**Green Pace Developer: Security Policy Guide**



Green Pace Secure Development Policy

Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

# Overview

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

# Purpose

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

# Scope

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

# Module Three Milestone

## Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Data should always be validated, especially from untrusted data sources such as command line arguments, network interfaces and environmental variables. By performing proper input validation, you can limit a vast majority of vulnerabilities such as SQL injection. |
| 1. Heed Compiler Warnings | Use static and dynamic analysis tools to detect flaws in the code. When the code is compiled, it should be done with the highest warning level available. All warnings should be addressed by modifying the code to eliminate the warning. |
| 1. Architect and Design for Security Policies | When designing software, it is important to follow best practices and secure coding standards. One approach is to design software with role-based privileges this way you can limit who can access the system as well as limiting access to certain parts of the system once the user has been authenticated. |
| 1. Keep It Simple | It is better to keep the design and implementation small and simple. Complex systems are more susceptible to errors and failed security mechanisms. |
| 1. Default Deny | Systems should be designed to deny access by default and access should be based on permission rather than exclusion. This ensures that only explicit access is given and helps avoiding giving unintended access. |
| 1. Adhere to the Principle of Least Privilege | Access should be restricted as possible. Only necessary access should be given and if escalated is needed it should be limited to least amount of time possible. This will help reduce the opportunity for bad actors to execute malicious code. |
| 1. Sanitize Data Sent to Other Systems | All data should be sanitized as the system permits especially data passed through subsystems. This will help mitigate attacks such as SQL injection and other injection attacks. |
| 1. Practice Defense in Depth | A layered security approach as the system permits should be implemented. This is so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from being exploited. A layered approach increased the effort needed to exploit the software and may dissuades attackers. |
| 1. Use Effective Quality Assurance Techniques | Use quality assurance techniques to help identify and eliminate vulnerabilities. This can be in form of penetration testing, audits, and external reviews. An external perspective can help uncover security flaws that would have gone unnoticed otherwise. |
| 1. Adopt a Secure Coding Standard | Always follow best practice and security standards for development of the chosen system language. |

## 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** |
| --- |
| In the example below the convertToInteger method checks whether a given value is within range of acceptable enumeration values. However, after casting the type it will not be able to represent the given integer value. |
| enum EnumType {  First,  Second,  Third };  void convertToInteger(int *var*) {  EnumType enumVar = static\_cast<EnumType>(*var*);    if(enumVar < First || enumVar > Third) {  *// Throw Error* } } |

| **Compliant Code** |
| --- |
| The compliant solution checks the value represent by the enumeration type before performing the conversion to guarantee the conversion doesn’t result in an unspecified value. In turn it restricts the converted value to once specific enumerator type. |
| enum EnumType {  First,  Second,  Third };  void convertToInteger(int *var*) {  if(*var* < First || *var* > Third) {  *// Throw Error* }  EnumType enumVar = static\_cast<EnumType>(*var*); } |

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

|  |
| --- |
| **Principles(s):** In some cases, the use of unspecified values can result in a buffer overflow, which can lead to the execution of arbitrary code by an attacker. However, because enumerators aren’t often used for indexing arrays or other forms of pointer arithmetic, it is more likely that this scenario will result in data integrity violations rather than arbitrary code execution. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bahaus Suite | 6.9.0 | CertC++ - INT50 | [Insert text.] |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP – INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of enumeration |
| PRQA QA- C++ | 4.4 | 3013 |  |
| PVS – Studio | 7.07 | V1016 |  |

### Coding Standard 2

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

| **Noncompliant Code** |
| --- |
| In this example, the position variable is invalidated after the first call to the insert.function, and a subsequent loop iterations will result in undefined behavior. |
| #include <deque>  void addToQueue(const double \* *items*, std::*size\_t count*) {  std::deque<double> myQueue;  auto position = myQueue.begin();    for (std::*size\_t* i = 0; i < *count*; ++i, **++**position) {  myQueue.insert(position, *items*[i] + 50.10)  } } |

| **Compliant Code** |
| --- |
| To remain compliant, we restructure the previous example so that the position variable is assigned a valid iterator each time insert is called which will prevent undefined behavior. |
| #include <deque>  void addToQueue(const double \* *items*, std::*size\_t count*) {  std::deque<double> myQueue;  auto position = myQueue.begin();   for (std::*size\_t* i = 0; i < *count*; ++i, **++**position) {  position = myQueue.insert(position, *items*[i] + 50.10);  } } |

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

|  |
| --- |
| **Principles(s):** Using invalid references, pointers or iterators to reference elements of container results in undefined behavior. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Overflow\_unpon\_dereference |  |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-CTR51-a | Do not modify container while iterating over it |
| PVS – Studio | 7.07 | V783 |  |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Do not attempt to create a string from a null pointer |

| **Noncompliant Code** |
| --- |
| In the following example, a string object is created when getenv() is called. However, because getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist. |
| #include <cstlib> #include <string>  void stringCorrectness() {  std::*string* tmp(std::getenv("temp"));  if(!tmp.empty()) {  *// Do something* } } |

| **Compliant Code** |
| --- |
| A better solution the problem above would be to check the result of the return value from the getenv() before the string object is constructed. |
| #include <cstlib> #include <string>  void stringCorrectness() {  const char \*tmpPtrVal = std::getenv("temp");  std::*string* tmp(tmpPtrVal ? tmpPtrVal : "");  if(!tmp.empty()) {  *// Do something* } } |

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

|  |
| --- |
| **Principles(s):** Dereferencing a null pointer is undefined behavior and can cause a program to terminate. Attackers could also leverage this exploit and use it execute arbitrary code. |

**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 | 2021.1 |  |  |
| ParasoftC/C++ | 2020.2 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Do not store already owned pointer value in an unrelated smart pointer |

| **Noncompliant Code** |
| --- |
| In this example, two unrelated smart pointers are constructed from the same underlying pointer value. However, when local p2 variable is destroyed, it deletes the pointer value and when the local p2 variable is destroyed it deletes the same pointer value resulting in a double-free vulnerability. |
| #include <memory>  void func() {  int \*i = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i); } |

| **Compliant Code** |
| --- |
| A better solution to the previous example would put forth the use of copy construction, this way when the p2 variable is destroyed, the use count for shared pointer value is decremented but still nonzero. When the p1 variable is destroyed the use count for shared pointer value is also decremented and the managed pointer is then destroyed. |
| #include <memory>  void func() {  std::shared\_ptr<int> p1 = std::make\_shared<int>();  std::shared\_ptr<int> p2(p1); } |

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

|  |
| --- |
| **Principles(s):** Passing a pointer value to a deallocation function that was obtained by the matching allocation function will result in undefined behavior. This can lead to attackers exploiting this vulnerability with methods such SQL injection. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Dangling\_pointer\_use |  |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-MEM56-a | Do not store an already-owned pointer value in an unrelated smart pointer |
| PVS – Studio | 7.01 | V1006 |  |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| In this example, the local variable space is passed as the expression to the placement new operator. The resulting pointer of that call is then passed to delete call, which will result in undefined behavior. This is because when delete is called the system is trying to free memory that was not returned when new is called. |
| #include <iostream>  struct y {  y() {std::cout **<<** "y::y()" **<<** std::endl;}  ~y() {std::cout **<<** "y::~y" **<<** std::endl;} };  void x() {  alignas(struct y) char space[sizeof(struct y)];  y \*y1 = new (&space) y;   *// Do Something* delete y1; } |

| **Compliant Code** |
| --- |
| To fix the previous example we can remove the delete call and instead explicitly call the y1 constructor. |
| #include <iostream>  struct y {  y() {std::cout **<<** "y::y()" **<<** std::endl;}  ~y() {std::cout **<<** "y::~y" **<<** std::endl;} };  void x() {  alignas(struct y) char space[sizeof(struct y)];  y \*y1 = new (&space) y;   *// Do Something* y1->~ y(); } |

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

|  |
| --- |
| **Principles(s):** Passing a pointer value to a deallocation function that was not previously obtained by the matching allocation function results in undefined behavior, which can lead to exploitable vulnerabilities. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Invalid\_dynamic\_memory\_alocation\_dangling\_pointer\_use |  |
| Axivion Bauhaus Suite | 6.9.0 | CERTC++MEM51 |  |
| Clang | 3.9 | Clang-analyzer-cplusplus.NewDeleteLeaks-Wmismatched-new-delete clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| CodeSonar | 6.0p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM | Free non-heap variable  Double free  Type mismatch |
| Helix QAC | 2021.1 |  |  |
| Klocwork | 2021.1 | CL.FFM.ASSIGNFM  CL.FFM.COPY  CL.FMM  FMM.MIGHT  FMM.MUST  FNH.MIGHT  FNH.MUST  FUM.GEN.MIGHT  FUM.GEN.MUST  UNINIT.CTOR.MIGHT  UNINIT.CTOR.MUST  UNINIT.HEAP.MIGHT  UNINIT.STACK.ARRAY.MIGHT  UNINIT.STACK.ARRAY.MUST |  |
| LDRA tool suite | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 438 S, 484 S, 485 S, 64 D, 112 D | Partially implemented |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d | Use the same form in corresponding calls to new/malloc and delete/free  Always provide empty brackets ([]) for delete when deallocation arrays  Both copy constructor and copy assignment operator should be declared for classes with a nontrivial destructor |
| PVS – Studio | 7.07 | V515, V554, V611, V701, V748, V773 |  |
| SonarQube C/C++ | 4.10 | S1232 |  |

### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| In this example a macro is used assert the property of a memory wrapped structure. |
| #include <assert.h>  struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT; };  int checkTime(void) {  assert(sizeof (struct timer) == sizeof(unsigned char)  + sizeof(unsigned int)  + sizeof(unsigned int)  ); } |

| **Compliant Code** |
| --- |
| A better solution to the previous example would be the use of preprocessor conditional statements as seen below. |
| #include <assert.h>  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 padding" #endif |

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

|  |
| --- |
| **Principles(s):** Static assertion is valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities at compile time. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion | 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 the assert() macro. |
| Compass/Rose |  |  | Could detect violations of this rule merely by looking for calls to assert() and if can evaluate the assertion. |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle exceptions thrown before main is executed |

| **Noncompliant Code** |
| --- |
| In this example, the constructor will throw an exception that is not caught when globalS is constructed. |
| struct S {  S() noexcept(false); };  static S globalS; |

| **Compliant Code** |
| --- |
| The proper solution would be to make globalS into a local variable with static storage duration, allowing any thrown exceptions to be caught. This is because the function will be constructed when globalS() is called instead being constructed at startup like the previous example. |
| struct S {  S() noexcept(false); };  S &globalS() {  try{  static S s;  return s;  } catch (e) {  *// Error Handling* } } |

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

|  |
| --- |
| **Principles(s):** Unhandled exceptions can lead to abnormal behavior and can lead to vulnerabilities such as denial-of-service attacks. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Potentially-throwing-static-initialization | Partially Checked |
| Axivion Bauhaus Suite | 6.9.0 | CERTC++ERR58 |  |
| Clang | 3.9 | Cert-eer58-cpp | Checked by clang-tidy |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ test | 2020.2 CERT\_CPP-ERR58a |  | Exceptions shall be raised only after start-up and before termination of the program. |
| PRQA QA-C++ | 4.4 | 4634, 4636, 4637, 4639 |  |
| Rule Checker | 20.10 | Potentially-throwing-static-initialization | Partially checked |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programing | STD-008-CPP | Do not invoke virtual functions from constructors or destructors |

| **Noncompliant Code** |
| --- |
| In this example, the base class attempts to seize and release an objects resource through calls to virtual functions from the constructor for the constructor and destructor. However, not only is it not recommended to invoke virtual members from constructor but the V() constructor calls the seize and release from V() instead calling it from D(). |
| struct V {  V() {seize();}  virtual ~V() {release();}  protected:  virtual void seize();  virtual void release(); };  struct D : V {  virtual ~D() = default;  protected:  void seize() override {  V::seize();  *// Seize derived resources...* }   void release() override {  *// Release derived resources...* V::release();  } }; |

| **Compliant Code** |
| --- |
| A better solution to the previous example would be that the constructors and destructors to call a non-virtual private member function instead of a virtual function. In the example below each class is responsible for their own resources. |
| class V{  void seize\_mine();  void release\_mine();  public:  V() {seize\_mine();}  virtual ~V() {release\_mine();}  protected:  virtual void seize() {seize\_mine();}  virtual void release() {release\_mine();} };   Class D : public V {  void seize\_mine();  void release\_mine();  public:  D() {seize\_mine();}  virtual ~D() {release\_mine();}  protected:  void seize() override {  V::seize();  seize\_mine();  }   void release() override {  release\_mine();  V::release();  } }; |

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

|  |
| --- |
| **Principles(s):** Order of construction starts with base classes and moves to more derived classes, attempting to call a derived class function from a base class under construction is dangerous because the derived class has not properly initialized all its resources when called. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Virtual-call-in constructor invalid\_function\_pointer | Fully checked |
| Axivion Bahause Suite | 6.9.0 | CertC++OOP50 | Checked by clang-tidy |
| Helix QAC | 2021.1 |  |  |
| LDRA tool suite | 9.7.1 | 467S, 92D | Fully implemented |
| Parasoft C/C++ test | 2020.2 | CERT CPP-OOp50-a  CERT CPP-OOp50-b  CERT CPP-OOp50-c  CERT CPP-OOp50-d | Avoid calling virtual functions from constructors  Avoid calling virtual functions from destructors |
| PRQA QA-C++ | 4.4 | 4260, 4261, 4273, 4274, 4275, 4276, 4277, 4278, 4279, 4280, 4281, 4282 |  |
| PVS-Studio | 20.10 | Virtual-call-in-customer | Fully checked |
| SonarQube C/C++ Plugin | 4.10 | S1699 |  |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input outoput | STD-009-CPP | Do not alternate input and output from a file stream without an intervening positioning call |

| **Noncompliant Code** |
| --- |
| In this example, a file stream is used to read a file and append data to the end of said file. However, there is no intervening positioning call resulting in undefined behavior. |
| #include <fstream> #include <string>  void func(const std::*string* &*filePath*) {  std::*fstream* file(*filePath*);  if (!file.is\_open()) {  *// Error handling* return;  }    file **<<** "Output some data";  std::*string* str;  file **>>** str; } |

| **Compliant Code** |
| --- |
| To resolve the previous example the seekg() function is called between the output and input which will eliminate the undefined behavior from the previous example. |
| #include <fstream> #include <string>  void func(const std::*string* &*filePath*) {  std::*fstream* file(*filePath*);  if (!file.is\_open()) {  *// Error handling* 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):** Alternating input/output from a stream without an intervening flush or positioning call will result in undefined behavior. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ | 2020.2 | CERT\_CPP\_FIO50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| Polyspace Bug Finder | R2020a | ERCT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous | STD-010-CPP | Value returning functions must return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| In the example below, the method only if it meets the criteria of the if statement resulting in undefined bahavior. |
| int absolute\_value(int *a*) {  if (*a* < 0) {  return -*a*;  } } |

| **Compliant Code** |
| --- |
| To resolve the previous example, we add a return statement in both the if statement and outside the if statement. |
| int absolute\_value(int *a*) {  if (*a* < 0) {  return -*a*;  }  return *a*; } |

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

|  |
| --- |
| **Principles(s):** Non void methods that fail to return a value will result in undefiled behavior which can be exploited and cause data integrity violations. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Return-implicit | Fully checked |
| Axivion | 6.9.0 | CertC++-MSC52 |  |
| Bauhaus Suite |  |  |  |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this rule, such as function-try-blocks |
| CodeSonar | 6.0p0 | LANG.STRUCT.MRS | Missing return statement |
| Helix QAC | 2021.1 |  |  |
| LDRA tool suite | 9.7.1 | 2 D, 36 S | Fully implemented |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-MSC52-a | All exit paths from a function with non-void return type shall have an explicit return statement with an expression |
| Polyspace Bug Finder | R2020a | Cert C++: MSC52-a | Checks for missing return statements |
| SonarQube C/C++ Plugin | 4.10 | S935 |  |
| PRQA QA - C++ | 4.4 | 1510 |  |
| PVS-Studio | 7.07 | V591 |  |
| RuleChecker | 20.10 | Return-implicit | Fully Checked |

## 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.

Security is often overlooked and sometimes treated as impediment in projects. Here at Green Pace, we do we do the opposite, security is what enables us to develop and deploy secure applications with confidence. However, achieving this is not trivial but through the use of tools we are able to leverage automation to make such tasks repeatable, scalable and reliable.

Here at Green Pace, we leverage a multitude of tools with each one serving a specific purpose. For writing code, we use linters such as PMD, this tool helps detects vulnerabilities as the code is being written and it can also be configured to detect deviations from the best practices that the company has established. For the build process we use tools such as SonarQube, this tool will scan the code and scan dependencies in our application. For the release we use tools such as terraform and ansible, this allows us to convert our configuration to reusable templates. This comes in handy because it allows us to create a reusable base layer that follows Green Pace best practices and helps cut down on human error. Lasty, for motoring we use tools such as firewalls, and log collectors. While firewalls help us detect intrusions and other suspicious activity, logs help us understand when the intrusion happened, how it happened and what was accessed. All in all, like seen in the diagram above security is an irritative and continuous process.

## Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | P4 | L3 |
| STD-002-CPP | High | Probable | High | P6 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-009-CPP | Low | Likely | Medium | P6 | L2 |
| STD-0010-CPP | Medium | Probable | Medium | P8 | L2 |

## Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Keeping customers data safe is of utmost importance. There are a couple ways to accomplish this, such data encryption at rest. Encryption at rest is designed to prevent the attacker from accessing the unencrypted data by ensuring the data is encrypted when on disk. If an attacker obtains a hard drive with encrypted data but not the encryption keys, the attacker must defeat the encryption to read the data. |
| Encryption at flight | In addition to the encryption at rest, data often needs to be accessed over the network. When the data sits on the disk the data remains encrypted, however, once it is in transmission it becomes vulnerable. To mitigate this vulnerability the data can also be encrypted in transit which protects your data if communications are intercepted while data moves between your site and the cloud provider or between two services. This protection is achieved by encrypting the data before transmission; authenticating the endpoints; and decrypting and verifying the data on arrival. |
| Encryption in use | As mentioned, protecting customer data is of utmost importance. While encrypting data at rest and in transit does help it does not fully protect the data. This is because to interact with the data it first must decrypted which again leaves data vulnerable. However, through the use of Homomorphic Encryption that data can be used without having being decrypted. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process where the user’s identity is confirmed. It essentially confirms the question, are you who you say you are? This is done through methods such login and password authentication and multi factor authentication. |
| Authorization | Authorization refers to the level of access a user has once the user has been authenticated. When speaking in terms of database, authorization would determine if the user has read, edit, create, and delete access. Once recommended approach to authorization is to implement role-based authorization. This allows us to grant access via a role instead of individual assignments which are often hard to maintain. |
| Accounting | Accounting refers to the audit of the system. System security should be reviewed and monitored at a regular cadence. Doing so help will help uncover suspicious and malicious activity. |

**\***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 | 08/08/2021 | Final Template | Jonathan Haro |  |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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