Green Pace Secure Development Policy

Contents

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.

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	Eliminate as many vulnerabilities as possible. In addition, validate incoming data from untrusted source and even trusted source to verify malicious data.
2. Heed Compiler Warnings	Compiling the code frequently. Also, testing the code should be priority as well because we want the code to have as little leaks as possible. Iterating through code frequently will allow us to keep in the loop of knowing if the code actually works.
3. Architect and Design for Security Policies	Design, test, and implement code with security as a top priority. In addition, there should be an emphasis on meeting the coding standards and making sure the code is doing what its supposed to.
4. Keep It Simple	Complexity is not always the best answer. If given the chance, keeping code simple with good indentations, and code comments can be productive. Others will read your code so keeping it cleans and simple can save a lot of time.
5. Default Deny	Denial as a standard is a great idea as it prevents anyone who should not have access to something from gaining access, while at the same time ensuring those who 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 accesses, it will be implemented that only the access that is truly needed and in return, will limit the access granted.
7. Sanitize Data Sent to Other Systems	Sending code with such information could lead to exploits which could have any number of terrible outcomes depending on the situation. Ensuring data sent is sanitized is crucial.
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.

Principles	Write a short paragraph explaining each of the 10 principles of security.
9. Use Effective Quality Assurance Techniques	Quality Assurance is a part of every process. Better to have someone that is on the team that finds 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 with other principles such as designing code with security policies in effect with quality assurance techniques. Having these ideas in as much as possible is a way to save time and prevent problems from later the developmental cycle.

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>::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): [Name the principle and explain how it maps to this standard.]

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Probable	medium	P12	L1

Automation

Tool	Version	Checker	Description Tool
Astree	20.10	function-ellipsis	Fully checked
Axivion Bauhaus Suite	6.9.0	CertC++DCL50	
Clang	3.9	cert-dc150-cpp	Checked by clang-tidy
CodeSonar	5.4p0	LANG.STRUCT. ELLIPSIS	[Insert text.]

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): [Name the principle and explain how it maps to this standard.]

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Unlikely	Low	P3	L3

Automation

Tool	Version	Checker	Description Tool
Astree		reserved-identifier	partially checked
Axivion Bauhaus Suite	20.10	CertC++-DCL51	

Tool	Version	Checker	Description Tool
Clang	6.9.0	-Wreserved-id-macro -Wuser-defined-literals	W-reserved-id-macro flag is not enabled by default. This flag does not catch all instances
[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]

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): [Name the principle and explain how it maps to this standard.]

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Unlikely	Low	P3	L3

Automation

Tool	Version	Checker	Description Tool
Axivion Bauhaus Suite	6.9.0	CERTC++-DCL52	
Parasoft C/C++ +test	2020.2	CERT_CPP-DCL52-a	Never qualify a reference type with 'const' or 'volatile'
Polyspace Bug Finder	R2020a	CERT C++: DCL52-CPP	Checks for: * Const-qualified reference * Modification of const-qualified reference types
PRQA QA C++	4.4	014	

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): [Name the principle and explain how it maps to this standard.]

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Unlikely	Medium	P2	L3

Automation

Tool	Version	Checker	Description Tool
LDRA tool Suite	9.7.1	296 S	Partially implement
Parasoft C/C++ +test	2020.2	CERT_CPP-DCL53-a CCERT_CPP-DCL53-b	Always declare functions at file Identifier declared in local or function protype.
Polyspace Bug Finder	[Insert text.]	CERT C++: DCL53-CPP	Check for declarations: * Function and object declaration * unnamed object or function
PRQA QA C++	[Insert text.]	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.

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): [Name the principle and explain how it maps to this standard.]

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Probable	Low	P6	L2

Automation

Tool	Version	Checker	Description Tool
Astree	20.10	new-delete-pairwise	Partially checked
Clang	3.9	misc-new-delete-overloads	Checked with clang-tidy
Parasoft C/C++ +test	2020.2	CERT_CPP-DCL54-a	Always provide delete and new
Polyspace Bug Finder	R2020a	CERT C++: DCL54-CPP	Check for mismatch between overloaded operator new and operator delete.

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.

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
#include

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));
```

Compliant Code

```
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): [Name the principle and explain how it maps to this standard.]

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Unlikely	High	P1	L3

Automation

Tool	Version	Checker	Description Tool
Axivion Bauhaus Suite	6.9.0	CertC++-DCL55	
Parasoft C/C++ +test	2020.2	CERT_CPP-DCL55-a	A pointer to a structure ought to not be passed to a function.
[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]
[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]

Coding Standard 7

Coding Standard	Label	Name of Standard
Exceptions	[STD-007-CPP]	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): [Name the principle and explain how it maps to this standard.]

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Unlikely	Medium	P2	L3

Automation

Tool	Version	Checker	Description Tool
LDRA tool suite	9.7.1	6 D	Enhanced Enforcement
Parasoft C/C++ +test	2020.2	CERT_CPP-DCL56-a	Avoid initialization order problems across translation units by replacing non-local static variables with local static variables.
[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]
[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]

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

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): [Name the principle and explain how it maps to this standard.]

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Likely	Medium	P6	L2

Automation

Tool	Version	Checker	Description Tool
Astree	20.10	destructor-without-noexcept delete-without-noexcept	Fully checked
Axivion Bauhaus Suite	6.9.0	CertC++-DCL57	
LDRA tool suite	9.7.1	453 S	Implemented partially
Parasoft C/C++ +test	2020.2	CERT_CPP-DCL57-a CERT_CPP-DCL57-b	No exception to be thrown from a destructor

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): [Name the principle and explain how it maps to this standard.]

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]	[Insert text.]

Automation

Tool	Version	Checker	Description Tool
Axivion Bauhaus suite	6.9.0	CertC++-DCL58	
Parasoft C/C++ +test	2020.2	CERT_CPP-DCL58-a	Don't modify the standard namespace 'std'
Polyspace Bug Finder	R2020.2	CERT C++: DCL58-CPP	This checks for modification of standard namespaces
PRQA QA C++	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.

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();
}
```

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): [Name the principle and explain how it maps to this standard.]

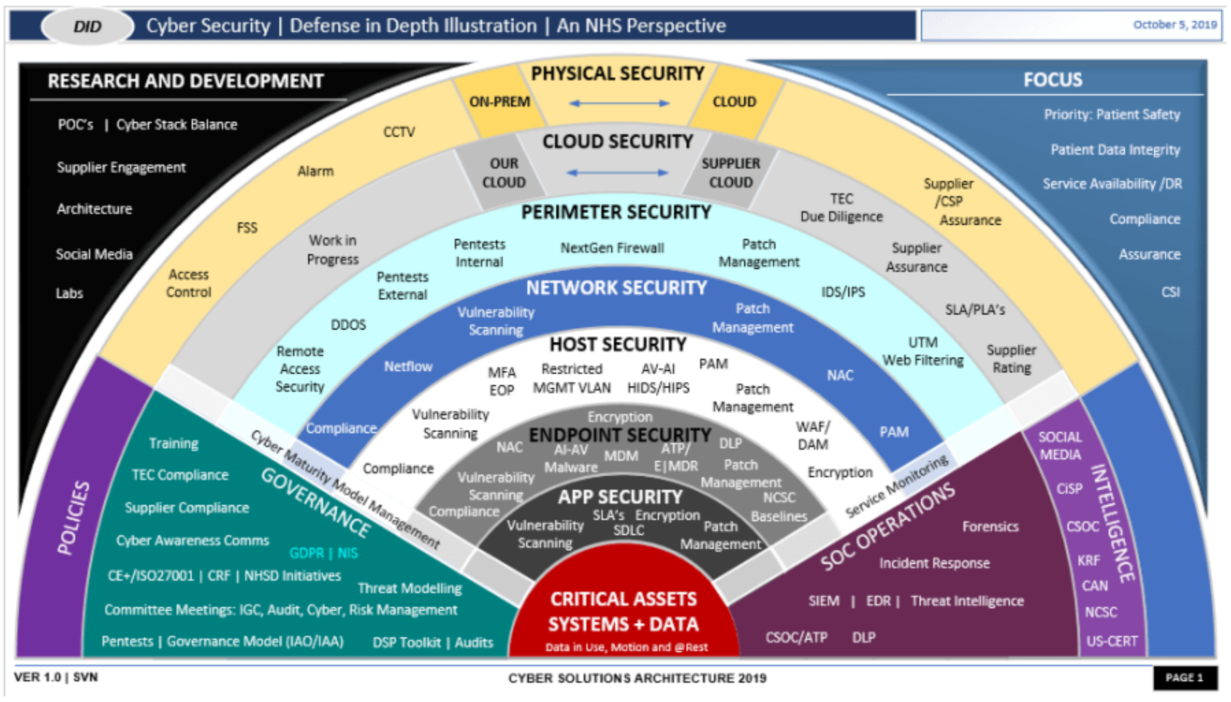
Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Medium	Unlikely	Medium	P4	L3

Automation

Tool	Version	Checker	Description Tool
Astree	20.10	unnamed-namespace-header	Fully checked
Axivion Bauhaus Suite	6.9.0	CertC++DCL59	
Clang	3.9	CERT-DCL59-CPP	Checks by clang-tidy
LDRA tool suite	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

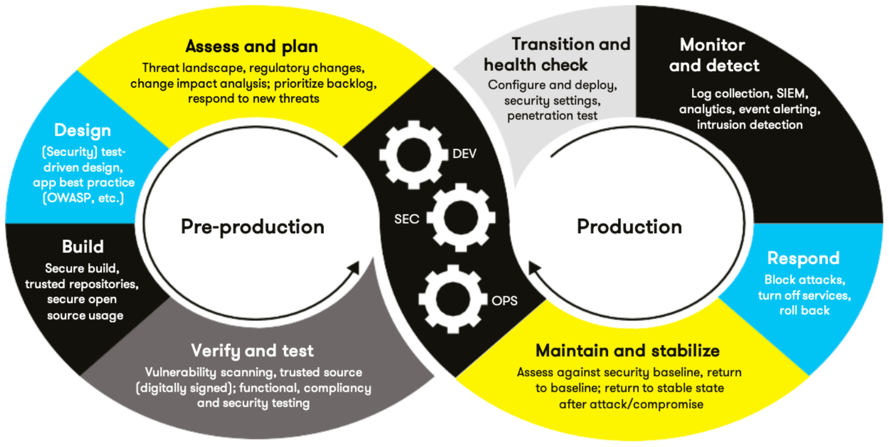
Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority,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


Assess and plan
Threat landscape, regulatory changes, change impact analysis; prioritize backlog, respond to new threats
Design
(Security) test-driven design, app best practice (OWASP, etc.)
Build
Secure build, trusted repositories, secure open source usage
Verify and test
Vulnerability scanning, trusted source (digitally signed); functional, compliancy and security testing
Pre-production
DEV
SEC
OPS
Transition and health check
Configure and deploy, security settings, penetration test
Monitor and detect
Log collection, SIEM, analytics, event alerting, intrusion detection
Production
Respond
Block attacks, turn off services, roll back
Maintain and stabilize
Assess against security baseline, return to baseline; return to stable state after attack/compromise

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.

[Insert your written explanations here.]

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
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.

a. Explain each type of encryption, how it is used, and why and when the policy applies.
b. 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. Encryption	**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.
Encryption at flight	Encryption of data in-flight is the process of securely encoding data as it is being transmitted in some fashion.
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 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
- Firewall logs
- Anti-malware logs

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

Audit Controls and Management

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

Evidence will include the following:

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

Enforcement

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

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

Exceptions Process

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

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

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

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

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

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

Policy Version History

Version	Date	Description	Edited By	Approved By
1.0	08/05/2020	Initial Template	David Buksbaum	
[1.11]	4/08.2022	Project One	Nijaz Kovacevic	Nijaz Kovacevic
[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