**Green Pace Developer: Security Policy Guide**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validating input data refers to verifying all input data from sources that are not trusted. This will help to eliminate most outside threats that use this method to exploit software vulnerabilities. |
| 1. Heed Compiler Warnings | Make use of the highest compiler warning levels that will assist you in finding even the slightest error when compiling code. This will help in the long run when small warnings that are not detected by lower-level detection turn into much larger issues further down the line. |
| 1. Architect and Design for Security Policies | When creating software, design it in such a way that encourages and enforces security policies on every possible level. Creating a system that is designed to encourage security policies will result in a more security first system that will prioritize security instead of sacrificing it for other systems. |
| 1. Keep It Simple | Keeping it simple refers to the overall complexity of the code design that may be thought of in the initial stages of design. Although overly complex code may run properly, there is a greater chance that there will be errors that occur due to overlapping or conflicting declarations and statements when trying to update or fix errors. The amount of time that is wasted can be saved as well when just keeping it simple and efficient. |
| 1. Default Deny | Make it so that the default condition is to deny access unless permission is granted. This prioritizes security and prevents initial access to sensitive data. |
| 1. Adhere to the Principle of Least Privilege | Create processes that only require the least number of privileges to accomplish the task. When a process requires more privileges, make it so that it receives it for the least amount of time possible. This helps to prevent too much access to too many system processes at once. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all the data that is sent through complex subsystems to prevent possible attacks invoked through unused functions. |
| 1. Practice Defense in Depth | Mitigate the risk of security breaches by layering defensive strategies on top of each other. The way this works is if one of the security measures fails, there is another one right behind it. By layering multiple securities in such a way, the overall defense of the system is better ensured. |
| 1. Use Effective Quality Assurance Techniques | High quality assurance techniques can help to identify vulnerabilities that may be easily missed. Fuzz and penetration testing along with source code audits all incorporated together are considered an effective assurance program. |
| 1. Adopt a Secure Coding Standard | Make sure to always create or apply a secure coding standard for development language and platform that will be used. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not cast to an out-of-range enumeration value. (INT50-CPP)  Enumerations come in both scoped that is fixed and unscoped form that can be either fixed or not. The range of values contained by both may include values that are not specified by the enumeration. |

| **Noncompliant Code** |
| --- |
| This example is trying to check if the value given is within an acceptable range but is doing so after it has already been casted to the enumeration type. If the value turns out to be outside the designated range, then it would result in an unspecified value and an unspecified behavior. |
| **enum** EnumType {    First,    Second,    Third  };    **void** f(**int** intVar) {    EnumType enumVar = **static\_cast**<EnumType>(intVar);    **if** (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| The example below shows how to properly check if the given value is within the acceptable range by first checking if it can be represented by the enumeration type before performing the conversion. This method helps to guarantee that there will not be n unspecified value. |
| **enum** EnumType {    First,    Second,    Third  };    **void** f(**int** intVar) {  **if** (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = **static\_cast**<EnumType>(intVar);  } |

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

| **Principles(s):** It’s possible for unspecified values to result in buffer overflow, which may lead to an attacker executing arbitrary code. |
| --- |

**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 | 2023.3 | C++3013 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| RuleChecker | 22.10 | Cast-integer-to-enum | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not access freed memory. (MEM50-CPP)  Evaluating a pointer into memory which has already been deallocated by a previous function is considered a “dangling pointer” and specified as undefined behavior. When accessing the data in a dangling pointer there is a risk of it resulting in exploitable vulnerabilities. |

| **Noncompliant Code** |
| --- |
| In this example, the s is dereferenced and deallocated. The vulnerability may occur if there is a write-after-free which can run arbitrary code within the permissions because of the vulnerable process. This memory allocation should have been removed to prevent this possible issue. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| In this example the dynamically allocated memory is not deallocated until after it is no longer required. This helps to prevent the write-after-free scenario in the noncompliant version. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...    s->f();  **delete** s;  } |

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

| **Principles(s):** The reading of a dynamically allocated memory that has already been deallocated may lead to abnormal program termination and denial-of-service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion  Bauhaus Suite | 7.2.0 | CertC++-MEM50 |  |
| CodeSonar | 8.0p0 | ALLOC.UAF | Use after free |
| Coverity | V77.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once |
| Helix QAC | 2023.3 | C++4304, C++4304 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Use valid references, pointers, and iterators to reference elements of a basic\_string. (STR52-CPP)  Std::basic\_string is a container of characters which supports iterators just like other containers found in Standard Template Library. |

| **Noncompliant Code** |
| --- |
| This example copies the input into a std::string and replaces the semicolon characters with spaces. This is noncompliant since loc is now invalidated after the first insert() call. |
| #include <string>    **void** f(**const** std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();  **for** (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

| **Compliant Code** |
| --- |
| This version of the example is compliant since loc is being updated with each call to insert(). |
| #include <string>    **void** f(**const** std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();  **for** (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      loc = email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

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

| **Principles(s):** The use of an invalid reference, pointer, or iterator to a string could possibly allow an attacker the ability to run arbitrary code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | ALLOC.UAF | Use After Free |
| Helix QAC | 2023.3 | DF4746, DF4747, DF4748, DF4749 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP\_STR52-a | Use valid references, pointers, and iterators to reference elements of a basic\_string |
| Polyspace Bug Finder | R2023b | CERT C++: STR52-CPP | Checks for use of invalid string iterator |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Do not write syntactically ambiguous declarations. (DCL53-CPP)  Write your code in a way that can only be read and understood one way so that it is not ambiguously read in a different way or format. |

| **Noncompliant Code** |
| --- |
| In this example the variable std::unique\_lock is meant to be used to lock and unlock the mutex m but the way it is a written could also suggest that an anonymous object is called. |
| #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** |
| --- |
| This version of the code shows that the lock object is given an identifier and in turn properly converted the constructor that is 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):** Syntactically ambiguous declarations can lead to unexpected program execution. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.STRUCT.DECL.FNEST | Nested Function Declaration |
| LDRA tool suite | 9.7.1 | 296 S | Partially Implemented |
| Klocwork | 2023.3 | CERT.DCL.AMBIGUOUS\_DECL |  |
| SonarQube C/C++ Plugin | 4.10 | S3468 |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Detect and handle memory allocation errors. (MEM52-CPP)  When using ::operator new(std::size\_t) it throws an exception if the allocation fails. When using a nonthrowing form though, an exception is not thrown in failure but instead it returns nullptr. |

| **Noncompliant Code** |
| --- |
| In this example, since ::operator new[](std::size\_t) is used the allocation is not checked. The user assumes that there will be no exception thrown but since it can throw an exception if failure occurs it could lead to and abnormal termination of the program. |
| #include <cstring>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** **int**[size];    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

| **Compliant Code** |
| --- |
| Since std::nothrow is being used, it can either return a null pointer or a pointer to an allocated space. |
| #include <cstring>  #include <new>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** (std::**nothrow**) **int**[size];  **if** (!copy) {      // Handle error  **return**;    }    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

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

| **Principles(s):** Failing to detect allocation failures may lead to abnormal program termination and denial-of-service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Find inconsistencies in how function call return values are handled |
| LDRA tool suite | 9.7.1 | 45 D | Partially implemented |
| Parasoft Insure++ |  |  | Runtime detection |
| PVS-Studio | 7.29 | V522, V668 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Avoid information leakage when passing a class object across a trust boundary. (DCL55-CPP)  Make sure that when passing a pointer to a class object instance across a trust boundary that there is no sensitive information being transferred over. |

| **Noncompliant Code** |
| --- |
| This example runs in kernel space and copies the data from arg to the user space. This does not prevent padding bits from containing sensitive information. |
| #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** |
| --- |
| In this method the solution serializes the structure data before it begins to copy it over to another source. This helps to ensure that there are no uninitialized padding bits with sensitive information. |
| #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):** Padding bits might inadvertently contain sensitive data such as pointers to kernel data structures or passwords. |
| --- |

**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 | 8.0p0 | MISC.PADDING.POTB | Padding Passed Across a Trust Boundary |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-DLC55-a | A pointer to a structure should not be passed to a function that is able to copy data to the user space |
| Polyspace Bug Finder | R2023b | CERT C++: DCL55-CPP | Checks for information leakage due to structure padding |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions. (ERR51-CPP)  All exceptions thrown by an application must be caught by a matching exception handler. |

| **Noncompliant Code** |
| --- |
| In this example throwing\_func() is not caught by either f() or main() which causes std::terminate() to be called. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| For this example the main entry point handles all the exceptions. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {  **try** {      f();    } **catch** (...) {      // Handle error    }  } |

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

| **Principles(s):** Allowing the application can lead to resources that are not being freed, closed, etc. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Helix QAC | 2023.3 | C++4035, C++4036, C++4037 |  |
| Klocwork | 2023.3 | MISRA.CATCH.ALL |  |
| RuleChecker | 22.10 | Main-function-catch-all early-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous | [STD-008-CPP] | Ensure your random number generator is properly seeded. (MSC51-CPP)  When calling a pseudorandom number generator (PRNG) without seeding or seeding with a constant value, the same sequence of random numbers becomes predictable and is now vulnerable. |

| **Noncompliant Code** |
| --- |
| This example only generates a sequence of 10 pseudorandom numbers which will always produce the same sequence. |
| #include <random>  #include <iostream>    **void** f() {    std::mt19937 engine;    **for** (**int** i = 0; i < 10; ++i) {      std::cout << engine() << ", ";    }  } |

| **Compliant Code** |
| --- |
| This example makes use of std::random\_device to help generate random value. |
| #include <random>  #include <iostream>    **void** f() {    std::random\_device dev;    std::mt19937 engine(dev());    **for** (**int** i = 0; i < 10; ++i) {      std::cout << engine() << ", ";    }  } |

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

| **Principles(s):** Ensure that a PRNG is always properly seeded with an initial seed value that won’t be predictable or controllable by a possible attacker. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | HARDCODED.SEED.MISC.CRYPTO.TIMESEED | Hardcoded Seed in PRNG Predictable Seed in PRNG |
| Helic QAC | 2023.3 | C++5041 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MSC51-a | Properly seed pseudorandom number generators |
| RuleChecker | 22.10 | Default-construction | Partially checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data Input | [STD-009-CPP] | Close files when they are no longer needed. (FIO51-CPP)  When there is a call to open a file, there must also always be a call to close said file before the program is terminated. |

| **Noncompliant Code** |
| --- |
| In this example, the object file is called to open but there is no call to close it prior to the programs termination. |
| #include <exception>  #include <fstream>  #include <string>    **void** f(**const** std::string &fileName) {    std::fstream file(fileName);  **if** (!file.is\_open()) {      // Handle error  **return**;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| This example shows the file being called to close prior to the termination call. |
| #include <exception>  #include <fstream>  #include <string>    **void** f(**const** std::string &fileName) {    std::fstream file(fileName);  **if** (!file.is\_open()) {      // Handle error  **return**;    }    // ...    file.close();  **if** (file.fail()) {      // Handle error    }    std::terminate();  } |

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

| **Principles(s):** Failing to properly close files may allow an attacker to exhaust system resources and can increase risk that data written into in-memory buffers will not be flushed in the event of abnormal program termination. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | ALLOC.LEAK | Leak |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2023b | CERT C++: FIO51-CPP | Checks for resource leak |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data | [STD-010-CPP] | Never qualify a reference type with const or volatile. (DCL52-CPP)  The value of a reference type cannot be changed and is treated as a const. |

| **Noncompliant Code** |
| --- |
| In this example p is correctly declared but the attempted change does not work. |
| #include <iostream>    **void** f(**char** c) {  **const** **char** &p = c;    p = 'p'; // Error: read-only variable is not assignable    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| This example shows the const qualifier being removed. |
| #include <iostream>    **void** f(**char** c) {  **char** &p = c;    p = 'p';    std::cout << c << std::endl;  } |

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

| **Principles(s):** A const or volatile reference type may result in undefined behavior instead of fatal diagnostic which causes unexpected values to be stored and leading to possible data integrity violations. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL52 |  |
| Helic QAC | 2023.3 | C++0014 |  |
| Klocwork | 2023.2 | CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-DCL52-a | Never qualify a reference type with ‘const’ or ‘volatile’ |

### 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.

The preproduction phase of the design is extremely important for various reasons. In the preproduction phase, the more stress on security, the less issues you will have in the production phase since most of the security steps will have already been implemented. I believe that the assessment and plan phase should be split into two, where one is focused more on development and the second is focused on the security planning. From there it will continue towards the design phase which will figure out the proper security needs that should be used depending on the threats that were previously assessed. It will proceed to build, then verify and test in accordance with all previous designs and plans before it is released into production. While in the production phase there are several steps that are taken to monitor and maintain the security and stability of the program. The production phase will basically verify that everything is still working as it should and report any issues that may have occurred so that they can be handled properly and sent back through the preproduction phase is issues occur. If any adjustments need to be made, I would suggest adding another testing phase between “transition and health check” and “monitor and detect” to double check that no new issues came up just before launching.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-002-CPP | High | Likely | Medium | P18 | L1 |
| STD-003-CPP | High | Probable | High | P6 | L2 |
| 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 | Probable | Medium | P4 | L3 |
| STD-008-CPP | Medium | Likely | Low | P18 | L1 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | Low | Unlikely | Low | P3 | L3 |

### 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 | Encryption at rest refers to the encryption of data that is typically held on a disk or a hard drive. This prevents the attacker from gaining access to the data on the disk even if they obtain the physical disk or hard drive. |
| Encryption at flight | Encryption at flight refers to the encryption of data that is in the process of being transmitted so it is considered “in flight”. This data is encrypted while in flight even if it was not encrypted while at rest. |
| Encryption in use | Encryption in use refers to the encryption of data while currently in use. This is a method of protecting the sensitive data that may be in use by encrypting access to them. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The authentication process helps to ensure that every time a user attempts to access sensitive information from the system, their identity will need to be verified via the credentials given to them. For example, a username and password is a way for the system to verify credentials at a low level. |
| Authorization | The authorization process goes along with authentication by providing the user with the credentials needed to access the information needed. This comes in levels and may only be given to users that meet the requirements for those credentials. An example of this would be systems that require different levels of security clearance such as confidential, secret, and top secret. |
| Accounting | The accounting process helps to tie the Triple-A framework together by monitoring both authentication and authorization. With the use of accounting, we can monitor how each user is using the credentials that are given to them by seeing how many times they have attempted access, were granted access, and denied access. This can be useful to see if any users are abusing credentials by trying to gain access to data that they do not have authorization for. |

**\***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 | 01/28/2024 | Coding Standard | Luis Arroyo |  |
| 3.0 | 02/17/2024 | Risk Assessment | Luis Arroyo |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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