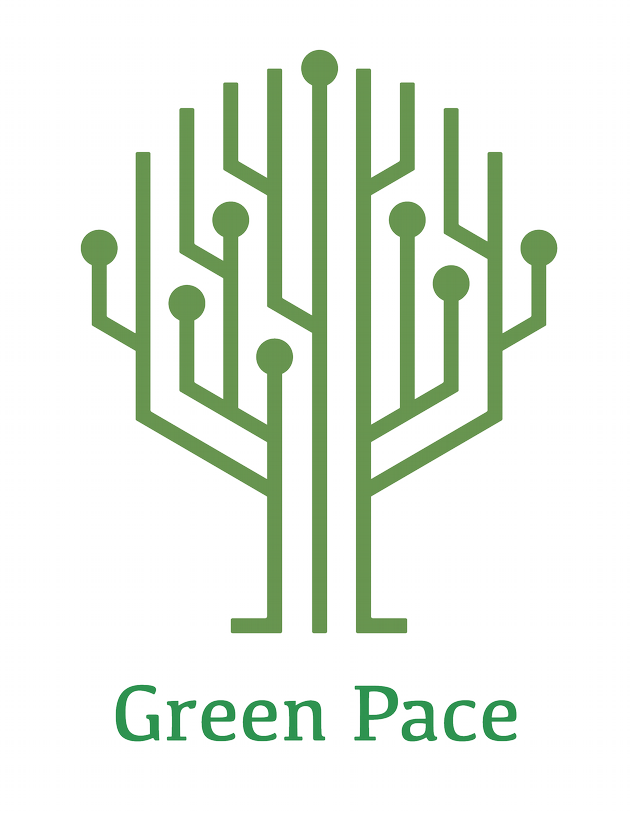
**Green Pace Developer: Security Policy Guide**



## Green Pace Secure Development Policy

Contents

[Overview 2](#_bookmark0)

[Purpose 2](#_bookmark1)

[Scope 2](#_bookmark2)

[Module Three Milestone 2](#_bookmark3)

[Ten Core Security Principles 2](#_bookmark4)

[C/C++ Ten Coding Standards 3](#_bookmark5)

[Coding Standard 1 4](#_bookmark6)

[Coding Standard 2 5](#_bookmark7)

[Coding Standard 3 6](#_bookmark8)

[Coding Standard 4 7](#_bookmark9)

[Coding Standard 5 8](#_bookmark10)

[Coding Standard 6 9](#_bookmark11)

[Coding Standard 7 10](#_bookmark12)

[Coding Standard 8 11](#_bookmark13)

[Coding Standard 9 13](#_bookmark14)

[Coding Standard 10 14](#_bookmark15)

[Defense-in-Depth Illustration 15](#_bookmark6)

[Project One 15](#_bookmark16)

1. [Revise the C/C++ Standards 15](#_bookmark17)
2. [Risk Assessment 15](#_bookmark18)
3. [Automated Detection 15](#_bookmark19)
4. [Automation 15](#_bookmark20)
5. [Summary of Risk Assessments 16](#_bookmark21)
6. [Create Policies for Encryption and Triple A 16](#_bookmark22)
7. [Map the Principles 17](#_bookmark23)

[Audit Controls and Management 18](#_bookmark24)

[Enforcement 18](#_bookmark25)

[Exceptions Process 18](#_bookmark26)

[Distribution 19](#_bookmark27)

[Policy Change Control 19](#_bookmark28)

[Policy Version History 19](#_bookmark29)

[Appendix A Lookups 19](#_bookmark30)

[Approved C/C++ Language Acronyms 19](#_bookmark31)

# 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,](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/) [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.** Validate Input Data | To try and eliminate as many vulnerabilities as possible validating incoming data from untrusted sources and even trusted sources just to verify no malicious data is coming through the system. |
| **2.** Heed Compiler Warnings | Compiling and testing the code as much as possible to try and keep the code as tight as possible. A warning might be just an annoyance, but it also could be something serious, that means check them all as much as possible just in case. |
| **3.** Architect and Design for Security Policies | If you design your code as much as possible with security in mind while still meeting the standard for what the code is ultimately supposed to do, you can save a ton of time later either building safeguards in or fixing vulnerabilities after they are found. |
| **4.** Keep It Simple | If you keep the code as clean and simple as possible it makes everything about it easier, including security of that code. This is a core principle for life in general and it applies even more for security as the easier something is to understand, the easier it is to protect. |
| **5.** Default Deny | Denial as a standard is a good idea as it prevents anyone who should not have access to something from gaining access, while at the same time ensuring those who actually need access go through the proper channels to gain that access. |
| **6.** Adhere to the Principle of Least Privilege | Principle of least privilege goes perfect with default denial, as it sets a standard of only those who need higher access are going to go through the proper channels to gain that access. While going through these processes of gaining access it will be expressly implemented that only the access that is truly needed is given thus limiting possible risk by having less information available in general. |

|  |  |
| --- | --- |
| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| **7.** Sanitize Data Sent to Other Systems | Ensuring that any data sent to other systems is sanitized of any possible vulnerabilities and/or sensitive information is crucial. Sending code with such information could lead to exploits which could have any number of terrible outcomes depending on the situation. |
| **8.** Practice Defense in Depth | Most systems require more than just one layer of protection, combining layers of redundant and varying defense can help safeguard a system that would otherwise suffer from a breach and cause chaos. |
| **9.** Use Effective Quality Assurance Techniques | Quality Assurance is a huge part of every process, never popular by design as they are purely meant to find issues with any process/design but are still crucial to the end result. Better to have someone that is on the team find any issues/vulnerabilities than it be discovered by a customer or possibly a malicious actor which can have much harsher consequences. |
| **10.** Adopt a Secure Coding Standard | Having a secure coding standard goes along perfect with other principles such as designing your code with security policies in mind and having effective quality assurance techniques. Having these ideas in the forefront as much as possible is a great way to save time and prevent problems from popping up down the road that most likely will cost double what they would have on the front end. |

## 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] | Defining C-style variadic functions can possibly lead to vulnerabilities because they don’t check arguments being passed. |

**Noncompliant Code**

This functions is designed to read the values until 0 value is found or if it isn’t found after two arguments it can cause issues.

#include <cstdarg>

int add(int first, int second, ...) { int r = first + second;

va\_list va; va\_start(va, second);

while (int v = va\_arg(va, int)) { r += v;

}

va\_end(va); return r;

}

**Compliant Code**

This function has a built in add statement that helps prevent the above issues

#include <type\_traits>

template <typename Arg, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>

int add(Arg f, Arg s) { return f + s; }

template <typename Arg, typename... Ts, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr> int add(Arg f, Ts... rest) {

return f + add(rest...);

}

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

**Principles(s):**

Validating Input Data – ensuring proper inputs

Architect and Design for Security Policies – building code to prevent vulnerabilities

Keep it simple – always applies as keeping code as lightweight as possible is best practice

Use Effective Quality Assurance Techniques – making tests that are as effective as possible Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

**Threat Level**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| High | Probable | Medium | **P12** | **L1** |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **function-ellipsis** | Fully checked |
| [Axivion](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite)  [Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite) | 6.9.0 | **CertC++-DCL50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-dcl50-cpp | Checked by clang-tidy. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 5.4p0 | **LANG.STRUCT.ELLIPSIS** | Ellipsis |

## Coding Standard 2

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| **Data Value** | [STD-002- CPP] | Defining a reserved identifier incorrectly can cause issues as it will not really be reserved. |

**Noncompliant Code**

Naming standards not met and causes undefined behavior.

#ifndef \_MY\_HEADER\_H\_ #define \_MY\_HEADER\_H\_

// Contents of <my\_header.h> #endif // \_MY\_HEADER\_H\_

**Compliant Code**

By removing the trailing and leading underscores it prevents the issue.

#ifndef MY\_HEADER\_H #define MY\_HEADER\_H

// Contents of <my\_header.h> #endif // MY\_HEADER\_H

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

**Principles(s):**

Heed compiler warnings – always pay attention to warnings as they are there for a reason Architect and Design for Security Policies – building code to prevent vulnerabilities

Keep it simple – always applies as keeping code as lightweight as possible is best practice Use Effective Quality Assurance Techniques – making tests that are as effective as possible Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [Insert text.] | [Insert  text.] | [Insert text.] | [Insert text.] |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **reserved-identifier** | Partially checked |
| [Axivion](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite)  [Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite) | 6.9.0 | **CertC++-DCL51** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreserved-id-macro  -Wuser-defined-literals | The -Wreserved-id-macro flag is not enabled by default or with  -Wall, but is enabled with  -Weverything. This flag does not catch all instances of this rule, such as redefining reserved  names. |

## Coding Standard 3

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| **String Correctness** | [STD-003- CPP] | Never qualify a reference type with const or volatile. cv-qualifying a reference type will result in undefined behavior. A compiler should deliver a fatal diagnostic and if not it may produce surprising results. |

**Noncompliant Code**

A const-qualified reference to a char is formed instead of a reference to a const-qualified char

#include <iostream>

void f(char c) { char &const p = c; p = 'p';

std::cout << c << std::endl;

}

**Compliant Code**

Remove the const qualifier to prevent such an issue from occurring.

#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):**

Heed compiler warnings – always pay attention to warnings as they are there for a reason Architect and Design for Security Policies – building code to prevent vulnerabilities

Keep it simple – always applies as keeping code as lightweight as possible is best practice Use Effective Quality Assurance Techniques – making tests that are as effective as possible Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [Axivion](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite)  [Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite) | 6.9.0 | **CertC++-DCL52** |  |
| [Parasoft C/C+](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft)  [+test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-DCL52-a** | Never qualify a reference type  with 'const' or 'volatile' |
| [Polyspace Bug](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace%2BBug%2BFinder) [Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace%2BBug%2BFinder) | R2020a | [CERT C++: DCL52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl52cpp.html) | Checks for:   * const-qualified reference types * Modification of const- qualified reference types   Rule fully covered. |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **0014** |  |

## Coding Standard 4

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| **SQL Injection** | [STD-004- CPP] | Do not write syntactically ambiguous declarations. Write code that can only be understood one way. |

**Noncompliant Code**

This argument can be taken to declare an anonymous object and calling its single-argument converting constructor or interpreted as declaring an object named 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**

The lock is given an identifier and proper converting constructor 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):**

Validating Input Data – ensuring proper inputs

Architect and Design for Security Policies – building code to prevent vulnerabilities

Keep it simple – always applies as keeping code as lightweight as possible is best practice Use Effective Quality Assurance Techniques – making tests that are as effective as possible Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **296 S** | Partially implemented |
| [Parasoft C/C+](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft)  [+test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-DCL53-a CERT\_CPP-DCL53-b** | Always declare functions at file scope  Identifier declared in a local or function prototype scope shall not hide an identifier declared in a  global or namespace scope |
| [Polyspace Bug](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace%2BBug%2BFinder) [Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace%2BBug%2BFinder) | R2020a | [CERT C++: DCL53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl53cpp.html) | Checks for declarations that can be confused between:   * Function and object declaration * Unnamed object or function parameter declaration   Rule fully covered. |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2502, 2510** |  |

## Coding Standard 5

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| **Memory Protection** | [STD-005- CPP] | Overload allocation and deallocation functions as a pair in the same scope. Failure to so will likely cause undefined behavior. |

**Noncompliant Code**

The allocation is overloaded at a global scale but there is no deallocation function declared.

#include <Windows.h> #include <new>

void \*operator new(std::size\_t size) noexcept(false) {

static HANDLE h = ::HeapCreate(0, 0, 0); // Private, expandable heap. if (h) {

return ::HeapAlloc(h, 0, size);

}

throw std::bad\_alloc();

}

// No corresponding global delete operator defined.

**Compliant Code**

The deallocation is declared which should prevent the overload condition.

#include <Windows.h> #include <new>

class HeapAllocator { static HANDLE h; static bool init;

public:

static void \*alloc(std::size\_t size) noexcept(false) { if (!init) {

h = ::HeapCreate(0, 0, 0); // Private, expandable heap. init = true;

}

if (h) {

return ::HeapAlloc(h, 0, size);

}

throw std::bad\_alloc();

}

static void dealloc(void \*ptr) noexcept {

**Compliant Code**

if (h) {

(void)::HeapFree(h, 0, ptr);

}

}

};

HANDLE HeapAllocator::h = nullptr; bool HeapAllocator::init = false;

void \*operator new(std::size\_t size) noexcept(false) { return HeapAllocator::alloc(size);

}

void operator delete(void \*ptr) noexcept { return HeapAllocator::dealloc(ptr);

}

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

**Principles(s):**

Validating Input Data – ensuring proper inputs

Architect and Design for Security Policies – building code to prevent vulnerabilities

Keep it simple – always applies as keeping code as lightweight as possible is best practice Use Effective Quality Assurance Techniques – making tests that are as effective as possible Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **new-delete-pairwise** | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | misc-new-delete-overloads | Checked with clang-tidy. |
| [Parasoft C/C+](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft)  [+test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-DCL54-a** | Always provide new and delete  together |
| [Polyspace Bug](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace%2BBug%2BFinder) [Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace%2BBug%2BFinder) | R2020a | [CERT C++: DCL54-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl54cpp.html) | Checks for mismatch between overloaded operator new and operator delete (rule fully  covered) |

## 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. The data passing needs to be verified before it can cause issues. |

**Noncompliant Code**

The data when transferred regardless of means may contain sensitive information in this example.

#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**

This serializes the structure data before copying it which should prevent these types of problems.

#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;

**Compliant Code**

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 – ensuring proper inputs

Architect and Design for Security Policies – building code to prevent vulnerabilities

Sanitize data sent to other systems – keeping data sent to other systems as only required/authorized Keep it simple – always applies as keeping code as lightweight as possible is best practice

Use Effective Quality Assurance Techniques – making tests that are as effective as possible Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [Axivion](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite)  [Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite) | 6.9.0 | **CertC++-DCL55** |  |
| [Parasoft C/C+](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft)  [+test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **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- CPP] | Avoid cycles during initialization of static objects. If a function is reentered during initialization of a static object, the behavior will be undefined. |

**Noncompliant Code**

This attempts to implement factorial function utilizing caching but the initialization of the static array cache involves recursion creating 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");

}

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 does not utilize the static cache which is the thing causing the issue.

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]) {

**Compliant Code**

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):**

Architect and Design for Security Policies – building code to prevent vulnerabilities

Keep it simple – always applies as keeping code as lightweight as possible is best practice Use Effective Quality Assurance Techniques – making tests that are as effective as possible Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **6 D** | Enhanced Enforcement |
| [Parasoft C/C+](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft)  [+test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **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- CPP] | Do not let exceptions escape from destructors or deallocation functions. |

**Noncompliant Code**

The class destructor may throw an exception and cause undefined behavior.

#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 will catch any exceptions and destroy them also.

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):**

Architect and Design for Security Policies – building code to prevent vulnerabilities

Keep it simple – always applies as keeping code as lightweight as possible is best practice Use Effective Quality Assurance Techniques – making tests that are as effective as possible Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **destructor-without-noexcept**  **delete-without-noexcept** | Fully checked |
| [Axivion](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite)  [Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite) | 6.9.0 | **CertC++-DCL57** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **453 S** | Partially implemented |
| [Parasoft C/C+](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft)  [+test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-DCL57-a CERT\_CPP-DCL57-b** | Never allow an exception to be thrown from a destructor, deallocation, and swap  Always catch exceptions |

## Coding Standard 9

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| [Student Choice] | [STD-009- CPP] | Do not modify the standard namespaces. Introducing new declarations in the namespace can cause undefined behavior when not utilized correctly. |

**Noncompliant Code**

x is added to the namespace causing undefined behavior

namespace std { int x;

}

**Compliant Code**

By placing without a reserved name this does not cause undefined behavior.

namespace nonstd { int x;

}

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

**Principles(s):**

Architect and Design for Security Policies – building code to prevent vulnerabilities

Keep it simple – always applies as keeping code as lightweight as possible is best practice Use Effective Quality Assurance Techniques – making tests that are as effective as possible Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

**Threat Level**

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

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [Axivion](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite)  [Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite) | 6.9.0 | **CertC++-DCL58** |  |
| [Parasoft C/C+](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft)  [+test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-DCL58-a** | Do not modify the standard  namespaces 'std' and 'posix' |
| [Polyspace Bug](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace%2BBug%2BFinder)  [Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace%2BBug%2BFinder) | R2020a | [CERT C++: DCL58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl58cpp.html) | Checks for modification of  standard namespaces (rule fully |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
|  |  |  | covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **4032, 4035, 4631** |  |

## Coding Standard 10

|  |  |  |
| --- | --- | --- |
| **Coding Standard** | **Label** | **Name of Standard** |
| [Student Choice] | [STD-010- CPP] | Do not define an unnamed namespace in a header file. Utilizing an unnamed namespace in a header file can cause issues. |

**Noncompliant Code**

the variable is defined in an unnamed namespace and as a result each translation unit operates on its own instance.

// 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();

## Noncompliant Code

}

**Compliant Code**

The variable is defined by one translation unit but visible to all and results in the expected output.

// 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):**

Architect and Design for Security Policies – building code to prevent vulnerabilities

Keep it simple – always applies as keeping code as lightweight as possible is best practice

Use Effective Quality Assurance Techniques – making tests that are as effective as possible Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

**Threat Level**

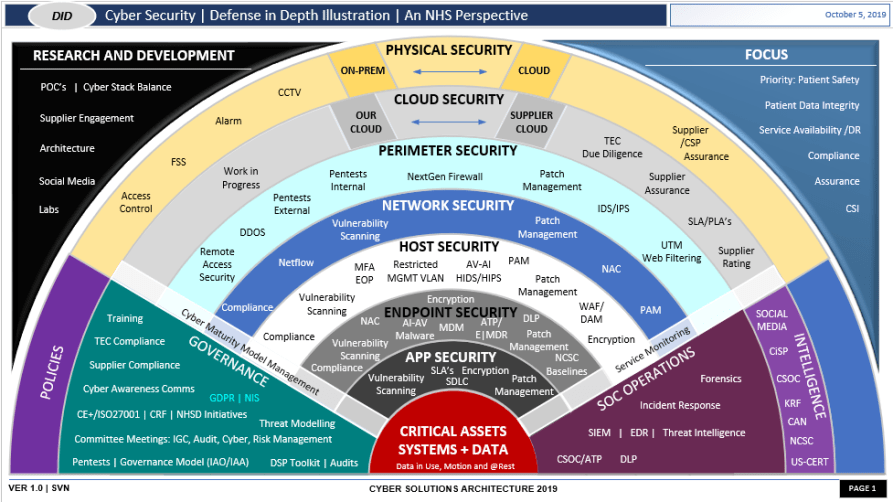
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| Medium | Unlikely | Medium | **P4** | **L3** |

**Automation**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description Tool** |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **unnamed-namespace-header** | Fully checked |
| [Axivion](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite)  [Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion%2BBauhaus%2BSuite) | 6.9.0 | **CertC++-DCL59** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-dcl59-cpp | Checked by clang-tidy |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **286 S, 512 S** | Fully implemented |

## 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

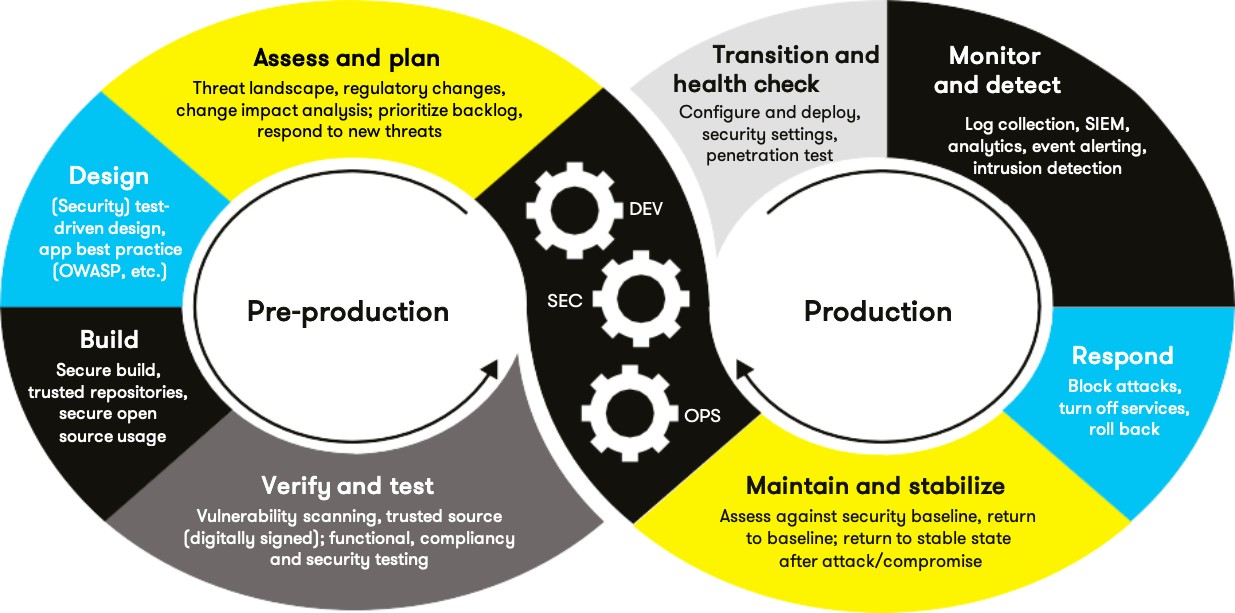
Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, 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 always a good idea throughout the entire process, to include after final release of the product. The product should always be in the process of updating and improving as more information and data on how the product function is gathered.

## 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 | Medium | **P12** | **L1** |
| STD-002-  CPP | Low | Unlikely | Low | **P3** | **L3** |
| STD-003-  CPP | Low | Unlikely | Low | **P3** | **L3** |
| STD-004-  CPP | Low | Unlikely | Medium | **P2** | **L3** |
| STD-005-  CPP | Low | Probable | Low | **P6** | **L2** |
| STD-006-  CPP | Low | Unlikely | High | **P1** | **L3** |
| STD-007-  CPP | Low | Unlikely | Medium | **P2** | **L3** |
| STD-008-  CPP | Low | Likely | Medium | **P6** | **L2** |
| STD-009-  CPP | High | Unlikely | Medium | **P6** | **L2** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Rule | Severity | Likelihood | Remediation  Cost | Priority | Level |
| STD-010-  CPP | Medium | Unlikely | Medium | **P4** | **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.

|  |  |
| --- | --- |
| **a. Encryptio n** | **Explain what it is and how and why the policy applies.** |
| Encryption in rest | Encryption for data at rest is the process of securely encoding data as it is written into storage and decrypting that data as it is pulled from storage for use. Using a symmetric encryption key when the data is written into storage to protect it from unauthorized access by anyone who does not have that key to decrypt the data from storage into use. It should be used anytime data/information is of any level of sensitivity and would cause harm if accessed by unauthorized actors. |
| Encryption at flight | Encryption of data in-flight is the process of securely encoding data as it is being transmitted in some fashion. Dependent on exactly how you will be transferring any data will determine how to apply encryption. When using a web browser always utilize secure protocols, when sending emails always encrypt before sending and use digital signatures. |
| Encryption in use | Encryption of data in-use is the process of protecting data as it is utilized in memory, the main way of doing this is by utilizing password protected profiles as they protect the memory of each user for the data stored in memory for that profile could be used to compromise their data in rest/flight. |

|  |  |
| --- | --- |
| **b. Triple-A**  **Framework**  **\*** | **Explain what it is and how and why the policy applies.** |
| Authentication | The process used to prove who a user is by, userID, passwords, possibly higher level security such as secure tokens, CAC/PIN and other hardware credentials. |
| Authorization | Once a user is authenticated, and allowed access to a system they are granted specific access to parts of that system. Authorized access to certain drives, folders, programs, or data allowed by the system administrators. |
| Accounting | After authentication and authorization it is always a good idea to monitor and record |

|  |  |
| --- | --- |
| **b. Triple-A**  **Framework**  **\*** | **Explain what it is and how and why the policy applies.** |
|  | activity of all users on the system. This process is called accounting, by ensuring this process is carried out you will have a clear picture of who is attempting to access a system and what exactly they are doing with that access when they are granted authorization to that data on the system. |

**\***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

 4- a simple solution that is not needlessly complicated

 5- by default all accounts are denied until given access

 6- prevent creeping privileges

 8- part of the whole but not alone

 10- it is part of the process

* Firewall logs

 4- a simple solution that is not needlessly complicated

 5- by default all accounts are denied until given access

 6- prevent creeping privileges

 7- prevent unnecessary data from being transmitted

 8- part of the whole but not alone

 10- it is part of the process

* Anti-malware logs

 4- a simple solution that is not needlessly complicated

 5- by default all accounts are denied until given access

 6- prevent creeping privileges

 7- prevent unnecessary data from being transmitted

 8- part of the whole but not alone

 10- it is part of the process



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 |  |
| 1.11 | 4/11/2021 | Project One | Daniel Ostrin | Daniel Ostrin |
| [Insert  text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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