**Green Pace Developer: Security Policy Guide**



Green Pace Secure Development Policy

Contents

[Overview 2](#_30j0zll)

[Purpose 2](#_1fob9te)

[Scope 2](#_3znysh7)

[Module Three Milestone 2](#_2et92p0)

[Ten Core Security Principles 2](#_tyjcwt)

[C/C++ Ten Coding Standards 3](#_1t3h5sf)

[Coding Standard 1 4](#_4d34og8)

[Coding Standard 2 5](#_2s8eyo1)

[Coding Standard 3 6](#_17dp8vu)

[Coding Standard 4 7](#_3rdcrjn)

[Coding Standard 5 8](#_26in1rg)

[Coding Standard 6 9](#_lnxbz9)

[Coding Standard 7 10](#_35nkun2)

[Coding Standard 8 11](#_1ksv4uv)

[Coding Standard 9 13](#_44sinio)

[Coding Standard 10 14](#_2jxsxqh)

[Defense-in-Depth Illustration 15](#_z337ya)

[Project One 15](#_3j2qqm3)

[1.](#_1y810tw) Revise the C/C++ Standards 15

[2.](#_4i7ojhp) Risk Assessment 15

[3.](#_2xcytpi) Automated Detection 15

[4.](#_1ci93xb) Automation 15

[5.](#_3whwml4) Summary of Risk Assessments 16

[6.](#_2bn6wsx) Create Policies for Encryption and Triple A 16

[7.](#_qsh70q) Map the Principles 17

[Audit Controls and Management 18](#_3as4poj)

[Enforcement 18](#_1pxezwc)

[Exceptions Process 18](#_49x2ik5)

[Distribution 19](#_2p2csry)

[Policy Change Control 19](#_147n2zr)

[Policy Version History 19](#_3o7alnk)

[Appendix A Lookups 19](#_23ckvvd)

[Approved C/C++ Language Acronyms 19](#_ihv636)

# 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 | Validating all inputs from untrusted data sources is imperative. By doing so, we can mitigate numerous software vulnerabilities. Additionally, it's essential to remain vigilant for other warning indicators, such as environmental variables and command line arguments. |
| 1. Heed Compiler Warnings | Exercise caution during code compilation and address warnings by making necessary code changes. Employ both dynamic and static analysis tools to assist in identifying and eliminating any security issues and flaws. |
| 1. Architect and Design for Security Policies | Remain mindful of designing software in alignment with existing security policies. |
| 1. Keep It Simple | Opting for simplicity and minimalism provides a safer approach to protection compared to more complex systems. |
| 1. Default Deny | Grant access based on permission, not exclusion, to ensure only authorized users access sensitive information and resources, minimizing the risk of data breaches. |
| 1. Adhere to the Principle of Least Privilege | Regularly reviewing and removing unnecessary permissions will minimize the risk of unauthorized access. |
| 1. Sanitize Data Sent to Other Systems | Ensure that the data sent to more complex subsystems is sanitized. This preventive measure against attacks will mitigate the risk of further complications. |
| 1. Practice Defense in Depth | Employ multiple layers of security. This approach provides numerous fallbacks in case the initial line of defense is compromised. |
| 1. Use Effective Quality Assurance Techniques | Utilizing effective quality assurance techniques enhances our capacity to identify and eliminate any weaknesses in our defense. Security reviews and multiple testing phases can significantly enhance system integrity. |
| 1. Adopt a Secure Coding Standard | It's crucial to consistently maintain and adhere to secure coding standards across all languages and platforms in use. |

## 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] | Do not cast to an out-of range enumeration value |

|  |
| --- |
| **Noncompliant Code** |
| It checks whether a given value falls within the acceptable range of enumeration values. However, after casting the type, it may not be capable of representing the given integer value. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

|  |
| --- |
| **Compliant Code** |
| It is ensured that the value can be represented through the enumeration type before conversion, to verify that the conversion does not lead to unspecified values. This is achieved by constraining the converted values to a single enumerator value. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  if (intVar < First || intVar > Third) {  // Handle error  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

|  |
| --- |
| **Principles(s):** **validate input data**; this is because it is possible for unstructured values to result in buffer overflow, results to the execution of arbitrary code by an attacker. But, since the enumerators are rarely utilized for indexing different forms of pointer arithmetic’s, makes the scenario to result to information integrity violations than arbitrary coding execution. |

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Axivion Bauhaus  Suite | 7.2.0 | CertC++- INT50 |  |
| Helix QAC | 2021.2 | C++3013 |  |
| Parasoft C/C++test | 2021.1 | CERT\_CPP INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| PROA QA-C+ + | 4.4 | 3013 | [Insert text.] |
| PSV-Studio | 7.13 | V1016 |  |

### Coding Standard 2

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| **Data Value** | [STD-002-CPP] | Do not attempt to create a std::string from a null pointer |

|  |
| --- |
| **Noncompliant Code** |
| The std::string object is generated from the output of a call to std::getenv(). However, since std::getenv() returns a null pointer upon failure, this code leads to undefined behavior when encountering any other errors. |
| #include <cstdlib>  #include <string>  void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

|  |
| --- |
| **Compliant Code** |
| In the compliant solution, the results from the call to std::getenv() are assessed for null before constructing the std::string object. |
| #include <cstdlib>  #include <string>  void f() {  const char \*tmpPtrVal = std::getenv("TMP");  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  if (!tmp.empty()) {  // ...  }  } |

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

|  |
| --- |
| **Principles(s): Default Deny;** Dereferencing the null pointer present undefined behavior that results to abnormal program damages. In other instances dereferencing a null pointer may result to execution arbitrary code. The provided severity is evident in platforms where it is not possible to exploit the null pointer dereference to perform arbitrary code, thus the actual severity is low. |

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Astree | 20.10 | Assert \_ failure |  |
| Helix QAC | 2021.2 | C+ + 4770, C+ +4771,  C+ +4772, C+ +4773,  C+ +4774 |  |
| Klocwork | 2021.1 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF |  |
| Parasoft C/C+ +test | 2021.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

### Coding Standard 3

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| **String Correctness** | [STD-003- CPP] | Use a static assertion to test the value of a constant expression |

|  |
| --- |
| **Noncompliant Code** |
| Under noncompliant code uses the assert () macro to assert a property concerning a memory mapped structure that is essential for the code to behave c0orrectly: |
| #include <assert.h>  struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned  int) + sizeof(unsigned int));  } |

|  |
| --- |
| **Compliant Code** |
| This involve only constant expressions, it is acceptable to utilize a preprocessor conditional statement, as demonstrated in this compliant solution. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned  int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

|  |
| --- |
| **Principles(s):** **Keep It Simple**; The static assertion is an important diagnostic tool that help in finding and eliminating software defects that may contribute in vulnerabilities at compile time. The absence of static assertions, however, does not mean that code is incorrect. |

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Axivion Bauhaus Suite | 6.9.0 | CERTC-DCL03 |  |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 6.0p0 | (customization) | Users can implement a custom check that reports uses of assert () macro |
| Compass/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 |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

### Coding Standard 4

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| **SQL Injection** | [STD-004-CPP] | Do not use pointer-to-member operators to access nonexistent members ("Rules explorer", 2021). |

|  |
| --- |
| **Noncompliant Code** |
| A pointer-to-member object is obtained from D::g but is subsequently upcast to a B::\*. When invoked on an object whose dynamic type is D, the pointer-to-member is well-defined. However, since the dynamic type of the underlying object is B, this results in undefined behavior. |
| struct B {  virtual ~B() = default;  };    struct D : B {  virtual ~D() = default;  virtual void g() { /\* ... \*/ }  };    void f() {  B \*b = new B;    // ...    void (B::\*gptr)() = static\_cast<void(B::\*)()>(&D::g);  (b->\*gptr)();  delete b;  } |

|  |
| --- |
| **Compliant Code** |
| The upcast is eliminated, rendering the original code ill-formed and highlighting the current challenge that B::g() does not exist. This compliant solution confirms that the programmer's intention was to utilize the correct dynamic type for the underlying objects. |
| struct D : B {  virtual ~D() = default;  virtual void g() { /\* ... \*/ }  };    void f() {  B \*b = new D; // Corrected the dynamic object type.    // ...  void (D::\*gptr)() = &D::g; // Moved static\_cast to the next line.  (static\_cast<D \*>(b)->\*gptr)();  delete b;  } |

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

|  |
| --- |
| **Principles(s):** **Heed Complier Warnings**; in the compliant solution the code is effectively initialized to create a valid pointer-to member position than of to the default value of the null code. |

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Astree | 20.10 | overflow\_upon\_dereference  invalid\_function\_pointer |  |
| Axivion Bauhaus Suite | 7.2.0 | CertC+ +-OOP55 |  |
| Helix QAC | 2021.2 | C++2810, C++2811, C++2812, C++2813, C++2814 |  |
| Klocwork | 2021.1 | CERT.OOP.PTR\_MEMBER.NO\_MEMBER |  |
| Parasoft C/C+ + test | 2021.1 | CERT\_CPP-OOP55-a | A cast shall not convert a pointer to a function, different pointer type and pointer to function type. |
| Parasoft insure + + |  |  | Runtime detection |
| PRQA QA- C+ + | 4.4 | 2810, 2811, 2812, 2813, 2814 |  |

### Coding Standard 5

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| **Memory Protection** | [STD-005-CPP] | Honor replacement dynamic storage management requirements |

|  |
| --- |
| **Noncompliant Code** |
| Assuming that the global operator new(std::size\_t) function is substituted with custom usage. However, the custom implementation fails to adhere to the requirements specified by the C++ Standards for the function it replaces. |
| #include <new>    void \*operator new(std::size\_t size) {  extern void \*alloc\_mem(std::size\_t); // Implemented elsewhere; may return nullptr  return alloc\_mem(size);  }    void operator delete(void \*ptr) noexcept; // Defined elsewhere  void operator delete(void \*ptr, std::size\_t) noexcept; // Defined elsewhere |

|  |
| --- |
| **Compliant Code** |
| This addresses the required behavior for the replaced global allocator function by removing the std::bad\_alloc exception when allocation fails. |
| #include <new>    void \*operator new(std::size\_t size) {  extern void \*alloc\_mem(std::size\_t); // Implemented elsewhere; may return nullptr  if (void \*ret = alloc\_mem(size)) {  return ret;  }  throw std::bad\_alloc();  }    void operator delete(void \*ptr) noexcept; // Defined elsewhere  void operator delete(void \*ptr, std::size\_t) noexcept; // Defined elsewhere |

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

|  |
| --- |
| **Principles(s):** **Architect and Design for Security Policies**; failure to meet the outlined components for the replaceable dynamic storage functions may result to undefined characteristics. The impacts of the risks are subjected heavily on the caller of the allocation responsibility, however, some instances, dereferencing a null pointer can result to execution arbitrary code. |

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Helix QAC | 2021.2 | C++4736, C++4737, C++4738, C++4739 |  |
| Klocwork | 2021.1 | CERT.MEM.OVERRIDE.DELETE  CERT.MEM.OVERRIDE.NEW |  |
| Parasoft C/C+ +test | 2021.1 | CERT\_CPP-MEM55-a | The user determine the new operators must implement to ‘std:: bad\_alloc' exception when the different allocations fails. |
| Polyspace Bug Finder | R2021a | CERT C++: MEM55-CPP | Finds for the appropriate replacement allocations that fail to meet the recommended standards. |

### Coding Standard 6

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| **Assertions** | [STD-006-CPP] | Expressions used in assertions must not produce side effects. |

|  |
| --- |
| **Noncompliant Code** |
| The noncompliant code attempts to remove null names from the list within an assertion. However, when assertions are disabled, Boolean expressions are not evaluated, as per "2 Rules - SEI CERT C++ Coding Standard - Confluence" (2021). |
| private ArrayList<String> names;    void process(int index) {  assert names.remove(null); // Side effect  // ...  } |

|  |
| --- |
| **Compliant Code** |
| To mitigate potential side effects in assertions, it is recommended to separate the Boolean expressions from the assertion. |
| private ArrayList<String> names;    void process(int index) {  boolean nullsRemoved = names.remove(null);  assert nullsRemoved; // No side effect  // ...  } |

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

|  |
| --- |
| **Principles(s): Use Effective Quality Assurance Techniques;** The principle relates to the code since the side effects in the assertions contribute to the program behavior which depend on whether the assertions are enabled and disabled. |

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| CodeSonar | 6.1p0 | JAVA.STRUCT.SE.ASSERT | Assertions contains impacts (Java) |
| SonarQube | 6.7 | S3346 | Expressions used in "assert" should not produce side effects [Insert text.] |
| PSV-Studio | 7.13 | V6055 |  |

### Coding Standard 7

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| **Exceptions** | [STD-007-CCP] | Do not let exceptions escape from destructors or deallocation functions |

|  |
| --- |
| **Noncompliant Code** |
| The class destructor fails to fulfill the implicit noexcept guarantee, as it may throw an exception even when called in a context where exceptions are suppressed. Additionally, while it's asserted as false, it can trigger undefined behavior. |
| #include <stdexcept>    class S {  bool has\_error() const;    public:  ~S() noexcept(false) {  // Normal processing  if (has\_error()) {  throw std::logic\_error("Something bad");  }  }  }; |

|  |
| --- |
| **Compliant Code** |
| The compliant solution code should behave consistently whether exceptions are active or not. Hence, it supports operations that do not suppress exceptions and must handle all exceptions. |
| class SomeClass {  Bad bad\_member;  public:  ~SomeClass()  try {  // ...  } catch(...) {  // Catch exceptions thrown from noncompliant destructors of  // member objects or base class subobjects.    // NOTE: Flowing off the end of a destructor function-try-block causes  // the caught exception to be implicitly rethrown, but an explicit  // return statement will prevent that from happening.  return;  }  }; |

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

|  |
| --- |
| **Principles(s): Adopt a Secure Coding Standard**; the principle relates to the code since it supports the attempts to enhance exceptions from destructors and deallocation functions which may result to undefined characteristics resulting to denial of service attacks. |

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| Low | Likely | Medium | P6 | L2 |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Astree | 20.10 | destructor-without-noexcept  delete-without-noexcept | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL57 |  |
| Helix QAC | 2021.2 | C++2045, C++2047, C++4032, C++4631 |  |
| LDRA tool suite | 9.7.1 | 453 S | Partly implemented |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-DCL57-a  CERT\_CPP-DCL57-b | Do not allow exceptions to be eliminated from a destructor, swap and deallocation.  Always identify exceptions |
| Polyspace Bug Finder | R2021a | CERT C++: DCL57-CPP | Finds class destructors operating under exceptions |
| PVS-Studio | 7.13 | V509, V1045 |  |
| RuleChecker | 20.10 | destructor-without-noexcept  delete-without-noexcept | Checked completely |

### Coding Standard 8

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| Object oriented programming | [STD-008-CCP] | Do not delete a polymorphic object without a virtual destructor |

|  |
| --- |
| **Noncompliant Code** |
| The explicit pointer operation has been replaced with smart pointer objects, highlighting that smart pointers face a similar issue as others. This stems from the default delete behavior of std::unique\_ptr, which calls delete on the internal pointer position, resulting in behavior identical to other examples. |
| #include <memory>    struct Base {  virtual void f();  };    struct Derived : Base {};    void f() {  std::unique\_ptr<Base> b = std::make\_unique<Derived()>();  } |

|  |
| --- |
| **Compliant Code** |
| In the compliant solution code, a virtual destructor for the base class has been explicitly declared, ensuring that polymorphic delete operations result in well-defined behavior. |
| struct Base {  virtual ~Base() = default;  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

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

|  |
| --- |
| **Principles(s): practice defense in depth;** the principle relates to the coding standard since the attempts to destruct the polymorphic objects without virtual destructors presented results through undefined behavior. During operation the outcomes include memory leaks and abnormal program terminations. |

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Astree | 20.10 | non-virtual-public-destructor-in-non-final-class | Partially checked. |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP52 |  |
| Clang | 3.9 | -Wdelete-non-virtual-dtor |  |
| Helix QAC | 2021.2 | C++3402, C++3403, C++3404 |  |
| Klocwork | 2021.1 | CL.MLK.VIRTUAL  CWARN.DTOR.NONVIRT.DELETE |  |
| LDRA tool suite | 9.7.1 | 303 S | Partially utilized |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-OOP52-a | Find virtual destructors in classes utilized as base classes consisting virtual functions. |
| PRQA QA-C++ | 4.4 | 3402, 3403, 3404 |  |
| Polyspace Bug Finder | R2021a | CERT C++: OOP52-CPP | Evaluates the situations for virtual functions while avoiding virtual destructor. |
| PVS-Studio | 7.13 | V599, V689 |  |
| RuleChecker | 20.10 | non-virtual-public-destructor-in-non-final-class | Partially checked |
| SonarQube C/C++ Plugin | 4.10 | S1235 |  |

### Coding Standard 9

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| Input output | [STD-009-CPP] | Close files when they are no longer needed |

|  |
| --- |
| **Noncompliant Code** |
| In this noncompliant code example, a std::fstream object 'file' is constructed. The constructor for std::fstream calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. Consequently, the underlying std::basic\_filebuf<T> object maintained by the 'file' object is not properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  std::terminate();  } |

|  |
| --- |
| **Compliant Code** |
| In the compliant solution code, std::fstream::close() is invoked before std::terminate() is called, guaranteeing that the file resources are appropriately closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // ...  file.close();  if (file.fail()) {  // Handle error  }  std::terminate();  } |

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

|  |
| --- |
| **Principles(s):** Sanitize Data Sent to Other Systems; the principle relates to the coding conduct since inputting and outputting close files can allow attacker to eliminate the system and can promote the threats that the data written into memory files will not be flushed in operations of undefined programs. |

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| CodeSonar | 6.1p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2021.2 | C++4786, C++4787, C++4788 |  |
| Klocwork | 2021.1 | RH.LEAK |  |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-FIO51-a | Ensures resources are freed |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2021a | CERT C++: FIO51-CPP | Determine leak resources |

### Coding Standard 10

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| Concurrency | [STD-010-CPP] | Preserve thread safety and liveness when using condition variables |

|  |
| --- |
| **Noncompliant Code** |
| In this noncompliant example, five threads are utilized with the aim of executing sequentially, based on the assigned step level to each thread upon creation. The variable "currentStep" holds the current step level and is incremented when a specific thread completes its task. Finally, another thread is signaled to proceed with the next step. |
| #include <condition\_variable>  #include <iostream>  #include <mutex>  #include <thread>    std::mutex mutex;  std::condition\_variable cond;    void run\_step(size\_t myStep) {  static size\_t currentStep = 0;  std::unique\_lock<std::mutex> lk(mutex);    std::cout << "Thread " << myStep << " has the lock" << std::endl;    while (currentStep != myStep) {  std::cout << "Thread " << myStep << " is sleeping..." << std::endl;  cond.wait(lk);  std::cout << "Thread " << myStep << " woke up" << std::endl;  }    // Do processing...  std::cout << "Thread " << myStep << " is processing..." << std::endl;  currentStep++;    // Signal awaiting task.  cond.notify\_one();    std::cout << "Thread " << myStep << " is exiting..." << std::endl;  }    int main() {  constexpr size\_t numThreads = 5;  std::thread threads[numThreads];    // Create threads.  for (size\_t i = 0; i < numThreads; ++i) {  threads[i] = std::thread(run\_step, i);  }    // Wait for all threads to complete.  for (size\_t i = numThreads; i != 0; --i) {  threads[i - 1].join();  }  } |

|  |
| --- |
| **Compliant Code** |
| In this compliant solution, notify\_all() is used to signal all waiting threads instead of a single random thread. Only the code for the run\_step() thread from the noncompliant code example is modified. |
| #include <condition\_variable>  #include <iostream>  #include <mutex>  #include <thread>    std::mutex mutex;  std::condition\_variable cond;    void run\_step(size\_t myStep) {  static size\_t currentStep = 0;  std::unique\_lock<std::mutex> lk(mutex);    std::cout << "Thread " << myStep << " has the lock" << std::endl;    while (currentStep != myStep) {  std::cout << "Thread " << myStep << " is sleeping..." << std::endl;  cond.wait(lk);  std::cout << "Thread " << myStep << " woke up" << std::endl;  }    // Do processing ...  std::cout << "Thread " << myStep << " is processing..." << std::endl;  currentStep++;    // Signal ALL waiting tasks.  cond.notify\_all();    std::cout << "Thread " << myStep << " is exiting..." << std::endl;  }    // ... main() unchanged ... |

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

|  |
| --- |
| **Principles(s):** Adhere to the Principle of Least Privilege; the awakening of all threads ensures the liveness property since each thread can execute its condition, forecast test and one will succeed to continue with execution. |

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| Helix QAC | 2021.2 | C++1778, C++1779 |  |
| Klocwork | 2021.1 | CERT.CONC.UNSAFE\_COND\_VAR |  |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-CON55-a | Do not use the 'notify\_one()' function when various threads are waiting on the similar condition variable |
| PRQA QA-C++ | 4.4 | 5020 |  |

## Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



# Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

## Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

## Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

## Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

## Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

Automation refers to the development and implementation of technology to deliver goods and services under minimum human intervention. The of DevOps process as automation technologies, approaches and processes to improve reliability, efficiency and speed up different tasks performed by human intervention. The development process involves assessing ad planning, building, designing, building, verifying and testing including monitoring and detecting to enhance better protection data.

## Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| STD-001-CPP | Medium | Unlikely | Medium | P4 (low) | 3 |
| STD-002-CPP | High | Likely | Medium | P18 high | 1 |
| STD-003-CPP | Low | Unlikely | High | P1 low | 3 |
| STD-004-CPP | High | Probable | High | P6 high | 2 |
| STD-005-CPP | High | Likely | Medium | P18 high | 1 |
| STD-006-CPP | Low | Unlikely | Low | P3 low | 3 |
| STD-007-CPP | Low | Likely | Medium | P6 medium | 2 |
| STD-008-CPP | Low | Likely | Low | P9 high | 2 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 medium | 3 |
| STD-010-CPP | Low | Unlikely | Medium | P2 Low | 3 |

## Create Policies for Encryption and Triple A

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

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

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

|  |  |
| --- | --- |
| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| Encryption in rest | This encryption method is specifically crafted to thwart attackers' attempts to access unencrypted information by ensuring that data is encrypted while stored on disk. Encryption at rest is deployed to transform critical customer information into different data formats. This is achieved by employing an undisclosed algorithm, making it inaccessible to users without the appropriate encryption key needed for decoding. |
| Encryption at flight | This process entails encrypting data during its sharing, commonly employed in applications to protect data during exchange. It utilizes real-time encryption algorithms to encrypt data files as they are accessed. |
| Encryption in use | Implemented to bolster the protection of data stored on storage devices, computers, and information in transit networks, encryption in use stands as an effective strategy ensuring the long-term safeguarding of sensitive data. |

|  |  |
| --- | --- |
| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| Authentication | Procedure aimed at authenticating the user's identity, commonly implemented in computers and other data storage devices to thwart unauthorized access. This process utilizes security passwords or biometrics to prevent individuals from gaining access to the information. |
| Authorization | Enables users to access the computer for specific functions. Authorization is implemented to control access, following authentication. This policy is applied to ensure effective data protection. |
| Accounting | Involves computerized software programs stored within an organization's computer or network server. These systems play a pivotal role in managing income and expenses accounts, while also ensuring the confidentiality of the organization's accounting information through user login implementations. |

**\***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 standard and justify 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

|  |  |  |
| --- | --- | --- |
| Coding Standard | Principles | Relationship |
| STD-001-CPP | validate input data | Coding standards are crucially linked to this principle due to the potential for unmanaged values to trigger buffer overflow, thereby enabling attackers to execute arbitrary code. However, because enumerators are seldom employed for indexing various pointer arithmetic operations, the likelihood of information integrity breaches outweighs that of arbitrary code execution in this scenario. |
| STD-002-CPP | Default Deny | The act of dereferencing a null pointer can lead to the execution of arbitrary code. The severity of this issue becomes apparent in platforms where exploiting null pointer dereference to execute arbitrary code is not feasible, thereby indicating a low actual severity. |
| STD-003-CPP | Simplifying | Static assertion serves as a crucial diagnostic tool, aiding in the detection and elimination of software defects that could potentially lead to vulnerabilities during compile time. Nevertheless, the lack of static assertions does not necessarily imply incorrect code. |
| STD-004-CPP | Heed Complier Warnings | In the compliant solution, the code is initialized in a manner that ensures a valid pointer-to-member position, rather than relying on the default null value |
| STD-005-CPP | Architect and Design for Security Policies | The implications of these risks fall on the caller responsible for allocation. However, in certain cases, dereferencing a null pointer can lead to the execution of arbitrary code. |
| STD-006-CPP | Use Effective Quality Assurance Techniques | The principle pertains to the code due to the influence of side effects in assertions on the program's behavior, contingent upon whether these assertions are enabled or disabled. |
| STD-007-CPP | Adopt a Secure Coding Standard | The principle is important for the code because it helps to improve exceptions from destructors and deallocation functions, which might otherwise cause undefined behaviors and possibly lead to denial-of-service attacks. |
| STD-008-CPP | Practice defense in depth | The principle is relevant to coding standards because attempting to destruct polymorphic objects without virtual destructors can lead to undefined behavior. This can result in outcomes such as memory leaks and abnormal program terminations during operation. |
| STD-009-CPP | Sanitize Data Sent to Other Systems | The principle is connected to coding conduct because failing to properly close input and output files can enable attackers to compromise the system. Additionally, it can introduce threats such as data written into memory files not being flushed during operations of undefined programs. |
| STD-010-CPP | Adhere to the Principle of Least Privilege | Ensuring that all threads are activated guarantees the liveness property, as each thread can evaluate its conditions and forecast tests. Eventually, one will successfully proceed with the execution. |

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 |

## References

*2 Rules - SEI CERT C++ Coding Standard - Confluence*. Wiki.sei.cmu.edu. (2021). Retrieved 11 August 2021, from https://wiki.sei.cmu.edu/confluence/display/cplusplus/2+Rules.

*Rules explorer*. Rules.sonarsource.com. (2021). Retrieved 11 August 2021, from https://rules.sonarsource.com/cpp.