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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_heading=h.gjdgxs)

[Purpose 2](#_heading=h.30j0zll)

[Scope 2](#_heading=h.1fob9te)

[Module Three Milestone 2](#_heading=h.3znysh7)

[Ten Core Security Principles 2](#_heading=h.4d34og8)

[C/C++ Ten Coding Standards](#_heading=h.2s8eyo1) 3

[Coding Standard 1](#_heading=h.3rdcrjn) 4

[Coding Standard 2](#_heading=h.26in1rg) 6

[Coding Standard 3](#_heading=h.lnxbz9) 8

[Coding Standard 4](#_heading=h.35nkun2) 10

[Coding Standard 5](#_heading=h.1ksv4uv) 12

[Coding Standard 6](#_heading=h.2jxsxqh) 14

[Coding Standard 7 1](#_heading=h.z337ya)6

[Coding Standard 8](#_heading=h.28h4qwu) 18

[Coding Standard 9](#_heading=h.nmf14n) 20

[Coding Standard 10](#_heading=h.37m2jsg) 22

[Defense-in-Depth Illustration](#_heading=h.1mrcu09) 24

[Project One](#_heading=h.46r0co2) 24

[1.](#_heading=h.2lwamvv) Revise the C/C++ Standards 24

[2.](#_heading=h.111kx3o) Risk Assessment 24

[3.](#_heading=h.3l18frh) Automated Detection 24

[4.](#_heading=h.206ipza) Automation 24

[5.](#_heading=h.4k668n3) Summary of Risk Assessments 25

[6.](#_heading=h.2zbgiuw) Create Policies for Encryption and Triple A 25

[7.](#_heading=h.1egqt2p) Map the Principles 26

[Audit Controls and Management](#_heading=h.3ygebqi) 27

[Enforcement](#_heading=h.2dlolyb) 27

[Exceptions Process](#_heading=h.sqyw64) 27

[Distribution](#_heading=h.3cqmetx) 28

[Policy Change Control](#_heading=h.1rvwp1q) 28

[Policy Version History](#_heading=h.4bvk7pj) 28

[Appendix A Lookups](#_heading=h.2r0uhxc) 28

[Approved C/C++ Language Acronyms](#_heading=h.1664s55) 28

## 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 | Any input a program or system receives should be validated. This means that the integrity and correctness of the data should be checked. This will ensure that only expected and safe inputs are accepted, preventing attacks such as injections. |
| 1. Heed Compiler Warnings | Compiler warnings can notify the developer of errors in their code. They can also provide insight into what is wrong with the section identified. This can help reduce the amount of vulnerabilities and bugs in code early on. |
| 1. Architect and Design for Security Policies | Architecting and designing software with security policies in mind can help ensure that robust security measures are implemented. |
| 1. Keep It Simple | Keeping code simple can improve security, as complex code can introduce vulnerabilities or errors that can be overlooked. Additionally, simple code can enhance its readability and maintainability. |
| 1. Default Deny | Access permissions should default to deny rather than only deny in certain situations. This can minimize the attack surface and ensures only those with explicit permission can access certain things. |
| 1. Adhere to the Principle of Least Privilege. | The Principle of Least Privilege says that users or processes should only have the minimum permissions to accomplish their tasks. By adhering to this, developers can ensure that users can not access more than they need to, and processes can be manipulated to access things that are not supposed to. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing the data sent to other systems by cleaning and validating the data helps protect other system components. |
| 1. Practice Defense in Depth | Defense in Depth is a security approach that utilizes multiple layers of security measures. This helps to protect the system because if one layer fails, then there are other layers that can stop the attack. |
| 1. Use Effective Quality Assurance Techniques | Effective quality assurance techniques can help ensure the program is secure by identifying potential vulnerabiulities to be fixed. Additionally, this helps eliminate bugs or other issues with the software. |
| 1. Adopt a Secure Coding Standard | Using a secure coding standard can give developers a guideline for coding best practices. Adhering to one can remove common security risks. |

### 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 | Never qualify a reference type with const or volatile.  C++ does not allow the value of reference types to be changed, thus treating all references as const-qualified. In turn, this results in C++ ignoring the cv-qualification of a reference type. Additionally, attempting to cv-qualify a reference type results in undefined behavior. |

| **Noncompliant Code** |
| --- |
| Here a const-qualified reference to a char is formed instead of a reference to a const-qualified char. This results in undefined behavior. |
| void ex(char c) {  char &const p = c;  p = ‘p’;  std::cout << c << std::endl;  } |
| Here p is correctly declared to be a reference to a const-qualified char. However, the subsequent modification of p makes the program ill-formed. |
| void ex(char c) {  const char &p = c;  p = ‘p’; // Error: read-only variable is not assignable  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| This solution is compliant as it allows the reference char p to be edited. |
| void ex(char c) {  char &p = c;  p = ‘p’;  std::cout << c << std::endl;  } |

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

| **Principles(s):** 4. Keep It Simple: Adding a const or volatile qualifier is unnecessary as C++ treats references as const-qualified already. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2023b | [CERT C++: DCL52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl52cpp.html) | Checks for   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| Clang | 3.9 |  | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options. |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-DCL52-a | Never qualify a reference type with ‘const’ or ‘volatile’ |
| SonarQube C/C++ Plugin | 4.10 | [S3708](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3708) |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Do not rely on the value of a moved-from object.  Many types support move semantics. However, in rare circumstances, an object of a type that supports move operations will be left in a valid but unspecified state after the object’s value has been moved. thus, do not rely on the value of a moved-from object unless the object type is documented to be in a well-specified state. This includes: unique\_ptr, shared\_ptr, weak\_ptr, basic\_ios, basic\_filebuf, thread, unique\_lock, shared\_lock, promise, future, shared\_future, and packaged\_task. |

| **Noncompliant Code** |
| --- |
| In this example, the integer values 0 through 9 are expected to be printed to the standard output stream from a std::string rvalue reference. However, because the object is moved and then reused under the assumption its internal state has been cleared, unexpected output may occur despite not triggering undefined behavior. |
| void print(std::string s) {  std::cout << s << std::endl;  }  void ex() {  std::string s;  for (unsigned i = 0; i < 10; ++i){  s.append(1, static\_cast<char>(‘0’ + i));  print(std::move(s));  }  } |

| **Compliant Code** |
| --- |
| Here, the std::string object is initialized to the expected value of each iteration of the loop. This ensures that the object is in a valid, specified state before attempting to access it in print(), resulting in the expected output. |
| void print(std::string s) {  std::cout << s << std::endl;  }  void ex() {  for (unsigned i = 0; i < 10; ++i){  std::string s(1, static\_cast<char>(‘0’ + i));  print(std::move(s));  }  } |

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

| **Principles(s):** 9. Use Effective Quality Assurance Techniques: This principle highlights the need for thorough testing. This is necessary to ensure we are not relying on the value of a moved-from object. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1 p0 | LANG.MEM.NPD  LANG.MEM.UVAR | Null Pointer Dereference  Uninitialized Variable |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-EXP63-a | Do not rely on the value of a moved-from objec |
| Polyspace Bug Finder | R2023b | [CERT C++: EXP63-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp63cpp.html) | Checks for read operations that read the value of a moved-from object (rule fully covered) |
| PVS-Studio | 7.29 | [V1030](https://pvs-studio.com/en/docs/warnings/v1030/) |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Do not attempt to create a std::string from a null pointer.  Creating a string from a null pointer can cause undefined behavior. |

| **Noncompliant Code** |
| --- |
| In this example, a std::string object is created from the results of a call to std::getenv(). However, std::getenv returns a null pointer on failure, which can cause undefined behavior when the environment variable does not exist. |
| void ex(){  std::string tmp(std::getenv(“TMP”));  if (!tmp.empty()){  // ...  }  } |

| **Compliant Code** |
| --- |
| Here, the results from the call to std::getenv are checked for null before the std::string object is constructed. Thus, avoiding the undefined behavior that can be caused in the noncompliant example. |
| void ex(){  const char \*tmpPtrVal = std::getenv(“TMP”);  std::string tmp(tmpPtrVal ? tmpPtrVal : “”);  if (!tmp.empty()) {  // ...  }  } |

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

| **Principles(s):** 1. Validate Input Data: This principle emphasizes the importance of validating input parameters to prevent undefined behavior, in this case, attempting to create a string from a null pointer. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Helix QAC | 2024.1 | DF4770, DF4771, DF4772, DF4773, DF4774 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2023b | [CERT C++: STR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr51cpp.html) | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Do not alternate input and output from a file stream without an intervening positioning call.  Alternating between input and output without a positioning call can lead to data being handled incorrectly, which can result in vulnerabilities, including SQL injections. |

| **Noncompliant Code** |
| --- |
| In this example, data is appended to the end of a file and then read from the same file. However, because there is no intervening positioning call between the formatted output and input calls, the behavior is undefined. |
| void ex(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }    file << “Output some data”;  std::string str;  file >> str;  } |

| **Compliant Code** |
| --- |
| In this solution, the seekg() function is called between the output and input, eliminating the undefined behavior. |
| void ex(const str::string &filename) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  file << “Output some data”;  std::string str;  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):** 3. Architect and Design for Security Policies: This principle emphasizes designing code with security in mind, particularly in handling file streams to prevent vulnerabilities like SQL injections. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | IO.IOWOP  IO.OIWOP | Input After Output Without Positioning  Output After Input Without Positioning |
| Helix QAC | 2024.1 | DF4711, DF4712, DF4713 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-FIO50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| Polyspace Bug Finder | R2023b | [CERT C++: FIO50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio50cpp.html) | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Detect and handle memory allocation errors.  The default memory allocation operator, ::operator new(std::size\_t), throws an exception if the allocation fails. Therefore, developers do not need to check if this call results in a nullptr. The nonthrowing form, ::operator new(std::size\_t, const std::nothrow\_t &), does not throw an exception if the allocation fails but instead returns nullprt. When using this form, it is very important to check that the return value is not nullptr before attempting to access the resulting pointer. |

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

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

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

| **Principles(s):** 9. Use Effective Quality Assurance Techniques: This principle emphasizes the need for thorough testing and handling of potential errors to ensure the reliability and robustness of memory allocation operations. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| Polyspace Bug Finder | R2023b | [CERT C++: MEM52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem52cpp.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |
| PVS-Studio | 7.29 | [V522](https://pvs-studio.com/en/docs/warnings/v522/), [V668](https://pvs-studio.com/en/docs/warnings/v668/) |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Do not treat a predefined identifier as an object if it might only be implemented as a macro. (Applies to C and C++)  In C++, macros can redefine identifiers, leading to unexpected behavior if those identifiers are treated as objects in the code. |

| **Noncompliant Code** |
| --- |
| In this example, the standard assert() macro is suppressed in an attempt to pass it as a function pointer to the execute\_handler() function. Attempting to suppress the assert() macro is undefined behavior. |
| typedef void(\*handler\_type)(int);  void execute\_handler(handler\_type handler, int value) {  handler(value);  }  void ex(int e) {  execute\_handler(&(assert), e < 0);  } |

| **Compliant Code** |
| --- |
| Here, the assert() macro is wrapped in a helper function, removing the undefined behavior. |
| typedef void (\*handler\_type)(int);  void execute\_handler(handler\_type handler, int value) {  handler(value);  }  static void assert\_handler(int value) {  assert(value);  }  void ex(int e){  execute\_handler(&assert\_handler, e < 0);  } |

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

| **Principles(s):** 4. Keep It Simple: This principle applies here because macros can redefine identifiers, which can lead to unexpected behavior. Thus, treating a predefined identifier as an object can cause unnecessary complexity. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | BADMACRO.STDARG\_H | Use of <stdarg.h> Feature |
| Parasoft C/C++test | 2023.1 | CERT\_C-MSC38-a | A function-like macro shall not be invoked without all of its arguments |
| Polyspace Bug Finder | R2023b | [CERT C: Rule MSC38-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemsc38c.html) | Checks for predefined macro used as an object (rule fully covered) |
| RuleChecker | 23.04 |  | Supported, but no explicit checker |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle all exceptions.  When an exception is thrown, control is transferred to the nearest handler with a type that matches the type of the exception thrown. If no matching handler is directly found within the handlers for a try block where the exception is thrown, the search for a matching handler continues to dynamically search for handlers in the surrounding try blocks of the same thread. If there is still no matching handler found, the function std::terminate() is called. The default terminate handler called by std::terminate() calls std::abort(), which abnormally terminates the process. When std::abort() is called, or if the implementation does not unwind the stack before calling std::terminate(), destructors for objects may not be called, and external resources can be left in an indeterminate state. Abnormal process termination is the typical vector for denial-of-service attacks. Thus, all exceptions thrown must be caught by a matching exception handler. At the very least, the stack should be unwound before terminating the process. |

| **Noncompliant Code** |
| --- |
| Here, neither ex() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. This could result in the stack not being unwound, which can lead to external resources being left in an indeterminate state. |
| void throwing\_func() noexcept(false);  void ex() {  throwing\_func();  }  int main() {  ex();  } |

| **Compliant Code** |
| --- |
| In this solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);  void ex() {  throwing\_func();  }  int main() {  try {  ex();  } catch(...) {  // Handle error  }  } |

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

| **Principles(s):** 8. Practice Defense in Depth: This principle emphasizes the layering of security measures. Thus, handling exceptions to prevent abnormal termination of the program is necessary.  9. Use Effective Quality Assurance Techniques: This principle emphasizes the need for thorough testing and handling of potential exceptions to ensure the program's reliability. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Helix QAC | 2024.1 | C++4035, C++4036, C++4037 |  |
| Parasoft C/C++test | 2023.1 | 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 | R2023b | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **noreturn Functions** | STD-008-CPP | Do not return from a function declared [[noreturn]].  The [[noreturn]] attribute specifies that a function does not return. A function that specifies [[noreturn]] can prohibit returning by throwing an exception, entering an infinite loop, or calling another function designated with the [[noreturn]] attribute. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, if the value 0 is passed, control will flow off the end of the function, resulting in an implicit return and undefined behavior. |
| [[noreturn]] void ex(int i) {  if (i > 0)  throw “Received positive input”;  else if (i < 0)  std::exit(0);  } |

| **Compliant Code** |
| --- |
| In this solution, the function does not return on any code path. |
| [[noreturn]] void ex(int i) {  if (i > 0)  throw “Received positive input”;  std::exit(0);  } |

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

| **Principles(s):** 9. Use Effective Quality Assurance Techniques: This principle emphasizes the need for thorough testing and handling of potential issues to ensure there are no errors in the program. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | invalid-noreturn | Fully checked |
| Polyspace Bug Finder | R2023b | [**CERT C++: MSC53-CPP**](https://www.mathworks.com/help/bugfinder/ref/certcmsc53cpp.html) | Checks for [[noreturn]] functions returning to caller (rule fully covered) |
| PVS-Studio | 7.29 | [**V1082**](https://pvs-studio.com/en/docs/warnings/v1082/) |  |
| SonarQube C/C++ Plugin | 4.10 | [**S935**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-935) |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Files** | STD-009-CPP | Close files when they are no longer needed.  A call to the std::basic\_filebuf<T>::open() function must be matched with a call to std::basic\_filebuf<T>::close() before the lifetime of the last pointer that stores the return value of the call has ended or before normal program termination, whichever occurs first. |

| **Noncompliant Code** |
| --- |
| Here, 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. |
| void ex(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  std::terminate();  } |

| **Compliant Code** |
| --- |
| In this solution, std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed. |
| void ex(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  file.close();  if (file.fail()) {  // Handle error  }  std::terminate();  } |

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

| **Principles(s):** 9. Use Effective Quality Assurance Techniques: This principle emphasizes the need for thorough testing and handling of potential issues to ensure the program is error-free and no files are left open. It also addresses the importance of resource management and preventing resource leaks, which can lead to security vulnerabilities and instability. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2024.1 | DF4786, DF4787, DF4788 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Polyspace Bug Finder | R2023b | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Containers** | STD-010-CPP | Guarantee that container indices and iterators are within the valid range |

| **Noncompliant Code** |
| --- |
| This example shows a function, insert\_in\_table(), that has two int parameters, pos and value, both of which can be influenced by data originating from untrusted sources. The function performs a range check to ensure that pos does not exceed the upper bound of the array, specified by tableSize, but fails to check the lower bound. Because pos is declared as a (signed) int, this parameter can assume a negative value, resulting in a write outside the bounds of the memory referenced by table. |
| void insert\_in\_table(int \*table, std::size\_t tableSize, int pos, int  value) {  if (pos >= tableSize) {  // Handle Error  return;  }  table[pos] = value;  } |

| **Compliant Code** |
| --- |
| In this solution, the parameter pos is declared as size\_t, which prevents the passing of negative arguments. |
| void insert\_in\_table(int \*table, std::size\_t tableSize, std::size\_t pos,  int value) {  if (pos >= tableSize) {  // Handle Error  return;  }  table[pos] = value;  } |

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

| **Principles(s):** 9. Use Effective Quality Assurance Techniques: This principle emphasizes the need for thorough testing and handling of potential errors caused by invalid indices or iterators. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2024.1 | C++3139, C++3140  DF2891 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-CTR50-a | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2023b | [CERT C++: CTR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcctr50cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |
| PVS-Studio | 7.29 | [V781](https://pvs-studio.com/en/docs/warnings/v781/) |  |

### 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 with 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.

Green Pace should work in automated enforcement of the standards into the ‘Verify and Test’ and ‘Monitor and Detect’ parts of the above DevSecOps process. This automation can be done through various code checkers, such as the one stated above, or through other testing methods, such as unit tests. Furthermore, Green Pace should modify its existing DevOps process to more closely match the above DevSecOps diagram.

That is, in pre-production they should start with planning and be sure to have security in mind, not just in this stage but all stages. Next, we move on to the system's design, again with security in mind. Then, we begin building and developing the system. In this stage, we also implement security features, which include following the coding standards discussed previously in this document and working on unit tests. The last stage in pre-production is ‘Verify and Test’. This stage is where we check for any vulnerabilities or bugs in the program and fix those.

From there, the DevSecOps process moves into the Production phase, starting with ‘Transition and health check.’ This stage configures the security settings and double-checks the program's security through a penetration test. The next stage is ‘Monitor and Detect,’ which is where we continuously monitor the system for issues and collect data. If an attack occurs, we move into the response stage. In this stage, we will attempt to block any attacks, turn off our services, and perform a rollback to restore any data that may have been affected. once the attack has been handled, we stabalize and maintain the system. Additionally, we move back into the pre-production stage to reassess and improve out security measures.

### 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 | Low | Unlikely | Low | Low | 3 |
| STD-002-CPP | Medium | Probable | Medium | Medium | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | Low | Likely | Medium | Medium | 2 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Unlikely | Medium | Low | 3 |
| STD-007-CPP | Low | Probable | Medium | Low | 3 |
| STD-008-CPP | Medium | Unlikely | Low | Low | 3 |
| STD-009-CPP | Medium | Unlikely | Medium | Low | 3 |
| STD-010-CPP | High | Likely | High | Medium | 2 |

### 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 | Encryption at rest applies to data that is not being used or transferred. This works to prevent unauthorized access to the data even if the device it is stored on is compromised. |
| Encryption in flight | Encryption in flight refers to data that is being transferred over a network. This ensures that data is safe even if the transfer is intercepted. |
| Encryption in use | Encryption in use refers to data that is being processed or used by applications or processes. This works to ensure that the data ios safe from data manipulation and unauthorized access. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication verifies that the user is who they say they are. It is typically used for user logins, database changes, and the addition of new users. |
| Authorization | Authorization checks the level of permissions a user has. It can be used to check if a user has permission to make changes to a database or access certain files. Additionally, depending on the system, only users of a certain level can register new users. |
| Accounting | Accounting is recording the various actions taken in relation to the system. This can be used to monitor who accesses what files and who makes what changes to which databases. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 03/24/2024 | Filled in the 10 security principles and part of the 10 coding standards sections. | Charlotte Armistead |  |
| 3.0 | 04/14/2024 | Filled in the rest of the template | Charlotte Armistead |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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