**eGreen 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 | Data must be validated from untrusted sources including users, networks, files and devices. |
| 1. Heed Compiler Warnings | Code should be compiled with the at the highest warning level, and warnings should be eliminated by correcting code. |
| 1. Architect and Design for Security Policies | Software structure should facilitate enforcement of security policies, especially by isolating subsystems that require different privileges. |
| 1. Keep It Simple | Design software to be as simple as possible, as difficulty in ensuring security scales with complexity. |
| 1. Default Deny | Deny access by default, and identify circumstances where access is required before permitting access. |
| 1. Adhere to the Principle of Least Privilege | Deny privileges by default and grant privileges only where necessary, only for as long as necessary. |
| 1. Sanitize Data Sent to Other Systems | The calling system is responsible for the data sent to an invoked system. This data must be sanitized before sending to the invoked system to mitigate attacks on the invoked system. |
| 1. Practice Defense in Depth | Layer defensive strategies to reduce the likelihood of a single flaw becoming a vulnerability that can be exploited. |

|  |  |
| --- | --- |
| 1. Use Effective Quality Assurance Techniques | Develop a quality assurance program that incorporates techniques to eliminate vulnerabilities, including penetration tests, code audits, and external reviews. |
| 1. Adopt a Secure Coding Standard | Coding standards should be developed for the development language and target 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** | **Check Ranges of Data Types** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Prevent buffer overflows and wrapping by bounds-checking data types. |

| **Noncompliant Code** |
| --- |
| This code increments result without regard to the limits of the data type, resulting in potential buffer overflows. |
| template <typename T>  T add\_numbers(T const& start, T const& increment, unsigned long int const& steps)  {  T result = start;  for (unsigned long int i = 0; i < steps; ++i)  {  result += increment;  }  return result;  } |

| **Compliant Code** |
| --- |
| This code checks the limits of the data type before incrementing result, preventing buffer overflow. |
| template <typename T>  T add\_numbers(T const& start, T const& increment, unsigned long int const& steps)  {  T result = start;  overflow\_flag = false;  T range\_check\_value = 0;  for (unsigned long int i = 0; i < steps; ++i)  {  range\_check\_value = std::numeric\_limits<T>::max() - result;  // Precondition range check to prevent overflow  if (increment <= range\_check\_value) {  result += increment;  }  else {  overflow\_flag = true;  break;  }  }  return result;  } |

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

|  |
| --- |
| **Principles(s):**  1. **Validate Input Data**: trusting input from users or outside systems can lead to accidental or intentional buffer overflows. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| 3 - High | 3 - Likely | 1 - High | **P9** | **L2** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **integer-overflow** | Fully checked |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **unsigned overflow** | Exhaustively verified. |

### Coding Standard 2

| **Coding Standard** | **Label** | **Always Return a Value** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | For any value-returning function, a value must be returned from all possible paths. A value-returning function that fails to return a value is undefined behavior (see SEI CERT MSC52-CPP). |

| **Noncompliant Code** |
| --- |
| This function will not always return a value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| This function will always return a value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  return a;  } |

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

|  |
| --- |
| **Principles(s):**  3. **Architect and Design for Security Policies**: Preventing undefined behavior is achieved in part by properly designed functions that always return a value if they are value-returning. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| 2 - Medium | 2 - Probable | 2 - Medium | **P8** | **L2** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **return-implicit** | Fully checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | **-Wreturn-type** | Does not catch all instances of this rule, such as function-try-blocks |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | **LANG.STRUCT.MRS** | Missing return statement |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | **return-implicit** | Fully checked |

### Coding Standard 3

| **Coding Standard** | **Label** | **Sufficient Space for String Storage** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Ensure that destinations have sufficient space to store string data (see SEI CERT STR50-CPP). |

| **Noncompliant Code** |
| --- |
| This code uses a bounded array to store string input, which can easily be overflowed. |
| char user\_input[20];  std::cout << "Enter a value: ";  std::cin >> user\_input;  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl; |

| **Compliant Code** |
| --- |
| Using the std::string class ensures that enough memory is allocated for the string input. |
| std::string user\_input;  std::cout << "Enter a value: ";  std::cin >> user\_input;  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl; |

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

|  |
| --- |
| **Principles(s):**  1. **Validate Input Data**: while std::string does not validate input, strictly speaking, it can help ensure that enough memory is dynamically allocated for string input.  4. **Keep It Simple**: std::string is a simple solution to the problem of memory allocation for strings. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| 3 – High | 3 – Likely | 2 – Medium | **P18** | **L1** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description** |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2018 | [**NNTS.MIGHT**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [**NNTS.TAINTED**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |

### Coding Standard 4

| **Coding Standard** | **Label** | **SQL injection** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Prevent SQL injections by screening or sanitizing input, or ideally by parameterizing input (see SEI CERT STR02-C). |

| **Noncompliant Code** |
| --- |
| This code allows any query that does not return a SQL error. This allows the string variable sql to be injected with an attacker-supplied modified query. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  // clear any prior results  records.clear();  char\* error\_message;  if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  } |

| **Compliant Code** |
| --- |
| This code screens the input supplied in the sql string variable for the use of an or SQL operator, which would allow an attacker to supply a tautological SQL statement. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  // clear any prior results  records.clear();  // Copies and transforms sql command to all lowercase to make subsequent find()  // functionally case-insensitive  std::string localCopy(sql);  std::transform(localCopy.begin(), localCopy.end(), localCopy.begin(), ::tolower);  // If transformed sql command contains ' or ', returns before running query  std::size\_t pos = localCopy.find(" or ");  if (pos != std::string::npos) {  // String contains ' or '  std::cout << std::endl << "\*\*\*POSSIBLE SQL INJECTION DETECTED\*\*\*" << std::endl;  return false;  }  char\* error\_message;  if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  } |

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

|  |
| --- |
| **Principles(s):**  7. **Sanitize Data Sent to Other Systems**: As the calling system is responsible for sent data, preventing SQL injection is the responsibility of the system or component querying a SQL database. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| 3 – High | 3 – Likely | 2 – Medium | **P18** | **L1** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 |  | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | **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](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | **TAINTED\_STRING** | Fully implemented |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2020a | [CERT C: Rec. STR02-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.str02c.html) | Checks for:   * Execution of externally controlled command * Command executed from externally controlled path * Library loaded from externally controlled path   Rec. partially covered. |

### Coding Standard 5

| **Coding Standard** | **Label** | **Do Not Access Freed Memory** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Evaluating a pointer to deallocated memory (dangling pointer) is undefined behavior. Do not write to or read from deallocated memory (See SEI CERT MEM50-CPP) |

| **Noncompliant Code** |
| --- |
| The pointer variable s is being dereferenced after it has already been deallocated. Such a flaw could allow an attacker to run arbitrary code from the referenced memory location with the dangling pointer. |
| #include <new>  struct S {  void f();  };  void g() noexcept(false) {  S\* s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| The memory for the pointer is not deallocated until after it is no longer needed. |
| #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):**  3. **Architect and Design for Security Policies**: Deleting pointers that could still be needed is a failure to properly design a component or system. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| 3 – High | 3 – Likely | 2 - Medium | **P18** | **L1** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **dangling\_pointer\_use** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | v7.5.0 | **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 |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |

### Coding Standard 6

| **Coding Standard** | **Label** | **Assertions** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Use assertions to check assumptions during development (See SEI CERT DCL03-C). |

| **Noncompliant Code** |
| --- |
| Here a comment is left to make a note about an assumption. |
| int i = 0;  int j = 9;  for (; (i < 10); (++i, --j)) {  array1[i] = array2[j];  }  // j should be -1 |

| **Compliant Code** |
| --- |
| Here an assertion is used to check an assumption |
| int i = 0;  int j = 9;  for (; (i < 10); (++i, --j)) {  array1[i] = array2[j];  }  ASSERT(j == -1); |

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

|  |
| --- |
| **Principles(s):**  8. **Practice Defense in Depth**: In addition to other means of writing secure code, assertions can halt execution on programs that violate any assumptions. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| 1 – Low | 1 – Unlikely | 1 – High | **P1** | **L3** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |

### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all Exceptions** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Unhandled exceptions can lead to abnormal process termination, which could lead to denial-of-service attacks or improper or incomplete management of external resources during termination (see SEI CERT ERR-51-CPP). |

| **Noncompliant Code** |
| --- |
| Any exceptions thrown by throwing\_func() are left uncaught. |
| void throwing\_func() noexcept(false);  void f() {  throwing\_func();  }  int main() {  f();  } |

| **Compliant Code** |
| --- |
| Here, the main() function catches any exceptions thrown by calling f(). |
| 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):**  3. **Architect and Design for Security Policies**: Handling exceptions is a key feature of secure code. Design should allow for proper resource management in the even of unexpected behavior. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| 1 – Low | 2 – Probable | 2 – Medium | **P4** | **L3** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **main-function-catch-all early-catch-all** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 6.9.0 | **CertC++-ERR51** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **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** | **Use Correct Allocation/Deallocation Pairings** |
| --- | --- | --- |
| Memory Protection | STD-008-CPP | Pointers passed to an incorrect deallocation function can result in undefined behavior (see SEI CERT MEM51-CPP). |

| **Noncompliant Code** |
| --- |
| Placement new is used to construct the s1 pointer. delete cannot be used to deallocate memory for placement new |
| #include <iostream>  struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S\* s1 = new (&space) S;  // ...  delete s1;  } |

| **Compliant Code** |
| --- |
| Instead of delete, ~S() (the destructor for s1) is explicitly called. |
| #include <iostream>  struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S\* s1 = new (&space) S;  // ...  s1->~S();  } |

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

|  |
| --- |
| **Principles(s):**  3. **Architect and Design for Security Policies**: Mating memory allocation functions properly is a key design consideration. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| 3 – High | 3 – Likely | 2 - Medium | **P18** | **L1** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **invalid\_dynamic\_memory\_allocation dangling\_pointer\_use** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 6.9.0 | **CertC++-MEM51** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks -Wmismatched-new-delete clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | **ALLOC.FNH ALLOC.DF ALLOC.TM** | Free non-heap variable Double free Type mismatch |

### Coding Standard 9

| **Coding Standard** | **Label** | **Range Check String Element Access** |
| --- | --- | --- |
| Strings | STD-009-CPP | Prevent passing out-of-range values to std::string::operator[]() (see SEI CERT STR53-CPP). |

| **Noncompliant Code** |
| --- |
| The value from get\_index() could potentially exceed the number of elements in the string. This is undefined behavior. |
| #include <string>  extern std::size\_t get\_index();  void f() {  std::string s("01234567");  s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| This revision uses at(), which will throw an exception if the element being accessed is out of range. |
| #include <stdexcept>  #include <string>  extern std::size\_t get\_index();  void f() {  std::string s("01234567");  try {  s.at(get\_index()) = '1';  }  catch (std::out\_of\_range&) {  // Handle error  }  } |

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

|  |
| --- |
| **Principles(s):**  3. **Architect and Design for Security Policies**: Good design utilizes safe features such as those provided by std::class |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| 3- High | 1 – Unlikely | 2 – Medium | **P6** | **L2** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **assert\_failure** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | **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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the valid range |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2020a | [CERT C++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) | 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** | **Do Not Use Unrelated Smart Pointers to Store the Same Pointer** |
| --- | --- | --- |
| Memory Protection | STD-010-CPP | When unrelated smart pointers reference the same pointer, destruction of any of the smart pointer objects deletes the referenced pointer value (see SEI CERT MEM56-CPP). |

| **Noncompliant Code** |
| --- |
| Two unrelated smart pointers are created for the same pointer value. When p2 is destroyed, pointer i is deleted. If p1 is subsequently destroyed, the same pointer is deleted again, which results in a double-free. |
| #include <memory>  void f() {  int\* i = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i);  } |

| **Compliant Code** |
| --- |
| The copy construction makes these smart pointers related. When p2 is destroyed, i is *not* deleted, because the counter for the shared pointer is not zero. |
| #include <memory>  void f() {  std::shared\_ptr<int> p1 = std::make\_shared<int>();  std::shared\_ptr<int> p2(p1);  } |

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

|  |
| --- |
| **Principles(s):**  3. **Architect and Design for Security Policies**: Good design utilizes safe features such as those provided by smart pointer classes. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| 3- High | 3 – Likely | 2 – Medium | **P18** | **L1** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **dangling\_pointer\_use** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-MEM56-a** | Do not store an already-owned pointer value in an unrelated smart pointer |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.07 | [**V1006**](https://www.viva64.com/en/w/v1006/) |  |

## 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.

As part of the of the Verify and Test phase of pre-production, automatic use of static testing tools will be employed. At a minimum, all code containing changes will be scanned before being committed. Various tools and their uses/efficacy are listed in the Automation sections of each coding standard. As part of the Transition and health Check phase of production, dynamic testing tools will be used to scan new code changes for vulnerabilities as they run.

## 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 | Likely | High | P9 | 2 |
| STD-002-CPP | Medium | Probable | Medium | P8 | 2 |
| STD-003-CPP | High | Likely | Medium | P18 | 1 |
| STD-004-CPP | High | Likely | Medium | P18 | 1 |
| STD-005-CPP | High | Likely | Medium | P18 | 1 |
| STD-006-CPP | Low | Unlikely | High | P1 | 3 |
| STD-007-CPP | Low | Probable | Medium | P4 | 3 |
| STD-008-CPP | High | Likely | Medium | P18 | 1 |
| STD-009-CPP | High | Unlikely | Medium | P6 | 2 |
| STD-010-CPP | High | Likely | Medium | P18 | 1 |

## 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 at rest | Data at rest is arguably the least vulnerable state. Encryption at rest can offer protection in the case of theft of physical media on which the data is stored. Methods such as full-disk encryption, file encryption, and database encryption can protect data at rest, and should be used to protect any data deemed sensitive or confidential. |
| Encryption in flight | Data in flight is considerably more vulnerable than data at rest. Encryption and decryption are implemented on opposite sides of a transmission. This can include PKI (Public Key Infrastructure) email encryption, MFT (Managed File Transfer) alternatives to insecure protocols like FTP, or Data Leak Prevention technologies which detect if attempts are being made to send data outside of an organization. |
| Encryption in use | Data in use essentially describes data loaded to memory or CPU registers and caches. This is data’s most vulnerable state, as at-rest and in-flight encryption are necessarily decrypted for data to enter this state. Full memory encryption is currently in development by the major CPU vendors, but not ready for mainstream use. Digital Rights Management is one current technology that is embedded in each file and can protect data after it has been decrypted from at-rest and in-flight states. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is employed to determine that each user of a system or component is positively identified. This is the most basic level of the AAA framework, and prerequisite for proper authorization and accounting. Where password authentication is employed, passwords will be hashed and salted using **scrypt** or **bcrypt** algorithms. |
| Authorization | Authorization is used to grant or deny permissions to authenticated users. Where appropriate, user and group policies will be developed and used to limit functionality only to what is needed on a per-user basis (observing the principle of least privilege). |
| Accounting | Accounting will measure and record usage information of authenticated and authorized users. Necessary information will be kept in logs, enabling audits of access and attempted access, to facilitate the Monitor and Detect phase of production. |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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