a buffer and then uses this string as an argument in a call to system() [Viega 2003]: |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| It is necessary to ensure that all valid data is accepted, while potentially dangerous data is rejected or sanitized. Doing so can be difficult when valid characters or sequences of characters also have special meaning to the subsystem and may involve validating the data against a grammar. In cases where there is no overlap, whitelisting can be used to eliminate dangerous characters from the data.  The whitelisting approach to data sanitization is to define a list of acceptable characters and remove any character that is not acceptable. The list of valid input values is typically a predictable, well-defined set of manageable size. This compliant solution, based on the tcp\_wrappers package written by Wietse Venema, shows the whitelisting approach: |
| static char ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  "1234567890\_-.@";  char user\_data[] = "Bad char 1:} Bad char 2:{";  char \*cp = user\_data; /\* Cursor into string \*/  const char \*end = user\_data + strlen( user\_data);  for (cp += strspn(cp, ok\_chars); cp != end; cp += strspn(cp, ok\_chars)) {  \*cp = '\_';  } |

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

| **Principles(s):**  Sanitize Data Sent to Other Systems - removes data that is compromising to the application, the user, and the company before sending that data outside of its own system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | 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** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CLG] | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):**  Adopt a Secure Coding Standard - implements a set of rules and guidelines to reduce vulnerabilities and errors during development. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | dangling\_pointer\_use | Supported  Astrée reports all accesses to freed allocated memory. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM30 | Detects memory accesses after its deallocation and double memory deallocations |
| CodeSonar | 7.3p0 | ALLOC.UAF | Use after free |
| Coverity | 2017.07 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |

#### Coding Standard 6

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

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

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

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

| **Principles(s):**  Use Effective Quality Assurance Techniques - ensures that the product meets the standard at the current level before beginning on to the next level. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 7.3p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

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

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

| **Compliant Code** |
| --- |
| In this compliant solution, the copy assignment operator provides the strong exception safety guarantee. The function allocates new storage for the copy before changing the state of the object. Only after the allocation succeeds does the function proceed to change the state of the object. In addition, by copying the array to the newly allocated storage before deallocating the existing array, the function avoids the test for self-assignment, which improves the performance of the code in the common case [Sutter 2004]. |
| #include <cstring>    class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }    IntArray(const IntArray& that); // nontrivial copy constructor    IntArray& operator=(const IntArray &rhs) {  int \*tmp = nullptr;  if (rhs.nElems) {  tmp = new int[rhs.nElems];  std::memcpy(tmp, rhs.array, rhs.nElems \* sizeof(\*array));  }  delete[] array;  array = tmp;  nElems = rhs.nElems;  return \*this;  }    // ...  }; |

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

| **Principles(s):**  Practice Defense in Depth - implements layers of monitoring and protection to reduce cybersecurity risks.  Adopt a Secure Coding Standard - implements a set of rules and guidelines to reduce vulnerabilities and errors during development.  Use Effective Quality Assurance Techniques- ensure that the product meets the standard at the current level before beginning on to the next level. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | ALLOC.LEAK | Leak |
| LDRA tool suite | 9.7.1 | 527 S, 56 D, 71 D | Partially implemented |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-ERR56-a  CERT\_CPP-ERR56-b | Always catch exceptions  Do not leave 'catch' blocks empty |
| Polyspace Bug Finder | R2023a | CERT C++: ERR56-CPP | Checks for exceptions violating class invariant (rule fully covered). |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Integers** | [STD-008-CPP] | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. On a two's complement system, the valid range of values that can be represented by EnumType are [0..3], so if a value outside of that range were passed to f(), the cast to EnumType would result in an unspecified value, and using that value within the if statement results in unspecified behavior. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);    if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| This compliant solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does this by restricting the converted value to one for which there is a specific enumerator 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):**  Adopt a Secure Coding Standard - implements a set of rules and guidelines to reduce vulnerabilities and errors during development. |
| --- |

**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** | **Name of Standard** |
| --- | --- | --- |
| **Input/Output** | [STD-009-CPP] | 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();  } file.seekg(0, std::ios::beg);  file >> str;  } |

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

| **Principles(s):**  Adopt a Secure Coding Standard - implements a set of rules and guidelines to reduce vulnerabilities and errors during development. |
| --- |

**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 |
| Helix QAC | 2023.1 | DF4786, DF4787, DF4788 |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Polyspace Bug Finder | R2023a | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Object Oriented Programming** | [STD-010-CPP] | Prefer special member functions and overloaded operators to C Standard Library functions |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a nontrivial class object is initialized by calling its default constructor but is later reinitialized to its default state using std::memset(), which does not properly reinitialize the object. Improper reinitialization leads to class invariants not holding in later uses of the object. |
| #include <cstring>  #include <iostream>    class C {  int scalingFactor;  int otherData;    public:  C() : scalingFactor(1) {}    void set\_other\_data(int i);  int f(int i) {  return i / scalingFactor;  }  // ...  };    void f() {  C c;    // ... Code that mutates c ...    // Reinitialize c to its default state  std::memset(&c, 0, sizeof(C));    std::cout << c.f(100) << std::endl;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the call to std::memset() is replaced with a default-initialized copy-and-swap operation called clear(). This operation ensures that the object is initialized to its default state properly, and it behaves properly for object types that have optimized assignment operators that fail to clear all data members of the object being assigned into. |
| #include <iostream>  #include <utility>    class C {  int scalingFactor;  int otherData;    public:  C() : scalingFactor(1) {}    void set\_other\_data(int i);  int f(int i) {  return i / scalingFactor;  }  // ...  };    template <typename T>  T& clear(T &o) {  using std::swap;  T empty;  swap(o, empty);  return o;  }    void f() {  C c;    // ... Code that mutates c ...    // Reinitialize c to its default state  clear(c);    std::cout << c.f(100) << std::endl;  } |

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

| **Principles(s):**  Adopt a Secure Coding Standard - implements a set of rules and guidelines to reduce vulnerabilities and errors during development. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | stdlib-use-ato  stdlib-use  stdlib-use-getenv  stdlib-use-system  include-time  stdlib-use-string-unbounded | Partially checked |
| CodeSonar | 7.3p0 | BADFUNC.MEMCMP  BADFUNC.MEMSET | Use of memcmp  Use of memset |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced Enforcement |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-OOP57-a  CERT\_CPP-OOP57-b | Do not initialize objects with a non-trivial class type using C standard library functions  Do not compare objects of nonstandard-layout class type with C standard library functions |

### 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 will be used to check and enforce compliance using analysis tools. Both static and dynamic analysis tools will be used so that the code conforms to the guidelines set forth in this security template. Some of these tools but not an exhaustive list has been supplied with each of the coding standards listed above. It is expected that developers incorporate all the principles and standards previously listed and that they ensure this by using the companies pre-approved tools to verify compliance or to correct any issues found which are not in compliance. As DevSecOps is an iterative process the developer is tasked with the responsibility of ensuring that they follow the steps shown above in an iterative fashion until their work can be labeled as being a well-polished product in a complete, functional, and secure state. Every step in the process should incorporate security from the design phase to the deployment and maintenance phase. By including security in every stage, we ensure the principle of don’t leave security to the end.

### 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 | Unlikely | Medium | P4 | L3 |
| [STD-002-CPP] | High | Probable | High | P6 | L2 |
| [STD-003-CPP] | High | Probable | High | P6 | L2 |
| [STD-004-CLG] | High | Likely | Medium | P18 | L1 |
| [STD-005-CLG] | High | Likely | Medium | P18 | L1 |
| [STD-006-CLG] | Low | Unlikely | High | P1 | L3 |
| [STD-007-CPP] | High | Likely | High | P9 | L2 |
| [STD-008-CPP] | Medium | Unlikely | Medium | P4 | L3 |
| [STD-009-CPP] | Medium | Unlikely | Medium | P4 | L3 |
| [STD-010-CPP] | High | Probable | High | P6 | L2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption in rest is the process of encrypting data when it is stored and not being used. This practice ensures that data is safe even when it is not being transferred or being used so that not even database administrators have access to view the raw data. |
| Encryption at flight | Encryption in flight is the process of encrypting data when it is being transferred (sent) somewhere. This practice ensures that the data is safe when being transferred out of the system so that malicious entities can’t intercept and read the raw data such as man in the middle attack. |
| Encryption in use | Encryption in use is the process of encrypting data when it is in use. This ensures that the data is safe even when it is actively in use so that an attacker cant gain access to the raw data if a system breach were to occur. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is the process to ensure every user’s identity is validated before allowing them access to the system through methods such as user logins and two step verification. |
| Authorization | This process is coupled with deny by default and least privilege access so that users have the minimum access to what is necessary for completing their task. When new users are added to the system then the principle of deny by default says they will have no access to anything within the system and according to the principle of least privilege access they will then only be given access to the bare minimum resources which allows them to complete their task. |
| Accounting | This process is used to log the users’ sessions’ statistics and usage information such as any changes to the database or access to files. |

**\***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 | 03/19/2023 | First Revision | Tyler Hollingsworth | [Insert text.] |
| [Insert text.] | 04/09/2023 | Final Revision | Tyler Hollingsworth | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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