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



# 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 | This principle is about preventing software invulnerabilities that arise from not validating incoming data both from trusted and untrusted sources. That is to say that validating input data is essential because it keeps threats out of the system. It’s also prudent to be knowledgeable of varying sources of data. |
| 1. Heed Compiler Warnings | As the name states, this principle is about paying attention to any warnings that appear when working with code and compiling it. Warnings that are left unattended can not only expose vulnerabilities, but also be indicative of bigger problems that currently exist or that could potentially develop. It’s important to fix compiler warnings early on before they become a bigger issue. |
| 1. Architect and Design for Security Policies | Here, we are reminded to take security policies into account as you design and build your code. It is prudent to follow security standards and embed them into the architecture and design itself because this method offers the biggest benefits for preventing the occurrence of vulnerabilities. Following this principle also saves time, and money, by allowing for a safer system to be built from the very beginning rather than having to perform constant revisions. |
| 1. Keep It Simple | An essential principle to keep in mind at all moments when working with code because code that is simple to follow is easy to work with and fix or improve upon. Adding security into code that is simple to follow and modify leads to greater protection because it is simple and straightforward to implement as opposed to having to muddle through complicated code. The easier it is to understand something, the easier it is to work with and improve it as needed. |
| 1. Default Deny | Default deny refers to denying access, both incoming and outgoing, to anyone who is not expressly permitted. This achieved by essentially whitelisting those who have been previously granted access. Doing so can potentially keep out anyone who might bring in malicious data and harm the system. |
| 1. Adhere to the Principle of Least Privilege | With this principle, security vulnerabilities are reduced by allowing users to only access what they need to so that they can perform their task. The more access a user has, the greater the risk of an inside attack. This can be prevented by assuring that users that are granted higher access gain that access through proper protocols, which helps to keep out unwanted users and security threats. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data sent to other systems helps to ensure that attacks and other threats are not introduced into another system. Not doing so can lead to vulnerabilities anywhere along the system and expose it to unwanted dangers. Sanitizing data is an easy way to keep a system safe and keep data as secure as possible. |
| 1. Practice Defense in Depth | Defense in Depth helps to ensure security by providing layers of defense. It is important to add layers that provide effective defense as they help to form an overall stronger safeguard method. Following the principle of Defense in Depth is important as hackers have become more resourceful and threats and attacks evolve. Thus, having multiple layers of protection is advisable in case one layer of protection has been breached. |
| 1. Use Effective Quality Assurance Techniques | Using effective quality assurance techniques has many great benefits, chief among them, however, is the early detection and resolution of potential vulnerabilities. This is especially beneficial because a trusted member of the “team” can keep these vulnerabilities a secret and quickly and quietly work on them as opposed to having the risk discovered by an outside user. This principle is crucial to the overall success of a system because the sooner a vulnerability is worked on, the better the outcome. Doing so does not allow hackers, or other malicious entities, access to the system through a flaw that they can exploit. |
| 1. Adopt a Secure Coding Standard | The overall security of the system cannot be implemented without adhering to a secure coding standard, especially in projects that have many people working on it. This principle goes hand in hand with keeping the code simple because it assures uniformity across the entire system. Adopting and adhering to this principle also helps to virtually guarantee savings in both time and money by preventing any issues that might arise from not following this principle. One can think of the effect of “too many chefs in the kitchen” here if not followed, which can lead to unnecessary complications and even lead to exploitable vulnerabilities. |

### 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] | Use valid references, pointers, and iterators to reference elements of a container |

| **Noncompliant Code** |
| --- |
| “Pos” here becomes invalidated after insert() first call, resulting in all loop iterations to have undefined behavior0 |
| #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** |
| --- |
| “Pos” here is assigned a valid iterator for each insertion, which helps avoid 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):** Default Deny – access to data is denied if valid references or pointers are not related to the data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 20.10 | Overflow\_upon\_deference |  |
| Helix QAC | 2021.2 | C++4746, C++4747, C++4748, C++4749 |  |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-CTR51-a | Do not modify container while iterating over it |
| PVS Studio | 7.19 | V783 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| This results in an unspecified behavior because EnumType’s values are [0…3], in which case, if a value outside of that range is passed to f(), the value passed to EnumType would result in an unspecified value, which would cause further error when that value is used within the if statement |
| 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 avoids the unspecified behavior by first checking if the value can be represented by the enumeration type before it performs the conversion, which guarantees that the conversion avoids the unspecified value result |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** Validate Input Data – input data cannot be validated in order to grant access if it is cast to an unspecified result |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| Helix QAC | 2022.2 | C++3013 |  |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| PRQA QA-C++ | 4.4 | 3013 |  |

#### Coding Standard 3

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

| **Noncompliant Code** |
| --- |
| This results in an error because std::string has been created from the results of std::getenv() |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| Here, std::getenv() results get checked for null before constructing 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):** Architecture and Design for Security Policies – Adhere to proper architecture to avoid possible attacks |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Assert\_failure |  |
| CodeSonar | 7.0p0 | LANG.MEM.NPD | Null Pointer Deference |
| Parasoft C/C++test | 2022.1 | CER\_CPP-STR51-a | Avoid null pointer deferencing |
| Helix QAC | 2022.2 | |  |  | | --- | --- | |  | C++4770, C++4771, C++4772, C++4773,  C++4774 | |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CCP] | Do not write syntactically ambiguous declarations |

| **Noncompliant Code** |
| --- |
| This code causes confusion as to whether it is declaring an anonymous object and calling its single-argument converting constructor or whether it is declaring an object “m” and default constructing it |
| #include <mutex>    static std::mutex m;  static int shared\_resource;    void increment\_by\_42() {    std::unique\_lock<std::mutex>(m);    shared\_resource += 42;  } |

| **Compliant Code** |
| --- |
| Here, the lock object has been given an identifier that is not “m” and now the correct converting constructor has been called |
| #include <mutex>    static std::mutex m;  static int shared\_resource;    void increment\_by\_42() {    std::unique\_lock<std::mutex> lock(m);    shared\_resource += 42;  } |

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

| **Principles(s):** Architecture and Design for Security Policies – Adhere to proper architecture to avoid possible attacks  Validating input data – being ambiguous do not help with proper validations |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | LANG.STRUCT.DECL.FNEST | Nested Function Declaration |
| Helix QAC | 2022.2 | C++2502, C++2510 |  |
| LDRA tool suite | 9.7.1 | 296 S | Partially implemented |
| PRQA-QA C++ | 4.4 | 2502, 2510 |  |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| This results in an undefined behavior because ::operator delete() attempts to free memory that was not returned by ::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** |
| --- |
| Here, the call to ::operator delete() is removed and s1’s destructor is called |
| #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):** Adop a secure coding standard – this helps to make code secure and prevents vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2022a | CERT C++: MEM51-CPP | |  |  | | --- | --- | |  | Checks for:   * Invalid deletion of pointer * Invalid free of pointer * Deallocation of previously deallocated pointer   Rule partially covered | |
| Parasoft Insure++ |  |  | Runtime detection |
| SonarQube C/C++Plugin | 4.10 | S1232 |  |
| Axivion Bauhaus Suite | 7.2.0 | CertC++MEM51 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CCP] | Avoid information leakage when passing a class object across a trust boundary |

| **Noncompliant Code** |
| --- |
| This code copies data from “arg” to user space, but it can use padding bits which have the potential to contain sensitive information that can then be leaked when the data is copied to user space |
| #include <cstddef>    struct test {    int a;    char b;    int c;  };    // Safely copy bytes to user space  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void do\_stuff(void \*usr\_buf) {    test arg{1, 2, 3};    copy\_to\_user(usr\_buf, &arg, sizeof(arg));  } |

| **Compliant Code** |
| --- |
| Here, the structure data is serialized before it gets copied to an untrusted context |
| #include <cstddef>  #include <cstring>    struct test {    int a;    char b;    int c;  };    // Safely copy bytes to user space.  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void do\_stuff(void \*usr\_buf) {    test arg{1, 2, 3};    // May be larger than strictly needed.    unsigned char buf[sizeof(arg)];    std::size\_t offset = 0;      std::memcpy(buf + offset, &arg.a, sizeof(arg.a));    offset += sizeof(arg.a);    std::memcpy(buf + offset, &arg.b, sizeof(arg.b));    offset += sizeof(arg.b);    std::memcpy(buf + offset, &arg.c, sizeof(arg.c));    offset += sizeof(arg.c);      copy\_to\_user(usr\_buf, buf, offset /\* size of info copied \*/);  } |

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

| **Principles(s):** Validating Input Data – helps to ensure proper inputs  Architecture and design for security policies – helps to prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL55 |  |
| CodeSonar | 7.0p0 | MICS.PASSING.POTB | Padding passed across a trust boundary |
| Helix QAC | 2022.2 | C++4941, C++4942, C++4943 |  |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-DCL55-a | A pointer to a structure should not be passed to a function that can copy data to the user space |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CCP] | Avoid cycles during initialization of static objects |

| **Noncompliant Code** |
| --- |
| Here, an attempt is made to implement factorial function by using caching, but undefined behavior is created because the static array cache involves recursion |
| #include <stdexcept>    int fact(int i) noexcept(false) {    if (i < 0) {      // Negative factorials are undefined.      throw std::domain\_error("i must be >= 0");    }      static const int cache[] = {      fact(0), fact(1), fact(2), fact(3), fact(4), fact(5),      fact(6), fact(7), fact(8), fact(9), fact(10), fact(11),      fact(12), fact(13), fact(14), fact(15), fact(16)    };      if (i < (sizeof(cache) / sizeof(int))) {      return cache[i];    }      return i > 0 ? i \* fact(i - 1) : 1;  } |

| **Compliant Code** |
| --- |
| This code avoids using the static cache that caused the undefined behavior |
| #include <stdexcept>    int fact(int i) noexcept(false) {     if (i < 0) {      // Negative factorials are undefined.      throw std::domain\_error("i must be >= 0");    }      // Use the lazy-initialized cache.    static int cache[17];    if (i < (sizeof(cache) / sizeof(int))) {      if (0 == cache[i]) {        cache[i] = i > 0 ? i \* fact(i - 1) : 1;      }      return cache[i];    }      return i > 0 ? i \* fact(i - 1) : 1;  } |

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

| **Principles(s):** Architecture and design for security policies – helps to prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | LANG.STRUCT.INIT.CYCLE  LANG.STRUCT.INIT.UNORDERED | Initialization cycle |
| Helix QAC | 2022.2 | C++1552, C++1554, C++1704 |  |
| LDRA tool suite | 9.7.1 | 6D | Enhanced enforcement |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-DCL56-a | Avoid initialization order problems across translation units by replacing non-local static objects with local static objects |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CCP] | Do not define an unnamed namespace in a header file |

| **Noncompliant Code** |
| --- |
| “v” is defined in an unnamed namespace, which causes an unexpected output because each translation unit operates on its own instance of “v” |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE    namespace {  int v;  }    #endif // A\_HEADER\_FILE    // a.cpp  #include "a.h"  #include <iostream>    void f() {    std::cout << "f(): " << v << std::endl;    v = 42;    // ...  }    // b.cpp  #include "a.h"  #include <iostream>    void g() {    std::cout << "g(): " << v << std::endl;    v = 100;  }    int main() {    extern void f();    f(); // Prints v, sets it to 42    g(); // Prints v, sets it to 100    f();    g();  } |

| **Compliant Code** |
| --- |
| Here, “v” has been defined in just one translation unit, where it is externally visible to all translation units |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE    extern int v;    #endif // A\_HEADER\_FILE    // a.cpp  #include "a.h"  #include <iostream>    int v; // Definition of global variable v    void f() {    std::cout << "f(): " << v << std::endl;    v = 42;    // ...  }    // b.cpp  #include "a.h"  #include <iostream>    void g() {    std::cout << "g(): " << v << std::endl;    v = 100;  }    int main() {    extern void f();    f(); // Prints v, sets it to 42    g(); // Prints v, sets it to 100    f(); // Prints v, sets it back to 42    g(); // Prints v, sets it back to 100  } |

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

| **Principles(s):** Architecture and design for security policies – helps to prevent vulnerabilities  Adopt a secure coding standard – prevent vulnerabilities by being mindful of security at all times |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL59 |  |
| Helix QAC | 2022.2 | C++2518 |  |
| LDRA tool suite | 9.7.1 | 286 S, 512 S | Fully implemented |
| PRQA QA C++ | 4.4 | 2518 |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | Do not let exceptions escape from destructors or deallocation functions |

| **Noncompliant Code** |
| --- |
| This results in undefined behavior because the class destructor may throw an exception even if it is called as the result of an exception being thrown |
| #include <stdexcept>    class S {    bool has\_error() const;    public:    ~S() noexcept(false) {      // Normal processing      if (has\_error()) {        throw std::logic\_error("Something bad");      }    }  }; |

| **Compliant Code** |
| --- |
| This code catches exceptions thrown by Bad::~Bad() when bad\_member is destroyed, it also catches exceptions thrown by the “try” block |
| class SomeClass {    Bad bad\_member;  public:    ~SomeClass()    try {      // ...    } catch(...) {      // Catch exceptions thrown from noncompliant destructors of      // member objects or base class subobjects.        // NOTE: Flowing off the end of a destructor function-try-block causes      // the caught exception to be implicitly rethrown, but an explicit      // return statement will prevent that from happening.      return;    }  }; |

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

| **Principles(s):** Architecture and design for security policies – helps to prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL57 |  |
| Helix QAC | 2022.2 | C++2045, C++2047, C++4032, C++4631 |  |
| Klocwork | 2022.2 | MISRA.DTOR.THROW |  |
| PVS-Studio | 7.19 | V509, V1045 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Do not modify namespaces |

| **Noncompliant Code** |
| --- |
| This results in an undefined behavior because the declaration of “x” has been added to the “std” namespace |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| The declaration has been placed into a namespace that does not have a reserved name |
| namespace nonstd {  int x;  } |

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

| **Principles(s):** Architecture and design for security policies – helps to prevent vulnerabilities  Keep it simple – it is best to keep code simple and consistent |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL58 |  |
| PVS Studio | 7.19 | V1061 |  |
| SonarQube C/C++ Plugin | 4.10 | S3470 |  |
| PRQA QA-C++ | 4.4 | 4032, 4035, 4631 |  |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

Automation is essential to future success of the product because updates can be automatically implemented. Functionality can also greatly improve with automation as updated security measures are put into place without having to pause the system and work on it. Implementing automation after release assures a system that will run for years.

### 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 | Probable | High | P6 | L2 |
| STD-002-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-008-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-009-CPP | Low | Unlikely | Medium | P6 | L2 |
| STD-010-CPP | High | Unlikely | Medium | P6 | 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 | This is simply encoding data as it is being written into storage and then decrypting that same data while it is being pulled from storage; this is achieved with a key that is required to both encrypt and decrypt the data. The policy applies because it prevents unauthorized users from accessing the data. |
| Encryption at flight | This is the process of encoding data while it is in transit, or “flight”. The type of “flight” dictates the type of encryption. The policy applies because it assures that information is always securely sent. |
| Encryption in use | This is the process of encoding data as it is being used in some way. This policy applies because encrypting data in use prevents issues when the same data is at rest or in flight. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This refers to using a process to ensure that a user is who they say they are; this can be achieved by using passwords or particular user IDs or other credentials. This policy applies because it prevents unwanted users from accessing the system. |
| Authorization | This refers to the process of authorizing access to specific folders, or other utilities within a program, to authenticated users. This policy applies because only authenticated users can access information within the system, thus keeping out unwanted parties. |
| Accounting | This refers to the process of keeping a log or record of all user’s activity within a system. This allows for easy and quick monitoring of activity and can reveal problem users. This policy applies because it keeps an eye on the entire system and makes it easy to remove unwanted parties. |

**\***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
* 5: until given access, all accounts are denied
* 8: measures already set in place that provide this type of defense
* 10: this is already implemented into the system, easy to maintain
* Firewall logs
* 4: very important to be aware of, this makes the code easier to work on
* 6: this prevents unwanted users from accessing the system
* 7: prevents injection attacks
* 10: this is already implemented into the system, easy to maintain
* Anti-malware logs
* 4: very important to be aware of, this makes the code easier to work on
* 5: until given access, all accounts are denied
* 6: this prevents unwanted users from accessing the system
* 8: measures already set in place that provide this type of defense
* 10: this is already implemented into the system, easy to maintain

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 | 08/07/2021 | Updates were made, all missing information has been included | Antonio Sobalvarro |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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