**Green Pace Developer: Security Policy Guide – Vitalie Cucuta**



# 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 | All input data from untrusted sources should be validated to mitigate software vulnerabilities. Ensure that input data is conforms to the type and numeric range requirement as well as to input variants within the valid program input. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. The use of static and dynamic analysis tools can be used to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | All system architecture and design should be created with security policies implemented and enforced, such as setting privileges. |
| 1. Keep It Simple | Keep the design as simple and small as possible because complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. The more complex the design the more difficult to ensure security. |
| 1. Default Deny | Instead of trying to exclude access, it is better to set permission on who can have access. Deny access by default but identify conditions under which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least set of privileges necessary to complete the job. If an elevated permission is required to complete a task, allow access only for as long as it is necessary to complete that task. Doing so will reduce the risk of an attacker executing malicious code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | The calling process must sanitize the data before it is sent to the invoked system. This is different from input validation because the complex subsystem being invoked will not understand the context of the call, so it is the job of the calling system to sanitize the data. |
| 1. Practice Defense in Depth | This cybersecurity strategy protects the system by using several layered, redundant defenses to protect itself from a variety of threats such as cyber-attacks, cyber espionage, and ransomware attacks. The more layers there are, the harder it is to break through a systems defense. |
| 1. Use Effective Quality Assurance Techniques | Ensure the use of good QA techniques such as fuzz testing, penetration testing, and source code audits to identify and eliminate vulnerabilities. This is recommended to be done through independent security reviews since external reviewers bring an independent, unbiased perspective to the system. |
| 1. Adopt a Secure Coding Standard | Coding standards should be developed based on the target developmental language used 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** | **Numeric Overflow and Underflow Regulation** |
| --- | --- | --- |
| **Data Type** | [STD-001-CLG] | [reference INT30-C] Ensure that all data types are validated and protected from numeric overflow and underflow when data is being input. |

| **Noncompliant Code** |
| --- |
| Here is a piece of code that doesn’t account for any numeric overflow. After 5 steps of the for loop in this code, the variable will overflow. It will still output a return value, but it will not be correct. |
| 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 is a compliant version of the same code. Using the enable\_if command we regulate whether the variable is an integral or floating value, then execute code inside the block. If there is overflow, it will throw an exception with a message notifying the user that an overflow occurred. |
| // use SFINAE to enable function if integrals are used  template<class T> typename std::enable\_if<std::is\_integral<T>::value, T>::type add\_numbers(T const& start, T const& increment, unsigned long int const& steps)  {  // set result to start value passed  T result = start;  // set auto variable to hold max value of passed type, calculated at compile time  constexpr auto max\_numeric\_limit = std::numeric\_limits<T>::max();    // variable to hold value of max - result to compare it to increment  T left\_to\_max{};  // loop increments based on number of steps passed  for (unsigned long i = 0; i < steps; ++i)  {  // get new value each iteration  left\_to\_max = max\_numeric\_limit - result;  // catch overflow and throw exception if condition met  if (left\_to\_max < increment)  {  throw std::overflow\_error("OVERFLOW!");  }  // being here means no overflow. continue.  result += increment;  }  // done  return result;  } |

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

| **Principles(s):** Validate Input Data – any data passed by the user must be validated so that it does not cause overflow or underflow of data. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | 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 |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | CertC-INT30 | Implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW**  **ALLOC.SIZE.SUBUFLOW**  **MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | INTEGER\_OVERFLOW | Implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Return a Value from All Exit Paths** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | [reference MSC52-CPP] A value-returning function must return a value from all the code paths in the function or risk undefined behavior. |

| **Noncompliant Code** |
| --- |
| Here we see a value-returning function that only returns a value from the try block and no return value is given in the catch block and will result in undefined behavior if an exception is thrown. |
| #include <vector>  std::size\_t f(std::vector<int>& v, std::size\_t s) try {  v.resize(s);  return s;  }  catch (...) {  } |

| **Compliant Code** |
| --- |
| The compliant code here returns a value from the catch block as well so that the correct value is returned when an exception occurs. |
| #include <vector>  std::size\_t f(std::vector<int>& v, std::size\_t s) try {  v.resize(s);  return s;  }  catch (...) {  return 0;  } |

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

| **Principles(s):** Architect and Design for Security Policies – ensuring that all value-returning functions return a value will prevent undefined behavior that can pose security risks. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | 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 |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 2 D, 36 S | Fully implemented |
| [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) | 6.2p0 | LANG.STRUCT.MRS | Missing return statement |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee Storage for Strings has Sufficient Space** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | [reference STR50-CPP] Ensure that the destination of the data has sufficient size to hold value. When copying data to a pre-determined buffer, it will cause buffer overflow. |

| **Noncompliant Code** |
| --- |
| This code block shows a char buffer for a password. If there are more than the allowed characters in the buffer, the code will overflow but will also make this code susceptible to a hack. |
| int main()  {  std::cout << "Buffer Overflow Example" << std::endl;  const std::string account\_number = "CharlieBrown42";    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** |
| --- |
| Use the C++ std::basic\_string template instead of the bounded array to guard against buffer overflow. |
| int main()  {  std::cout << "Buffer Overflow Example" << std::endl;  const std::string account\_number = "CharlieBrown42";    // instead of using a fixed char input amount, change to C++  // basic\_string class  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):** Validate Input Data – dynamically allocating memory using std::string ensures enough memory is allocated.  Keep It Simple – dynamically allocating memory using std::string is a simple solution to ensuring there is enough space allocated. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.2p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type Overrun |
| [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 |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | CERT C++: STR50-CPP | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation   Rule partially covered. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL Injection** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CLG] | [reference STR02-C] Untrusted data may maliciously alter an SQL query and result in information leaks or data modification. Sanitization and validation can be implemented to prevent this. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code we see a code that will permit an SQL injection because it permits an unsanitized argument for the sql parameter. |
| 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** |
| --- |
| The compliant code will check if an SQL injection has been passed and if there is one, it will immediately throw an exception and exit the program before sensitive information can be attained. |
| bool run\_query(sqlite3\* db, const std::string& sql,  std::vector< user\_record >& records)  {  // clear any prior results  records.clear();  // create local copy of sql string parameter  std::string localSQL(sql);  // transform the sql command to be case sensitive  std::transform(localSQL.begin(), localSQL.end(),  localSQL.begin(), ::tolower);  // find if sql string contains ' or '  std::size\_t findOr = localSQL.find(" or ");  // if sql command contains ' or ', throw exception  if (findOr != std::string::npos)  {  throw std::exception("SQL INJECTION DETECTED");  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):** Sanitize Data Sent to Other Systems – 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** |
| --- | --- | --- | --- | --- |
| High | Likely | 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) | 6.2p0 | IO.INJ.COMMAND  IO.INJ.FMT  IO.INJ.LDAP  IO.INJ.LIB  IO.INJ.SQL  IO.UT.LIB  IO.UT.PROC | Command injection Format string injection LDAP injection Library injection SQL injection Untrusted Library Load Untrusted Process Creation |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | TAINTED\_STRING | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 108 D, 109 D | Partially implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Explicitly Construct and Destruct Manually Managed Objects** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | [reference MEM53-CPP] When manually managing the lifetime of an object, there must be a constructor which initiates the lifetime and a destructor with terminates the lifetime. |

| **Noncompliant Code** |
| --- |
| This noncompliant code has a custom container class which allocates storage for arbitrary element types. The copy\_elements() function is only called but does not explicitly call the default constructor for any additional elements causing undefined behavior once one of the additional elements are accessed through the operator[]() function. |
| #include <memory>  template <typename T, typename Alloc = std::allocator<T>>  class Container {  T\* underlyingStorage;  size\_t numElements;  void copy\_elements(T\* from, T\* to, size\_t count);  public:  void reserve(size\_t count) {  if (count > numElements) {  Alloc alloc;  T\* p = alloc.allocate(count); // Throws on failure  try {  copy\_elements(underlyingStorage, p, numElements);  }  catch (...) {  alloc.deallocate(p, count);  throw;  }  underlyingStorage = p;  }  numElements = count;  }  T& operator[](size\_t idx) { return underlyingStorage[idx]; }  const T& operator[](size\_t idx) const { return underlyingStorage[idx]; }  }; |

| **Compliant Code** |
| --- |
| The compliant solution shows all elements being properly initialized by explicitly calling copy or default constructors. |
| #include <memory>  template <typename T, typename Alloc = std::allocator<T>>  class Container {  T\* underlyingStorage;  size\_t numElements;  void copy\_elements(T\* from, T\* to, size\_t count);  public:  void reserve(size\_t count) {  if (count > numElements) {  Alloc alloc;  T\* p = alloc.allocate(count); // Throws on failure  try {  copy\_elements(underlyingStorage, p, numElements);  for (size\_t i = numElements; i < count; ++i) {  alloc.construct(&p[i]);  }  }  catch (...) {  alloc.deallocate(p, count);  throw;  }  underlyingStorage = p;  }  numElements = count;  }  T& operator[](size\_t idx) { return underlyingStorage[idx]; }  const T& operator[](size\_t idx) const { return underlyingStorage[idx]; }  }; |

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

| **Principles(s):** Architect and Design for Security Policies – constructor and destructor statements must be called and memory managed to prevent undefined behavior which could put security at risk. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | C++4761, C++4762, C++4766, C++4767 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | CERT\_CPP-MEM53-a | Do not invoke malloc/realloc for objects having constructors |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.16 | V630, V749 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Incorporate Diagnostic Tests Using Assertions** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | [reference MSC11-C] Use assertions throughout the code when debugging to protect against incorrect programmer assumptions. |

| **Noncompliant Code** |
| --- |
| The noncompliant code here does not check that the value of the variable is correct but assumes it will return the intended value. |
| int addition(int num1, int num2)  {  int sum = num1 + num2;    return sum;  }  int main()  {  int firstNum = 15;  int secondNum = 5;    //will return 20  std::cout << addition(firstNum, secondNum) << std::endl;  } |

| **Compliant Code** |
| --- |
| The compliant code would instead use the assert function to check that the value assumed to be returned is the one being returned by the function. |
| int addition(int num1, int num2)  {  int sum = num1 + num2;    return sum;  }  int main()  {  int firstNum = 15;  int secondNum = 5;  std::cout << addition(firstNum, secondNum) << std::endl;  // check if value returned is assumed value  assert(addition(firstNum, secondNum) == 20);  } |

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

| **Principles(s):** Practice Defense in Depth – assertions can halt execution when assumption is violated. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | ASSERT\_SIDE\_EFFECT | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | CERT\_C-MSC11-a | Assert liberally to document internal assumptions and invariants |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all Exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | [reference ERR51-CPP] All exceptions that are thrown must be caught by a matching exception handler. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the thread\_start() function does not catch if any exception is thrown by the throwing\_funct() function. If an exception occurs, std::terminate() will be called. |
| #include <thread>  void throwing\_func() noexcept(false);  void thread\_start() {  throwing\_func();  }  void f() {  std::thread t(thread\_start);  t.join();  } |

| **Compliant Code** |
| --- |
| The compliant code will incorporate a try/catch block to catch the exception when it occurs and terminating the thread normally. |
| #include <thread>  void throwing\_func() noexcept(false);  void thread\_start(void) {  try {  throwing\_func();  }  catch (...) {  // Handle error  }  }  void f() {  std::thread t(thread\_start);  t.join();  } |

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

| **Principles(s):** Architect and Design for Security Policies – ensuring that all exception are caught with an exception handler mitigates unexpected behavior.  Adopt a Secure Coding Standard – catching exceptions is a key part of secure coding. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | 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 |
| [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) | 2021.2 | 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 |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Range Check Element Access** |
| --- | --- | --- |
| **String Range** | [STD-008-CPP] | [reference STR53-CPP] Ensure that out of range values are not passed to a std::string::operator[] () |

| **Noncompliant Code** |
| --- |
| This noncompliant code asks for the index value in a string but the value returned 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 uses the std::basic\_string::at() function with a try/catch block which will throw a std::out\_of\_range exception if the position is greater than or equal to the number of elements stored in the string. |
| #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):** ValidateInput Data – ensuring user input is valid to avoid out of range value will mitigate undefined behavior.  Architect and Design for Security Policies – using try/catch blocks to check for exceptions mitigates unexpected behavior.  Adopt a Secure Coding Standard – catching exceptions is a key part of secure coding. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | 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) | 6.2p0 | 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) | 2021.2 | 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) | R2021b | 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 9

| **Coding Standard** | **Label** | **Detect Errors When Converting String to Number** |
| --- | --- | --- |
| **Conversion** | [STD-009-CPP] | [reference ERR62-CPP] When parsing an integer or floating-point number from a string, detect and address error condition when a string-to-number conversion is performed a formatted input stream such as std::istream. |

| **Noncompliant Code** |
| --- |
| This noncompliant code converts multiple numeric values from standard input stream and if the text received cannot be converted into a number value that is represented by int, the resulting value may be unexpected. |
| #include <iostream>  void f() {  int i, j;  std::cin >> i >> j;  // ...  } |

| **Compliant Code** |
| --- |
| The compliant code enables exceptions that can be thrown if conversion failure occurs. The standard input stream is tested for validity before reading the next value in the sequence so that error recovery occurs on a per-value basis. |
| #include <iostream>  #include <limits>  void f() {  int i;  std::cin >> i;  if (std::cin.fail()) {  // Handle failure to convert the value.  std::cin.clear();  std::cin.ignore(std::numeric\_limits<std::streamsize>::max(),  ' ');  }  int j;  std::cin >> j;  if (std::cin.fail()) {  std::cin.clear();  std::cin.ignore(std::numeric\_limits<std::streamsize>::max(),  ' ');  }  // ...  } |

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

| **Principles(s):** ValidateInput Data – ensuring user input is valid before moving on to the rest of the code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-err34-c | Checked by clang-tidy; only identifies use of unsafe C Standard Library functions corresponding to ERR34-C |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.2p0 | BADFUNC.ATOF  BADFUNC.ATOI  BADFUNC.ATOF  BADFUNC.ATOF | Use of atof  Use of atoi  Use of atol  Use of atoll |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | C++3161 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | CERT\_CPP-ERR62-a | The library functions atof, atoi, and atoll from library stdlib.h shall not be used |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do Not Read Uninitialized Memory** |
| --- | --- | --- |
| **Expressions** | [STD-010-CPP] | [reference EXP53-CPP] Ensure that all variables are initialized before they are read because local, automatic variables assume unexpected values otherwise. |

| **Noncompliant Code** |
| --- |
| This noncompliant code is attempting to print a local variable that has been uninitialized which will result in undefined behavior. |
| #include <iostream>  void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| The compliant solution shows the variable being initialized prior to printing the value. |
| #include <iostream>  void f() {  int i = 0;  std::cout << i;  } |

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

| **Principles(s):** Architect and Design for Security Policies – best practice is to always initialize a variable before they are read to avoid unexpected behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | uninitialized-read | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wuninitialized  clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated-memory |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.2p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local  Uninitialized variable |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially implemented |

### 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.

The most logical and effective phase that these standards will be enforced would be in the Verify and Test phase of pre-production. Any enforcement of and compliance to the standards in the production phase would be too late and any errors or flaws in the code would be released into production before being fixed. Using the static testing tools listed within each standard we can scan for vulnerabilities, test for functionality, compliancy, and security. If changes need to be made, we can stay within the pre-production phase to assess and plan how to make the changes, design the changes, build the changes then verify and test the changes have taken effect and there are no vulnerabilities. Once the code is compliant, we can move into the Transition and Health Check phase of production so that we can once again check the code once it is deployed. Once we can confirm that the code is compliant once deployed, we can move further into production.

### 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-CLG | High | Likely | High | P9 | L2 |
| STD-002-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CLG | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CLG | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | High | Probable | Medium | P12 | L1 |

### Create Policies for Encryption and Triple A

Include all three types of encryptions (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 at rest protects your data wherever you’ve stored it, whether that’s on your hard drive or in the cloud. It’s like storing your data in a vault, even if you steal the entire vault, you won’t be able to know what’s inside. Data at rest can be encrypted through strong encryption methods such as AES or RSA and many operating systems come with built-in full disk encryption. These encryptions should stay in place even if access controls fail. Encryption at rest is important in protecting your data even if the physical device on which that data is stored on is lost or stolen. |
| Encryption at flight | Encryption at flight is the process of encrypting data while it is transmitted. Data can be protected using SCP (Secure Copy Protocol) and SFTP (Secure File Transfer Protocol). These encrypt data through the SSH protocol to secure communication. The data is encrypted before being sent and decrypted once received. It is important to secure data at flight because if sensitive files are intercepted while being moved from one place to another then you risk this sensitive information leaking to a bad actor. |
| Encryption in use | Encryption in use encryption of data at its most vulnerable state, when it’s being read, process or modified. This is because this data is directly accessible to an individual, so it is much more susceptible to attack or just human error. This type of encryption is done through full memory encryption, a very new approach. AMDs Secure Memory Encryption and Intel’s MKTME have successfully been able to do this using new keys generated in the OS kernel to protect application data. One of the best ways to secure data in use is to restrict access by user role, limiting system access to only those who need it, i.e., following the principle of least privilege. Being able to protect data at its most vulnerable state is arguably the most important type of encryption policy. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the act of confirming the truth of an attribute of a single piece of data claimed true by an entity. Most often this is seen in the form of user logins but there is different type of credential that can be used for this. Again, the most common is static passwords, but one-time password, digital certificates, and biometric credentials are all types of authentications. Multi-Factor Authentication is a very reliable Authentication policy and makes it difficult for someone to authenticate as another person. Authentication is the first barrier that must be passed and protects the data within a device or server even if login credentials of the user are compromised. |
| Authorization | Authorization is the function of specifying access rights/privileges to resources related to the user. User and group policies will be developed so that users have access to only as much resources as they need to do the job which they need to do, observing the principle of least privilege. This policy goes hand in hand with the Encryption in Use policy. |
| Accounting | Accounting is the process that keeps track of a user’s activity while attached to a system. It’s essentially a paper trail of all the resources accessed, time spent in each resource, and how much data has been transferred while in the use of the user. This is also known as event logging and will be utilized by the system. Enforcing this policy will help to keep track of trends while also detecting breaches and forensic investigations because being able to trace back to events leading up to a security incident can prove to be a valuable piece of information that will help to prevent it in the future. |

**\***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 | 01/22/2022 | Create coding standards | Vitalie Cucuta |  |
| 1.2 | 02/09/2022 | Map principles, updating automatic detection, automation and updating encryption and Triple A framework policies. | Vitalie Cucuta |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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