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

Southern New Hampshire University

CS 405 Secure Coding

Professor Mike Prasad

Student: Tyanna Prince

11/12/2022



# 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 | Input validation is used to make sure that only correctly formed data is being entered in the workflow and prevents any malformed data from entering the database and causing malfunction. Input validation should be implemented as early as possible. Any data from an outside source should always be subject to input validation to prevent any compromises or malformed data. However, Input validation should never be used as a primary method for preventing XSS, SQL Injection, or any other possible attacks. |
| 1. Heed Compiler Warnings | Compiler warnings are messages sent from a compiler to inform developers of problems within a programs code fragment. These warnings must not be ignored and instead the developer should try and fix the issue that is causing the warning. These warnings are not errors of the programming language being used, but they can be software bugs. Many compilers can be customized so that the process of compilation is not interrupted. |
| 1. Architect and Design for Security Policies | This looks at how information security controls and safeguards are implemented in an IT system. This is done to protect the confidentiality, integrity, and availability of the data that is used, processed, and stored in those systems. |
| 1. Keep It Simple | Keeping the code, you write small and simple helps to avoid catering to complex systems which can be more susceptible to failed security mechanisms and errors. |
| 1. Default Deny | Standardizing your access decisions and basing them on the permissions rather than exclusion. Default access should be denied with certain conditions that permit access. |
| 1. Adhere to the Principle of Least Privilege | The least number of privileges should be set to process executions. Elevated permissions should only be accessible for the amount of time it would take to complete the task. This will help reduce the possibility of an attack by reducing the use of arbitrary code within the elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize data passed through the complex subsystems to dissuade an attacker from using injection attacks. This prevents the attacker from manipulating any components of the subsystems. |
| 1. Practice Defense in Depth | Defense practice is extremely important. Practice defense in depth with the use of multiple layers of security. This is done so that if one layer was to fail, another layer is there to help in the prevention of any security flaws or exploitable vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques are important to use to increase chances of identifying and vulnerabilities. By doing so, these vulnerabilities can be eliminated. Use multiple testing phases, external security reviews, as well as independent security reviews provide a more secure system. |
| 1. Adopt a Secure Coding Standard | Secure coding standards are a necessity with any program language or platform. Without them there will be many security holes that a hacker can access causing damage. |

### C/C++ Ten Coding Standards

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Does not case to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| Check if a given value is in range of acceptable enumeration values. Once casting the type is done, it may not be representing the integer value. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  EnumType enumVar = static\_cast(intVar);  if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| Checks the value represented by the enumeration type BEFORE performing the conversion. This will guarantee the conversion does not result in an unspecified value. This also restricts the converted value to on specific enumerator type. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  if (intVar < First || intVar > Third) {  // Handle error  }  EnumType enumVar = static\_cast(intVar);  } |

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

| **Principles(s):** There is a possibility for an unspecified value to result in a buffer overflow. This leads to the execution, by an attacker, of arbitrary code. Rarely used are enumerators for indexing into arrays and/or any other forms of pointer arithmetic. However, it’s more likely this scenario will result in data integrity violations instead of arbitrary code execution. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | CertC++ - UNT50 | Software erosion protection |
| Helix QAC | 2021.1 | C++3066, C++3128, C++3166 | static analysis tool in tightly regulated and quality-critical industries |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-INT50-a | automated software testing capabilities are also made for today’s high-velocity Agile DevOps environments. It integrates tightly into your C and C++ IDE, CI/CD pipeline and containerized deployments to detect defects |
| PRQA QA-C++ | 4.4 | 3013/ CON52-CPP | Prevent data races when accessing bit-fields from multiple threads |
| PVS Studio | 7.7 | V1016 | Static code analyzer that detects typos, dead code, potential vulnerabilities (Static Application Security Testing, SAST), and other errors. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | The use of references, iterators, and pointers are required to reference elements of a container. |

| **Noncompliant Code** |
| --- |
| POS is invalidated after 1st call to insert(). Loop iterations have 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){  }  } |

| **Compliant Code** |
| --- |
| Pos is assigned an iterator for each insertion to prevent 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){  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):** The use of invalid pointers, iterators , and references to reference elements of a container will result 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 | Analyze pull-requests to find and fix code quality issues before you merge. |
| Helix QAC | 2021.1 | C++3066, C++3128, C++3166 | static analysis tool in tightly regulated and quality-critical industries |
| Parasoft C/C++ | 2020.2 | CERT\_CPP-CTR51-a | automated software testing capabilities are also made for today’s high-velocity Agile DevOps environments. It integrates tightly into your C and C++ IDE, CI/CD pipeline and containerized deployments to detect defects |
| PVS Studio | 7.7 | V783 | Static code analyzer that detects typos, dead code, potential vulnerabilities (Static Application Security Testing, SAST), and other errors. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Never attempt to create std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| Std::string object created from call to std::getenv() results. Since std::getenv() returns a null pointer the code may have undefined behavior. |
| #include <cstdlib>  #include <string>  void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| Results from call to std::getenv() will be checked for null before the object std::string |
| #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):** Dereferencing a null pointer is undefined behavior. In certain situations, dereferencing a null pointer may cause the execution of arbitrary code. This severity is for severe cases where it’s not possible to exploit a null pointer. In actuality the severity is lower. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Assert Failure | Analyze pull-requests to find and fix code quality issues before you merge. |
| Helix QAC | 2021.1 | C++3066, C++3128, C++3166 | static analysis tool in tightly regulated and quality-critical industries |
| ParasoftC/C++ test | 2020.2 | CERT\_CPP-STR51-a | automated software testing capabilities are also made for today’s high-velocity Agile DevOps environments. It integrates tightly into your C and C++ IDE, CI/CD pipeline and containerized deployments to detect defects |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Never store owned pointer values in a smart pointer that is not related. |

| **Noncompliant Code** |
| --- |
| Two separate smart pointers are made from the same pointer value. When the local variable p2 gets destroyed, it will then delete the pointer value that it manages. After which, the local variable p1 will be destroyed, it will then delete the same pointer value which will result in double free vulnerability. |
| #include <memory>  void f() {  int \*i = new int;  std::shared\_ptr p1(i);  std::shared\_ptr p2(i);  } |

| **Compliant Code** |
| --- |
| Std::shared\_ptr objects are related to each other via copy construction. When the local variable p2 is destroyed the cound for the shared pointer value is lowered but will still be non-zero. After this happened the local variable p1 is destroyed the use count for the shared pointer is lowered to zero and the managed pointer is then destroyed. This will also call std::make\_shared() to prevent the allocation of a raw pointer thus storing its value in a local variable. |
| #include <memory>  void f() {  std::shared\_ptr p1 = std::make\_shared();  std::shared\_ptr p2(p1);  } |

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

| **Principles(s):** Undefined behavior will result if passing a pointer value to a deallocation function that was not previously obtained by the matching allocation. This may cause exploitable vulnerabilities. |
| --- |

**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 | Analyze pull-requests to find and fix code quality issues before you merge. |
| Helix QAC | 2021.1 | C++3066, C++3128, C++3166 | static analysis tool in tightly regulated and quality-critical industries |
| Parasoft C/C++ | 2020.2 | CERT\_CPP-MEM56-a | automated software testing capabilities are also made for today’s high-velocity Agile DevOps environments. It integrates tightly into your C and C++ IDE, CI/CD pipeline and containerized deployments to detect defects |
| PVS Studio | 7.7 | V1006 | Static code analyzer that detects typos, dead code, potential vulnerabilities (Static Application Security Testing, SAST), and other errors. |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| The local variable space will be passed with the expression to the relevant placement of a new operator. This results in the pointer of that particular call then being passed to ::operator delete() which will result in an undefined behavior. This is because ::operator delete() attempts to free the memory that was not returned via ::operator new(). |
| #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** |
| --- |
| Removes the call ::operator delete() but instead calls s1’s destructor. |
| #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;  // …  S1->~S();  } |

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

| **Principles(s):** Undefined behavior will result if passing a pointer value to a deallocation function that was not previously obtained by the matching allocation. This may cause 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 | Analyze pull-requests to find and fix code quality issues before you merge. |
| Helix QAC | 2021.1 | C++3066, C++3128, C++3166 | static analysis tool in tightly regulated and quality-critical industries |
| Axivion Bauhaus Sutie | 6.9 | CERTC++-MEM51 | Software erosion protection |
| CodeSonar | 6.0p | ALLOC.DF  ALLOC.TM  ALLOC.FNH | Defects are persistent and tracked across builds, even if code changes. They can be annotated, ranked, assigned, searched for and compared. Support for many team-tools is provided out of the box. |
| Clang | 3.9 | Clang-analyzer-cplusplus.NewDeleteLeaks-Wmismatched-new-deleteclang-analyzer-unix.MismatchedDeallocator | provides a language front-end and tooling infrastructure for languages in the C language family (C, C++, Objective C/C++, OpenCL, CUDA, and RenderScript) for the [LLVM](https://www.llvm.org/) project. |
| Klocwork | 2021.1 | UNINIT.CTOR.MIGHT  FMM.MIGHT FMM.MUST FNH.MIGHT FNH.MUST FUM.GEN.MIGHT FUM.GEN.MUST  CL.FFM.ASSIGNFM  CL.FFM.COPY  CL.FMM  UNINIT.CTOR.MUST  UNINIT.HEAP.MIGHT  UNINIT.HEAP.MUST  UNINIT.STACK.ARRAY.MIGHT UNINIT.STACK.ARRAY.MUST UNINIT.STACK.MIGHT UNINIT.STACK.MUST | code analysis and [**SAST**](https://www.perforce.com/blog/kw/what-is-sast) tool for C, C++, C#, Java, JavaScript, Python, and Kotlin identifies software security, quality, and reliability issues |
| Parasoft C/C++ | 2020.2 | CERT\_CPP-MEM51-a CERT\_CPP-MEM51-b CERT\_CPP-MEM51-c CERT\_CPP-MEM51-d | automated software testing capabilities are also made for today’s high-velocity Agile DevOps environments. It integrates tightly into your C and C++ IDE, CI/CD pipeline and containerized deployments to detect defects |
| PVS Studio | 7.7 | V783 | Static code analyzer that detects typos, dead code, potential vulnerabilities (Static Application Security Testing, SAST), and other errors. |
| LDRA tool suite | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D | flexible platform for producing safety, security, and mission-critical software in an accelerated, cost effective and requirements driven process. |
| Parasoft Insure++ |  | CERT\_CPP-MEM52-b | detects memory problems, runtime errors, and security vulnerabilities |
| Polyspace Bug Finder | R2020a | CERT C++:MEM51-CPP | Identifies run-time errors, concurrency issues, security vulnerabilities, and other defects in C and C++ embedded software. |
| PRQA QA-C++ | 4.4 | 2110,2111,2112,2113,2118,3337,3339,4262,4263,4264 | Provides support for growing development teams transitioning to modern C++ |
| SonarQubeC/C++ Plugin | 4.10 | S1232 | Detects bugs and code smells in C++ |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| Use assert() to assert a property that concerns a memory mapped structure. This is important for the code to behave properly. |
| #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** |
| --- |
| Assertions that involve only a constant expression you may use a preprocessor conditional statement. Such as: |
| 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):** A valuable diagnostic tool for finding and fixing software defects that can result in vulnerabilities is static assertion. This does not mean the code will be incorrect without the use of static assertion. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool suite | 9.7.1 | 44 S | flexible platform for producing safety, security, and mission-critical software in an accelerated, cost effective and requirements driven process. |
| Axivion Bauhaus Sutie | 6.9 | CERTC-DCL03 | Software erosion protection |
| CodeSonar | 6.0p | Customized | Defects are persistent and tracked across builds, even if code changes. They can be annotated, ranked, assigned, searched for and compared. Support for many team-tools is provided out of the box. |
| Clang | 3.9 | Misc\_Static\_Assert | provides a language front-end and tooling infrastructure for languages in the C language family (C, C++, Objective C/C++, OpenCL, CUDA, and RenderScript) for the [LLVM](https://www.llvm.org/) project. |
| Éclair | 1.2 | CC2.DCL03 | Finds critical defects while software is being coded. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handling any and all exceptions thrown before the main() is executed |

| **Noncompliant Code** |
| --- |
| Constructor for S can throw an exception that isn’t caught if globalS is constructed at program startup |
| struct S {  S() noexcept(false);  };  static S globalS; |

| **Compliant Code** |
| --- |
| GlobalS is made into a local variable with a static storage duration. This allows any exceptions that may be thrown while the object is being constructed to be caught. This is because the constructor S will be executed the very first time globalS() is called. However, this required the programmer to modify the source code so any previous use of globalS is replaced with a call to globalS(). |
| struct S {  S() noexcept(false);  };  S &globalS(){  try {  static S s;  return s;  } catch (...) {  // Handle error, terminating the application.  }  // Unreachable.  } |

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

| **Principles(s):** An exception thrown that can’t be caught will result in abnormal termination. This can lead to denial-of-service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Sutie | 6.9 | CERTC++-ERR58 | Software erosion protection |
| Clang | 3.9 | Cert-eer58-cpp | provides a language front-end and tooling infrastructure for languages in the C language family (C, C++, Objective C/C++, OpenCL, CUDA, and RenderScript) for the [LLVM](https://www.llvm.org/) project. |
| PRQA QA-C++ | 4.4 | 4634, 4636, 4637, 4639 | Provides support for growing development teams transitioning to modern C++ |
| Parasoft C/C++ | 2020.2 | CERT\_CPP-CTR51-a | automated software testing capabilities are also made for today’s high-velocity Agile DevOps environments. It integrates tightly into your C and C++ IDE, CI/CD pipeline and containerized deployments to detect defects |
| Helix QAC | 2021.1 | C++3066, C++3128, C++3166 | static analysis tool in tightly regulated and quality-critical industries |
| Rule Checker | 20.10 | Potentially\_Throwing\_Static\_Initialization | Automatically checks your code for compliance with MISRA rules, CERT recommendations. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Output/Input | [STD-008-CPP] | Never input and output from a file stream without intervening a positioning call |

| **Noncompliant Code** |
| --- |
| Appends data to the end of a file and reads from that same file. Because there is not an intervening positioning call between the output and input calls the behavior will be undefined. |
| #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** |
| --- |
| The std::basic\_istream<T>::seekg() fuction will be called between the input and output which will eliminate any 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"; |

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

| **Principles(s):** Outputting and Inputting from a stream without the use of an intervening flush or a positioning call is undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2020a | ECRT C++: FIO50-CPP Checks for alternating input and | Identifies run-time errors, concurrency issues, security vulnerabilities, and other defects in C and C++ embedded software. |
| Helix QAC | 2021.1 | C++3066, C++3128, C++3166 | static analysis tool in tightly regulated and quality-critical industries |
| Parasoft C/C++ | 2020.2 | CERT\_CPP\_FIO50-a | automated software testing capabilities are also made for today’s high-velocity Agile DevOps environments. It integrates tightly into your C and C++ IDE, CI/CD pipeline and containerized deployments to detect defects |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object oriented programming | [STD-009-CPP] | Never invoke virtual functions from destructors or constructors |

| **Noncompliant Code** |
| --- |
| The base class will attempt to seize and then release the objects resources. This is done through calls to virtual functions from the destructor and constructor. |
| 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** |
| --- |
| Destructors and constructors will call a nonvirtual/private member function instead of calling a virtual one. This makes each class responsible for seizing and releasing its own resources. |
| 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):** Encapsulation, polymorphism, abstraction, and inheritance are the four principles of object-oriented programming. |
| --- |

**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 | Analyze pull-requests to find and fix code quality issues before you merge. |
| Axivion Bauhaus Sutie | 6.9 | CERTC++-OOP50 | Software erosion protection |
| Clang | 3.9 | Clang-analyzer-Alpha.cplusplus.VirtualCall | provides a language front-end and tooling infrastructure for languages in the C language family (C, C++, Objective C/C++, OpenCL, CUDA, and RenderScript) for the [LLVM](https://www.llvm.org/) project. |
| Helix QAC | 2021.1 | C++3066, C++3128, C++3166 | static analysis tool in tightly regulated and quality-critical industries |
| LDRA tool suite | 9.7.1 | 467S, 92D | flexible platform for producing safety, security, and mission-critical software in an accelerated, cost effective and requirements driven process. |
| Parasoft C/C++ | 2020.2 | CERT\_CPP-OOP50-a  CERT\_CPP-OOP50-b  CERT\_CPP-OOP50-c  CERT\_CPP-OOP50-d | automated software testing capabilities are also made for today’s high-velocity Agile DevOps environments. It integrates tightly into your C and C++ IDE, CI/CD pipeline and containerized deployments to detect defects |
| PRQA QA-C++ | 4.4 | 4260, 4261, 4273, 4274, 4275, 4276, 4277, 4278, 4279, 4280, 4281, 4282 | Provides support for growing development teams transitioning to modern C++ |
| PVS Studio | 7.7 | Virtual\_Call\_In\_Customer | Static code analyzer that detects typos, dead code, potential vulnerabilities (Static Application Security Testing, SAST), and other errors. |
| SonarQubeC/C++ Plugin | 4.10 | S1699 | Detects bugs and code smells in C++ |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Misc | [STD-010-CPP] | Value returning functions have to return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| Forgotten return of input value for + input as well as not all the code paths returning a value. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| Code paths will now all return values. |
| 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):** If a value from a code path fails to return in a value-returning function will result in undefined behavior. This may exploit 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 | Analyze pull-requests to find and fix code quality issues before you merge. |
| Axivion Bauhaus Sutie | 6.9 | CERTC++-MSC52 | Software erosion protection |
| Clang | 3.9 | -Wreturn\_Type | provides a language front-end and tooling infrastructure for languages in the C language family (C, C++, Objective C/C++, OpenCL, CUDA, and RenderScript) for the [LLVM](https://www.llvm.org/) project. |
| CodeSonar | 6.0p | Lang.Struct.Mrs | Defects are persistent and tracked across builds, even if code changes. They can be annotated, ranked, assigned, searched for and compared. Support for many team-tools is provided out of the box. |
| Helix QAC | 2021.1 | C++3066, C++3128, C++3166 | static analysis tool in tightly regulated and quality-critical industries |
| LDRA tool suite | 9.7.1 | 2 D, 36 S | flexible platform for producing safety, security, and mission-critical software in an accelerated, cost effective and requirements driven process. |
| PRQA QA-C++ | 4.4 | CERT\_CPP-MSC52-a | Provides support for growing development teams transitioning to modern C++ |
| Polyspace Bug Finder | R2020a | CERT C++: MSC52-a | Identifies run-time errors, concurrency issues, security vulnerabilities, and other defects in C and C++ embedded software. |
| SonarQubeC/C++ Plugin | 4.10 | S935 | Detects bugs and code smells in C++ |
| PRQA QA-C++ | 4.4 | 1510 | Provides support for growing development teams transitioning to modern C++ |
| PVS Studio | 7.7 | V591 | Static code analyzer that detects typos, dead code, potential vulnerabilities (Static Application Security Testing, SAST), and other errors. |
| Rule Checker | 20.10 | Return\_Implicit | Automatically checks your code for compliance with MISRA rules, CERT recommendations. |

### 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.

Defense in depth is something a developer should always be mindful of. Testing early and often will help detect errors, bugs, and vulnerabilities in your code preventing waisted time, money, and attacks from outside sources.

### 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.

| **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | This is designated to keep attackers from accessing any unencrypted data. It does this by ensuring the data is encrypted when placed on a disk. If this data is obtained from an attacker, they much crack the encryption keys in order to read the data. |
| Encryption at flight | This is a process of encrypting data during transmission. For example, remote replication data can be unencrypted while resting on a drives array and encrypted while in transit. |
| Encryption in use | Enabling access to encrypted data at rest and data in motion compromises data in use. This means if a person or persons that have access to random memory can parse that memory and locate the encryption keys for the data at rest. After getting those keys they are able to decrypt the encrypted data at rest. |

| **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The process in which a user is authenticated to access the system. This includes things like log in ID and password information inputted by the user. There are other forms of authentication such as two-step verification where the user inputs that same info and then gets sent a code to their email or phone number that they must enter before accessing the system. |
| Authorization | This the level of access a user would have within the system. This tells the system if the user can read, delete, create, or even modify files within its database. |
| Accounting | This is the process in which the user is monitored on what they are doing within the database. A record is kept of the user’s access and where as well as what the user did and when. It also records the username. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 11/30/2022 | First revision | Tyanna Prince |  |
| 1.2 | 12/11/2022 | Final revision | Tyanna Prince |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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