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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_gjdgxs)

[Purpose 2](#_30j0zll)

[Scope 2](#_1fob9te)

[Module Three Milestone 2](#_3znysh7)

[Ten Core Security Principles 2](#_2et92p0)

[C/C++ Ten Coding Standards 3](#_3dy6vkm)

[Coding Standard 1 4](#_1t3h5sf)

[Coding Standard 2 5](#_4d34og8)

[Coding Standard 3 6](#_2s8eyo1)

[Coding Standard 4 7](#_17dp8vu)

[Coding Standard 5 8](#_3rdcrjn)

[Coding Standard 6 9](#_26in1rg)

[Coding Standard 7 10](#_lnxbz9)

[Coding Standard 8 11](#_35nkun2)

[Coding Standard 9 13](#_1ksv4uv)

[Coding Standard 10 14](#_44sinio)

[Defense-in-Depth Illustration 15](#_2jxsxqh)

[Project One 15](#_z337ya)

[1.](#_3j2qqm3) Revise the C/C++ Standards 15

[2.](#_1y810tw) Risk Assessment 15

[3.](#_4i7ojhp) Automated Detection 15

[4.](#_2xcytpi) Automation 15

[5.](#_1ci93xb) Summary of Risk Assessments 16

[6.](#_3whwml4) Create Policies for Encryption and Triple A 16

[7.](#_2bn6wsx) Map the Principles 17

[Audit Controls and Management 18](#_qsh70q)

[Enforcement 18](#_3as4poj)

[Exceptions Process 18](#_1pxezwc)

[Distribution 19](#_49x2ik5)

[Policy Change Control 19](#_2p2csry)

[Policy Version History 19](#_147n2zr)

[Appendix A Lookups 19](#_3o7alnk)

[Approved C/C++ Language Acronyms 19](#_23ckvvd)

## 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 | This principle of security is checking and filtering the data being entered into the system by the user. This ensures that the data being entered meets the needed criteria before the data gets processed. An example of this is usernames and passwords to enter into certain areas, this helps prevent unwanted people from accessing data they aren't meant to see. |
| 1. Heed Compiler Warnings | This principle of security is compiling code using the highest warning level you're compiler will let you use and then editing the code so that those warnings disappear. This is typically paired with static and dynamic tools that allow the developer to locate additional security flaws not found with the high level of warning. |
| 1. Architect and Design for Security Policies | This is the principle of creating a process of implementing security controls and safe guards within the IT systems to help protect the data. These policies are put in place to help protect the users from outside attacks and keep their data safe. |
| 1. Keep It Simple | This guideline is in place to remind the developers of the importance of keeping their code simple. A good system is able to run and complete everything it needs to without being filled with redundant and/or unnecessary code. This makes it easier to find mistakes done and make it easier when transitioning in new team members so they can recognize the intent of the code. |
| 1. Default Deny | This principle is that access should be immediately denied unless the user uses the specific conditions needed to gain access. This helps to reduce security breaches through malware and other dangerous viruses. |
| 1. Adhere to the Principle of Least Privilege | This principle is that users should be given the minimum amount of resources and applications needed in order to complete their tasks. This helps prevent them stumbling along system areas or accessing information that is above their associate level. |
| 1. Sanitize Data Sent to Other Systems | This principle is that data should be fully destroyed and wiped clean from the storage device to ensure it can not be recovered from any device by any means. This is to help protect sensitive data that the company doesn't want to fall into the wrong hands. |
| 1. Practice Defense in Depth | This principle is to remind developers to utilize multiple layers of security within their system so that is one of their organizations level of security gets breached there will be fall backs to help stop the attacker from getting to the data. This is important for organizations that carry sensitive data so that it can't be easily hacked and leaked. |
| 1. Use Effective Quality Assurance Techniques | This principle states that developers should use effective quality techniques when checking their work such as penetration testing in order to locate vulnerabilities within the code. This is important for developers to remember to do because if they are using methods that aren't helpful at finding as many vulnerabilities as possible then it is a waste of time and resources and can still lead to security risks. |
| 1. Adopt a Secure Coding Standard | This is important for all developers as it will help them have standards in place to help prevent security vulnerabilities within their code and keep their company and its users as safe as possible from outside attacks. |

### 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 | Don't cast out to an out of range enumeration value |

| **Noncompliant Code** |
| --- |
| This will check if a given value is in the allotted range of allowed enumeration values. If the type is casted, it potentially cant represent the assigned 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** |
| --- |
| This code checks the represented value by its enumeration type prior to completing the conversion so that the conversion wont create an unspecified value. This helps to restrict the type to a specific enumerator type. |
| 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):** There is a possibility that if given unspecified values those could cause a buffer overflow which would allow the attacker to utilize arbitrary code. Since enumerators are rarely ever used however when it comes to indexing arrays, there is a higher chance that this scenario would result in a data integrity violation rather then an execution of arbitrary code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2020.2 | CERT+CPP-INT50-a | An expression that contains an enum underlying type will only contain values that correspond with the enumerators of enumeration. |
| PVS-Studio | 7.07 | V10106 |  |
| Helix QAC | 2021.1 |  |  |
| Axivion Bauhaus Suite | 6.9.0 | CertC++ - INT50 |  |
| PRQA QA- C++ | 4.4 | 3013 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Utilize valid pointers, integrators, and references when referencing a containers element |

| **Noncompliant Code** |
| --- |
| This code has a POS which is invalid after calling for the first time and if looped again it caused undefined behavior. |
| #include <deque>  void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  d.insert(pos, items[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| This code ensures the POS is given a valid iterator so that undefined behavior doesn't happen. |
| #include <deque>  void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  pos = d.insert(pos, items[i] + 41.0);  }  } |

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

| **Principles(s):** Utilizing invalid references, iterators, or pointers as container reference elements will result in undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

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

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not try to make a std::string from a null location |

| **Noncompliant Code** |
| --- |
| This code will have the std::string object is made from calling to std:getenc() and since the std::getenc() will return a null pointer if it fails the code will create an undefined behavior. |
| #include <cstdlib>  #include <string>  void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| This code will have the results be checked 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):** This coding standard has a varying degree of severity depending on the situation. When you dereference a null pointer, it makes a undefined behavior it results in a abnormal program termination. In some cases, though it can lead to an arbitrary code execution. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.1 |  |  |
| Astree | 20.10 | Assert\_failure |  |
| ParasoftC/C++ test | 2020.2 | CERT\_CPP-STR51-a | Avoid null pointer deferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Do not take an assigned pointer value and put it in an unrelated pointer, |

| **Noncompliant Code** |
| --- |
| This code has two unrelated pointers be created from the same pointer value. This can lead to a double free vulnerability within the code. |
| #include <memory>  void f() {  int \*i = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i);  } |

| **Compliant Code** |
| --- |
| This code has the objects be related through a copy construction. This will make it so if the points get destroyed the code will call std:make\_shared() instead of going into a local variable. |
| #include <memory>  void f() {  std::shared\_ptr<int> p1 = std::make\_shared<int>();  std::shared\_ptr<int> p2(p1);  } |

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

| **Principles(s):** If you take a pointer value and pass it to a deallocation function that was not originally received by a matching allocation function you will create an undefined behavior which can be exploitable. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS – Studio | 7.01 | V1006 |  |
| Astree | 20.10 | Dangling-pointer\_use |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-MEM56-a | Don’t put a claimed pointer value into a unrelated smart pointer |
| Helix QAC | 2021.1 |  |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Deallocate dynamically allocated resources properly. |

| **Noncompliant Code** |
| --- |
| This code has the local variable space be passed as an expression for a new operator. This will result in undefined behavior. |
| #include <iostream>  struct S {  S() {  std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;  // ...  delete s1;  } |

| **Compliant Code** |
| --- |
| This code will eliminate the call to ::operator delete() which will stop the undefined behavior. |
| #include <iostream>  struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };  void f() {  alignas(struct S) charspace[sizeof(struct S)];  S \*s1 = new (&space) S;  //...  s1->~S();  } |

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

| **Principles(s):** If you take a pointer value and pass it to a deallocation function that was not originally received by a matching allocation function you will create an undefined behavior which can be exploitable. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Clang-analyzer-cplusplus.NewDeleteLeaks-Wmismatched-new-deleteclang-analyzer-unix.MismatchedDeallocator | Clang tidy will check this however it isn’t the most reliable at catching this rules violations |
| Klockwork | 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.MIG |  |
| CodeSonar | 6.0p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM | Free nonheap variable  Double free  Type mismatch |
| Astree | 20.10 | Invalid\_dynamic\_memory\_alocation\_dangling\_pointer\_use |  |
| Axivion Bauhaus Suite | 6.9.0 | CERTC++-MEM51 |  |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| This code uses the assert() macro for asserting a memory-mapped property which is needed for the code to work correctly. |
| #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 code revolves around constant expressions in which a preprocessor conditional statement is allowed. |
| 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):** Static assertions are utilized for locating and destroying software defects that would be found when attempting to compile. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/Rose |  |  | This could also detect violations for the rule by looking for assert() calls and can identify if the code should utilize static assert which would allow ROSE to come in to recognize macro invocation. |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 6.0p0 | (customization) | This would allow the user to implement their own check that could report assert() macro uses. |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| Axivion Bauhaus Suite | 6.9.0 | CERT-DCL03 |  |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemente |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all the thrown exceptions before the execution of main() |

| **Noncompliant Code** |
| --- |
| This code contains the potential for constructor S to throw an exception that wouldnt be noticed when globalS is being made. |
| struct S {  S() noexcept(false);  };  static S globalS; |

| **Compliant Code** |
| --- |
| This code will turn GlobalS into a local variable with a static storage duration which will make it so any exceptions thrown will be caught before execution. |
| struct S {  S() noexcept(false);  };  S &globalS() {  try {  static S s;  return s;  } catch (...) {  }  // Unreachable.  } |

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

| **Principles(s):** Purposely throwing an exception that is unable to be caught will immediately result in a program termination which can then lead into denial of service attacks |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.1 |  |  |
| 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 |
| Rule Checker | 20.10 | Potentially-throwing-static-initialization | Partially checked |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-ERR58-a | An exception will only be raised after the code is started but before the code is terminated. |
| PRQA QA-C++ | 4.4 | 4634, 4636, 4637, 4639 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Do not alternate a file streams inputs and outputs without using an intervening positioning call. |

| **Noncompliant Code** |
| --- |
| The code will append data to the end of the file then continue to read from the same file which as there is no intervening call it causes an undefined behavior. |
| #include <fstream>  #include <string>  void f(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** |
| --- |
| This code has a function be called to be the intervention between the output and input so that the undefined behavior is eliminated. |
| #include <fstream>  #include <string>  void f(const std::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):** You will create an undefined behavior if you input or output from a stream without putting in an intervening flush or position call. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2020a | ECRT C++: FIO50-CPP | This will fully cover the rule by checking for the input and outputs from a stream with no calls or flushes |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP\_FIO50-a | Don’t input or output alternately from stream without a call or flush |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | Don't involve virtual functions from constructors or destructors. |

| **Noncompliant Code** |
| --- |
| This code has the base class attempt to grab and let go of the objects resources however it mixers up the functions. |
| struct B {  B() { seize(); }  virtual ~B() { release(); }  protected:  virtual void seize();  virtual void release();  };  struct D : B {  virtual ~D() = default;  protected:  void seize() override {  B::seize();  // Get derived resources...  }  void release() override {  // Release derived resources...  B::release();  }  }; |

| **Compliant Code** |
| --- |
| This code makes it so constructors and destructers call a non virtual private member function instead of a virtual one. |
| class B {  void seize\_mine();  void release\_mine();  public:  B() { seize\_mine(); }  virtual ~B() { release\_mine(); }  protected:  virtual void seize() { seize\_mine(); }  virtual void release() { release\_mine(); }  };  class D : public B {  void seize\_mine();  void release\_mine();  public:  D() { seize\_mine(); }  virtual ~D() { release\_mine(); }  protected:  void seize() override {  B::seize();  seize\_mine();  }  void release() override {  release\_mine();  B::release();  }  }; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Clang-analyzer-alpha.cplusplus.VirtualCall | Checked by clang tidy |
| Astree | 20.10 | Virtual-call-in-constructorinvalid\_function\_pointer | Fully checked |
| SonarQube C/C++ Plugin | 4.10 | S1699 |  |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-OOP50 |  |
| PVS-Studio | 20.10 | Virtual-call-in-customer | Fully Checked |
| LDRA tool suite | 9.7.1 | 467s, 92D | Fully implemented |
| PRQA QA C++ | 4.4 | 2460, 4261, 4273, 4274, 4275, 4276, 4277, 4278, 4279, 4280, 4281, 4282 |  |
| Parasoft C/C++ test | 2020.2 | CERT CPP-OOP50-a  CERT CPP-OOP50-b  CERT CPP-OOP50-c  CERT CPP-OOP50-d | Avoid calling for virtual functions from constructors  Avoid calling for virtual functions form destructors  Don’t attempt to invoke virtual functions from constructors  Don’t attempt to invoke virtual functions from destructors |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | returning functions must return a value from all its exit paths |

| **Noncompliant Code** |
| --- |
| This code is an example of a programmer forgetting to return a value for a positive input resulting in code paths with an incomplete paths. |
| int absolute\_value(int a) {  if (a < 0) {return -a;  }  } |

| **Compliant Code** |
| --- |
| In this code all paths return a value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  return a;  } |

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

| **Principles(s):** data integrity violations can be caused if a value returning function fails to return a code path value. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS-Studio | 7.07 | V591 |  |
| Axivion Bauhaus Suite | 6.9.0 | CertC++MSC52 |  |
| Astree | 20.10 | Return-implicit | Fully Checked |
| Rule Checker | 20.10 | Return implicit | Fully checked |
| CodeSonar | 6.0p0 | LANG.STRUCT.MRS | Missing the return statement |
| Clang | 3.9 | Wreturn-type | Will not catch all of the instances of this rule |
| Helix QAC | 2021.1 |  |  |
| PRQA QA- C++ | 4.4 | 1510 |  |
| LDRA tool suite | 9.7.1 | 2D, 36S |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-MSC52-a | All the exit paths coming from a non void return type shall always return a return statement with an expression |

### 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.

This diagram gives a detailed description of the system process. The only thing I would add is that I feel the diagram doesn’t cover that it is important to begin and continue testing as early as possible when coding. This would help quickly identify any flaws and vulnerabilities which will make it easier to patch so that less time and resources go into having to fix and move around code.

### 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 | likely | Medium | P6 | L2 |
| STD-009-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-010-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 at rest | This is designed to help prevent the attacker from getting unencrypted data by ensuring that the data on the disk is secure. If an attacker was able to get the disk but not the key then they still must defeat the encryption inorder to get the data. |
| Encryption in flight | This is to protect the data while it is being transmitted somewhere. This will help with data being transmitted around remote locations so that it isn’t attacked before it gets to its disk array |
| Encryption in use | This allows access to data that is encrypted at rest and in motion. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is when the user has been confirmed as a person with access to the system. |
| Authorization | This is the level of access given to a user within their system. This would include what permissions someone has and what they can and cant access. |
| Accounting | The is process where the user is being checked on what they are attempting to do with their access and keep a database on what is being accessed and when to look for any potential violations. |

**\***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 | 07/22/2024 | Coding standards | Morgan Masapollo |  |
| 3.0 | 08/11/2024 | Completed Security Poliicy | Morgan Masapollo |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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